Frequent Pattern-based Map-matching on low sampling rate trajectories

Yukun Huang[§] Weixiong Rao[§] Zhiqiang Zhang[§] Peng Zhao[§] Mingxuan Yuan[†] Jia Zeng[†]
[§]School of Software Engineering, Tongji University, China [†]Huawei Noah's Ark Lab, Hong Kong
wxrao@tongji.edu.cn {yuan.mingxuan, zeng.jia}@huawei.com

***Abstract*—Map-matching is an important preprocessing task for many location-based services (LBS). It projects each GPS point in trajectory data onto digital maps. The state of art work typically employed the Hidden Markov model (HMM) by shortest path computation. Such shortest path computation may not work very well for very low sampling rate trajectory data, leading to low matching precision and high running time. To solve this problem, this paper, we first identify the frequent patterns from historical trajectory data and next perform the map matching for higher precision and faster running time. Since the identified frequent patterns indicate the mobility behaviours for the majority of trajectories, the map matching thus has chance to satisfy the matching precision with high confidence. Moreover, the proposed FP-forest structure can greatly speedup the lookup of frequent paths and lead to high computation efficiency. Our experiments on real world data set validate that the proposed FP-matching outperforms state of arts in terms of effectiveness and efficiency.**

I. INTRODUCTION

Recent years witness the proliferation of location-based services (LBS), such as Uber, Wechat and Baidu Map on mobile devices (MDs). Such services use GPS sensors to record life trajectory of humans and generate massive trajectory data. Many applications, such as urban planning and intelligent transportation, have used the trajectory data to understand peoples mobility patterns.

To enable the applications above, an important preprocessing task is to perform an map-matching step, such that each GPS point in the trajectory data is projected onto digital maps. For example, trajectory data frequently contains many noisy data, e.g., GPS position errors [12], which could deviate from true position. The literature has proposed dozens of map-matching algorithms [13], [15], [10], [11], [3], [4], [18].

Nevertheless, when the trajectory data contains very low sampling rate of GPS points, the design of an accurate map-matching algorithm is rather challenging [8], [9]. For example, people often switch off GPS sensors for energy saving or some private reasons. Consequently, the sampling rate is low and the GPS points are very sparse. The state of art essentially adopted Hidden Markov Model (HMM) and its variation [10], [11] to perform map-matching for low-sampling-rate GPS trajectories. The fundamental assumption of such approaches is that the path between two GPS points prefers to be the shortest path of a certain metric. Unfortunately, such an assumption may not work well in real life, leading to low matching precision. We take Figure 1 for illustration. In this figure, blue box and purple box represent the source point S and destination point D of a car moving trajectory, respectively. Green line is the shortest path in terms of source-destination nearest distance, and red line is the true route between these two points, indicating that the car driver does not follow the shortest path with the nearest distance. Instead, the driver first follows the most familiar route from S to A, and next drives on the high speed road from A to B and finally arrives at D with the least driving time. Among the entire path of this read line from S to D, all the three subpaths are with various metrics. Thus, it is hard to use the shortest path computation by a single certain metric to perform map-matching. In addition, the computation for shortest paths required by the HMM-based approaches is inefficient. How to perform an map-matching algorithm highly efficiently is important, too.

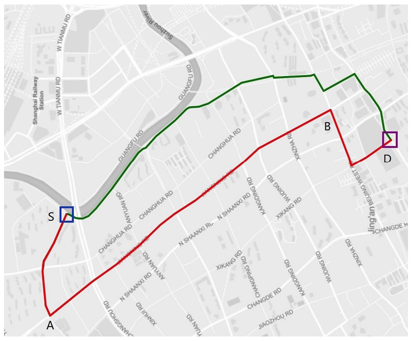

Fig. 1: Observation

To optimize the map-matching precision for low sampling rate trajectories, in this paper, we propose a frequent pattern (FP)-based method. Our basic idea is first to identify frequent patterns (FPs) from a third-party historical trajectory data. Next to perform a map-matching of a low sampling rate trajectory, we find those FPs which most match a trajectory and link such found FPs as the map-matching result. Such patterns indicate the mobility behaviours of the majority of driving behaviours among historical trajectories: each of the PFs is with either the most familiar route or most fastest route or most time saving routes or others. Thus, our map-matching result of a trajectory could contain multiple FPs and mix various metrics, depending upon such FPs. As a result, it makes sense to use the identified patterns to truly match personalized driving behaviors for higher map-matching precision. In particular, nowadays almost all modern smart phones and vehicles are equipped with GPS sensors to record moving trajectories and there are many publicly available third-party trajectories, e.g., those provided by Openstreet Map. Such trajectories could contain high sampling rate trajectories. When the patterns mined from such high sampling rate trajectories are available, it is highly possible to design an accurate map-matching algorithm.

Beyond the precision, the FPs can benefit the computation efficiency during map-matching. Specifically, when the frequent patterns are identified, we design an FP-forest structure to index the identified FPs. FP-forest offers fast query response for map-matching, such that the proposed FP-matching algorithm can find the global optimum results very quickly. In summary, this paper makes the following contributions:

1) We propose a novel map matching algorithm called FP-matching personalized for the majority of driving behaviours. Differing from the state-of-art work using a certain metric, our work explores frequent patterns (mixed with various metrics) to effectively and efficiently map low sampling rate trajectories onto digital map and complement the route between the matching road segments.
2) We build an FP-forest index to quickly respond to map-matching queries in order to transform raw GPS sequence to road segment sequence. Thus, the proposed FP-matching approach can achieve both meaningful matching effectiveness and high computation efficiency.
3) We evaluate the effectiveness and efficiency of the proposed FP-matching approaches against state-of-art methods. Experimental results show that our method outperforms such methods in terms of both matching accuracy and running time.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 gives solution overview. Section 4 describes FP-forest and Section 5 presents the FP-matching algorithm. After that, Section 6 evaluates FP-matching and finally Section 7 concludes the paper.

II. RELATED WORK

In this section, we investigate related work in terms of map-matching and frequent pattern.

A. Map Matching

The problem of map matching is to locate a sequence of GPS points (consisting of timestamp, longitude and latitude) onto a digital map. In literature the map matching algorithms can be broadly classified into three categories [14].

The *1st* category is the incremental approach [13], [15]. First, [13] predicts the road segment of a GPS point based on its previous prediction. This algorithm takes into account 1) the project distance between current GPS point to candidates and 2) the difference between the GPS heading and direction of candidate road segment, in order to decide which road segment should be mapped to the current GPS point. [15] evaluates four incremental algorithms and indicates that all such algorithms are with low accuracy. The main reason is that incremental methods select a best candidate for each sample at the current time stamp based on a small range of recent samples. However, such incremental methods can work well in an online manner.

The *2nd* category is the global approach [10], [11]. Such an approach is similar to the *1st* one by the search for candidate set for each GPS sample and next computing a weight for each candidate. Nevertheless, the global one considers the whole trajectory and calculates the aggregated weight of an entire candidate sequence to find out the maximum one. Both [10] and [11] employ the HMM model and next follow a normal distribution of project distance between GPS points and road candidates to calculate emission probability. [10] integrates the spatial function and temporal function together to compute the transition probability, while [11] adopts an exponential function of route distance and straight-line distance to compute transition probability. Because they both need to compute the shortest distance between candidate points. Thus the computation of transition probability incurs very high running time.

The *3rd* category is the geometrical approach [3], [4]. Such an approach frequently needs to search an optimal path on a digital map by geometric similarity measures, e.g., Frechet distance [5], and does not find a candidate set for each GPS point. [3], [4] search the path with the minimum Frechet distance to the GPS sequence.

In general, the proposed FP-matching approach belongs to a global method. Since [10], [11], [18] outperform other map matching algorithms at low sampling rate, we therefore evaluate our method against these three algorithms. Here, we briefly highlight the difference between our work and two HMM algorithms [10], [11], [18] as follows. First, our method uses the frequency of historical trajectories on at least two continuous road segments to replace the transition probability of two connected road segments, and employs a dynamic programming algorithm to map the GPS points. Second, we adopt the FP-forest structure to find the most frequent routes between two mapped road segments, and combine all the routes to complement the whole trajectory. The FP-matching method avoids the shortest distance computation, leading to trivial running time.

Note that a recent work, the history based route inference system (HRIS) [18], performs map-matching with help of reference trajectory. Instead the proposed work depends upon the identified FPs, which essentially indicates the mobility patterns of those successfully map-matched historical trajectories onto road networks. Since the identified FPs are with map-matching result and HRIS is still with raw trajectories, our work thus outperforms HRIS in terms of accuracy and efficiency.

B. Frequent Pattern

Frequent pattern plays an important role in many data mining tasks to solve practical problems [6]. The classic algorithms of frequent patterns are Apriori [1] and FP-growth [7]. FP-growth solves high cost of Apriori on time and space efficiency. In this paper, the proposed FP-matching employs FP-tree to index the matching result of historical trajectories. In the FP-tree, its root is the starting point, and any path is a history trajectory. In addition, GSP [2] and SPADE [17] extend the Apriori algorithm into sequential FP problem.

III. Solution Overview

In this section, we first give the problem definition and next highlight our solution.

Definition 1 *A trajectory $T = \langle (p_1, t_1) \dots (p_n, t_n) \rangle$ consists of a sequence of GPS samples. Each sample contains a GPS position p_i and a timestamp t_i. The number n means the length of the trajectory. The position p_i is a GPS longitude and latitude pair.*

Definition 2 *A digital map is a directed graph $\mathcal{M} = (\mathcal{V}, \mathcal{S})$, in which $\mathcal{V}$ is a set of intersection or termination points on road segments, and $\mathcal{S}$ is a set of road segment edges between two points of V.*

Definition 3 *The sampling rate refers to the average time interval to record GPS samples. The sampling rate of a trajectory is calculated by the formula $\frac{\sum_{i=2}^{n} t_i - t_{i-1}}{n-1}$.*

Definition 4 *A route is a sequence of connected road segments, i.e., $S =< s_1 \dots s_n >$, where the start point of road segment s_{i+1} is the end point of road segment s_i.*

Definition 5 *Map matching is to project the GPS points in a trajectory T of low sampling rate onto to the road segments in a digital map $\mathcal{M}$. Specifically, given a trajectory T, we want to generate a route of connected road segments $\langle s_1 \dots s_{n'} \rangle)$, such that the trajectory T is most likely moving on the route. The number n' is the size of recovered route.*

Due to very low sampling rate, the very sparse GPS points in trajectory T could be projected to disconnected road segments. Such disconnected road segments indicate no connectivity between source and destination segments road segments. Thus, we need to find a path in the digital map $\mathcal{M}$ to complement such disconnected road segments. In this way, we recover the entire route of connected road segments.

To match a very sparse trajectory T, our general idea is to leverage a third-party historical trajectory database that could contain high sampling rate trajectories. Such historical trajectory database is available nowadays, for example generated by taxi and other mobile devices. We use the historical data to optimize the map-matching of an input trajectory.

Problem 1 *Given a historical trajectory database D containing $|D|$ high sampling rate trajectories $T_1 \dots T_{|D|}$, we want to identify a set of meaningful trajectory patterns from database D. With help of the patterns, we want to generate a most likely route for a very sparse input trajectory T onto the map $\mathcal{M}$.*

As shown in Figure 2, in the *offline* phase, we use the history trajectory database to maintain an FP-forest structure. That is, for those high sampling rate trajectories in the database, we perform a classic map-matching algorithm to find the sequence of matching road segments to which GPS points in the trajectories are projected. Next, we explore a step to extract subsequences from the sequences of matching road segments. All such sequences (and subsequences) are then indexed by an FP-forest index.

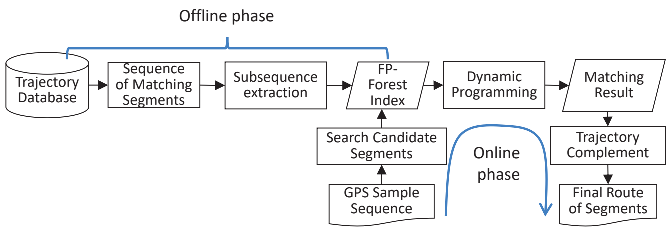

Fig. 2: System Overview

Next in the *online* phase, when given a trajectory of very sparse sampling GPS points, we first search digital maps to find the nearest road segment candidates. With help of the maintained FP-forest structure, we find the FPs containing such candidates and next employ the dynamic programming algorithm to best matching path for the trajectory. The purpose of dynamic programming is to select the best candidate for each GPS sample depending on the pattern frequency (which is achieved from FP-Forest). Finally, in case that the best matching path contains disconnected road segments, we then complement the segments to recover the entire route.

Intuitively, the offline phase is used to maintain the FP-forest; and the online phase is to evaluate the input trajectory T against the FP-forest to generate the most likely route of connected road segments.

IV. FP-Forest

In this section, we first introduce the FP-forest structure, next describe the creation of FP-forest, and finally show the lookup operation.

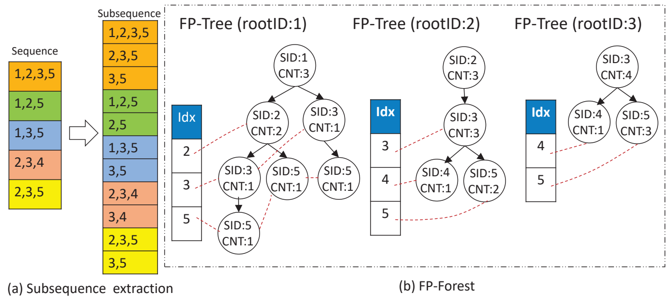

Fig. 3: FP-forest Example

A. FP-Forest Structure

The FP-forest consists of multiple FP-trees rooted at the associated segment IDs. The structure of FP-tree is similar to the tree in FP-growth whose root is null. Each node inside an FP-tree contains two items: (1) road segment ID (SID) and (2) counter (CNT) of trajectories such that the segment is one matching segment of such trajectories. The counter records the frequency of such trajectories matched to the road segment. For each FP-tree, we have an associated dictionary. Each key in the dictionary is a segment ID (SID), pointing to a list of tree nodes whose segment IDs are just equal to the key.

As shown in Figure 3, the FP-forest contains three FP-trees, rooted at the SIDs 1, 2 and 3, respectively. For the FP-tree rooted at SID 1, the associated dictionary maintains three keys 2, 3 and 5. Each key points to a list of internal nodes in the FP-tree. For example, the key of 3 points to the list of two nodes with the pairs $\langle 3, 1\rangle$; and the key of 5 points to the list of three nodes with the pairs $\langle 5, 1\rangle$.

B. FP-forest Creation

Given the historical database D, we could find the those high sampling trajectories T_i. By matching such trajectories T_i onto the map $\mathcal{M}$ by the classic map matching algorithms [10], [11], we have the sequences of matching road segments. Specifically, when a GPS point in T_i is mapped to some real position on a road segment $s_j \in \mathcal{M}$, we then have a sequence of distinct segments s_j with respect to (w.r.t) T_j. After that, we employ the FP-forest structure in Figure 3 to index all road segment sequences w.r.t the high sampling trajectories T_i.

Now we give the detail to create FP-Forest by two steps: 1) *subsequence extraction* from each road segment sequence after map-matching each high sampling trajectory $Ti \in D$. 2) *subsequence insertion* into the FP-forest structure. We will show that the subsequence extraction enables fast lookup of FP-forest very soon.

(Sub)Sequence extraction: Given a sequence of road segments $S = s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k$, we extract the following $(k-1)$ subsequences: $\{s_2 \rightarrow s_3 \rightarrow \ldots \rightarrow s_k\}$, $\{s_3 \rightarrow \ldots \rightarrow s_k\}$...$\{s_{k-1} \rightarrow s_k\}$. Together with the original sequence S, this step totally generates k subsequences. Figure 3 gives the 11 subsequences extracted from 5 sequences.

Subsequence insertion: Given an FP-forest structure, we show how to insert a new (sub)sequence. Specifically, when given a sequence of road segments $S = s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k$, we first check whether or not the FP-forest contains a tree whose root is just equal to the segment ID s_1. If not, we then build a new FP-tree by taking s_1 as the root ID, and meanwhile set the root counter to be 1. Otherwise, if yes, we increment the counter of this root by 1.

After that, we traverse the FP-tree with the rest of sequence S, i.e., $s_2 \rightarrow \ldots \rightarrow s_k$, by two operations. One is to update the counter of an existing node in this tree, another is to create a new node which becomes a child of one existing node. For example, consider a sequence $...s_i \rightarrow s_{i+1}...$ and suppose that we have just visited an existing node whose segment ID is s_i. Next we need to visit the segment s_{i+1}. First if the node s_i contains a child with segment ID s_{i+1}, then the counter of such an existing child s_{i+1} will be added by 1. Otherwise we will create a new child of node s_i with segment ID s_{i+1} and counter 1.

Following the idea above, Algorithm 1 gives the details of creating an FP-forest. In this algorithm, for each trajectory of the input database D, it contains a sequence $S \in D$ of connected road segments $S = \{s_1 ... s_{|S|}\}$ after performing a classic map-matching algorithm such as [10], [11]. In addition, Algorithm 2 finds a specific FP-tree in FP-Forest, and Algorithm 3 finds the child of a given node.

Complexity Analysis: Assume that each trajectory sequence $S \in D$ contains at most k road segments. The generated subsequences is thus smaller than k, leading to $O(k)$ operations (i.e., **findTree**). Thus, the running time of Algorithm 1 is $O(|D| \times k)$, where $|D|$ is the amount of trajectories in D.

Algorithm 1: **CreateForest**(Trajectory Database D)

```
Output: FP-model F
1  $F = \Phi$;
2  foreach $S \in D$ do
3      TREE $t \leftarrow$ FindTree($F$, $S.s_1$); NODE $r$ = null ;
4      if $t != null$ then {$r = t$.root; $r$.counter++ };
5      else $r$.SID = $S.s_1$; $r$.counter = 1; $t$.root = $r$; add $t$ to $F$;
6      NODE $curNode = r$;
7      for $i = 2; i \le |S|; i++$ do
8          NODE $n \leftarrow$ FindChild($curNode$, $s_i$);
9          if $n != null$ then {$curNode = n$; $n$.counter++};
10         else
11             $n$.counter = 1;
12             $n$.parent = $curNode$; $curNode$.addChild($n$);
13             $t$.addIndex($n$); $curNode = n$;
14 return $F$;
```

Algorithm 2: **FindTree** (FOREST F, SID s)

```
Output: return the tree in $F$ whose root has the SID $s$
1  if $F$ contains a tree whose root has the segment ID $s$ then return the tree in $F$
   whose root is the SID $s$;
2  else return NULL;
```

Algorithm 3: **FindChild** (NODE $curNode$, SID s)

```
Output: return the node having the child with $SID == s$
1  if $curNode$ has a child whose segmentId is $s$ then return the child of $curNode$;
2  else return NULL;
```

Algorithm 4: **GetFreq** (FOREST F, SID sid, SID eid)

```
Output: Frequency f
1  $f = 0$; TREE $t \leftarrow$ findTree($F$, $sid$);
2  for NODE $n$ in $t.index(eid)$ do $f \leftarrow f + n$.counter;
3  return $f$;
```

C. FP-Forest Lookup

In this section, we introduce two lookup operations on FP-forest: frequency lookup and trajectory Complement.

Frequency Lookup Suppose that we need to find the counter (frequency) of trajectories passing from road segment *sid* to road segment *eid*. To this end, we first find the FP-tree whose root has the SID equal to *sid*. After that, with help of the dictionary, we find the list of tree nodes with the segment IDs equal to *eid*. Finally, for each found tree node, if there exists a path from the root to the tree node, we then add the counter of such a tree node to the returned frequency.

We give an example to show the frequency lookup from *sid* = 1 to *eid* = 5. First we lookup the FP-Tree with the root ID 1. In the FP-tree, we then make sure that the associated dictionary contains the key 5. The key t contains the list of three internal nodes with the pairs $\langle 5, 1\rangle$. Then we check whether or not there exists a path from the root 1 to each of such nodes 5. Since all such paths are found, we then sum the counters of such nodes t and return the frequency 3.

It is not hard to find that the frequency lookup is very *fast*. It is mainly because we only need to check the connectivity of the path from the root *sid* to each internal node with segment ID *eid* with no further pruning. Note that the fast lookup is enabled by the aforementioned subsequence extraction: each extracted subsequence starting from *sid* is inserted to the FP-tree rooted at *sid*. Thus, the lookup only needs to check the connectivity from root *sid* to the node *eid*. On the overall, the time complexity of the lookup operation is $O(|Tree|)$, where $|Tree|$ denotes the amount of nodes in an FP-Tree.

Trajectory Complement When given two disconnected road segments, we need to complement a route between the two segments, such as the road segments in the entire route are all connected. The main idea is to select the most frequent route. To this end, we first find the FP-tree whose root has the segment id *sid*. Then by checking the dictionary, we find the list of tree nodes whose segment IDs are *eid*. Based on the found tree nodes, we then find the associated paths from the root to the tree nodes. Depending upon the counters, we can rank the paths by the counters and return the path with the highest counter. Algorithm 4 describes the detail to complement the route of two disconnected segments again with the time complexity $O(|Tree|)$.

V. FP-Matching Algorithm

In this section, we will present the detail of the proposed FP-matching algorithm with help of FP-Forest. In general, the matching algorithm contains two steps: 1) candidate search, and 2) finding a best path by dynamic programming.

A. Searching Candidate

Given a sequence of GPS points $\mathcal{P} = \{p_1 \rightarrow \ldots \rightarrow p_n\}$, we search a set of candidate road segments w.r.t each point p_i within a certain radius r $(1 \leq i \leq n)$. In Figure 4(a), the GPS point p_i is with three road segments S_i^1, S_i^2, S_i^3 within the search radius r. Then we compute the distance between p_i and the candidate S_i^j $(1 \leq j \leq 3)$ by $dist(p_i, S_i^j) = min_{\forall c \in S_i^j} dist(p_i, c)$ where c is a point within segment S_i^j. In Figure 4(a), the point c_i^j is the point inside S_i^j nearest to p_i.

To find candidate segments efficiently, we firstly build a R-Tree index for the whole digital map. The search for the top-k nearest road segments to a given point p_i involves two steps. First we find those a set of candidate minimal boundary rectangles (MBR) nearest to p_i. After that, we scan those road segments inside (or intersected by) such MBRs. By exploring more MBRs and road segments, we could finally select the top-k nearest road segments as candidates.

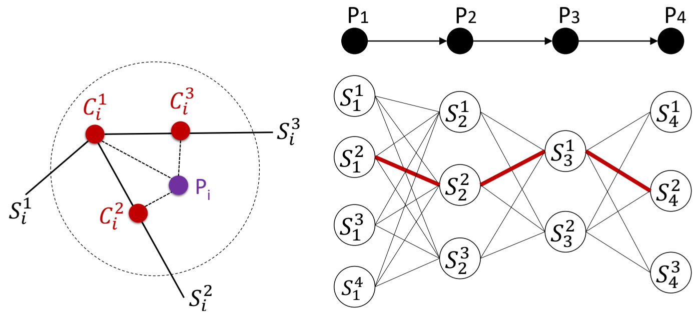

Fig. 4: (a) Candidate Search (b) Finding Best Path

B. Dynamic Programming

When the candidate sets w.r.t a sequence of GPS points are found as mentioned above, we now search a path of road segments which can best match such points.

As shown in Figure 4(b), the solid black circles indicate the raw GPS points $p_1 \ldots p_4$, and the hollow black circles under each solid one indicate the candidate road segments which are nearest to each GPS point p_i. S_i^j denotes the j-th candidate road segment of point p_i. Now let us consider two continuous GPS points p_i and p_{i+1} and associated candidate road segments S_i^j and $S_{i+1}^{j'}$. Here, the two candidate road segments S_i^j and $S_{i+1}^{j'}$ might be directly connected or disconnected on the road network map. If the two segments are disconnected, we could follow Section IV-C to complement them by frequent patterns. In case that no frequent patterns are available to complement the disconnected road segments, we still could complement them by the shortest path in terms of their associated distance. After that, we connect the segments by an edge and define the edge weight as follows.

Specifically, the edge weight between two candidate road segments (say S_i^j and $S_{i+1}^{j'}$) is decided by 3 factors. First, the

Algorithm 5: **ComplementRoute** (Forest F, SID *sid*, SID *eid*)

```
    Output: Path p
1   $f = 0$; Tree $t \leftarrow$ findTree($F$, $sid$); path $p$ = [];
2   Node $curNode$ ;
3   for Node $n$ in $t.index(eid)$ do
4       if $f \leq n.counter$ then  $f = n$.counter; $curNode = n$;
5   while $curNode \neq NULL$ do
6       $p$.append($curNode$.SID); $curNode \leftarrow curNode$.parent();
7   return $p.reverse()$;
```

frequency between S_i^j and $S_{i+1}^{j'}$ which can be found by the lookup operation (See Section IV-C). Second, the distance between the raw GPS points p_i and associated candidate road segment S_i^j (resp. the distance between p_{i+1} and $S_{i+1}^{j'}$). Following the previous work [10], we assume that the distance between a GPS point and its real position follows a normal distribution $N(\mu, \sigma^2)$. Thus, we define the effect of two candidates on edge weight:

$$W(S_i^j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[Dist(i,j)-\mu]^2}{2\sigma^2}} \tag{1}$$

where $Dist(i, j) = Dist(S_i^j, p_i)$ denotes the distance between p_i and candidate road segment S_i^j. Consider that the path $S_i^j \rightarrow S_{i+1}^{j'}$ involves two candidate segments S_i^j and $S_{i+1}^{j'}$. Thus, we need a parameter $\beta \geq 0$ to mediate the factors w.r.t S_i^j and $S_{i'}^{j'}$ and define the following weight.

$$Wight(S_i^j, S_{i+1}^{j'}) = \frac{(1+\beta) * W(S_i^j) * W(S_{i+1}^{j'})}{\beta * W(S_i^j) + W(S_{i+1}^{j'})} \tag{2}$$

where β indicates the relative importance of S_i^j over $S_{i'}^{j'}$. When β is equal to 1, the two candidates S_i^j and $S_{i'}^{j'}$ have the same influence on the path between them. If β is smaller than 1, the candidate S_i^j has higher weight on the edge and otherwise if β is more than 1, the candidate $S_{i'}^{j'}$ has higher weight.

Finally, We denote $W(S_i^j \rightarrow S_{i'}^{j'})$ to be the weight of the edge between two candidates S_i^j and $S_{i'}^{j'}$, and compute the following edge weight:

$$Weight(S_i^j \rightarrow S_{i+1}^{j'}) = Weight(S_i^j, S_{i+1}^{j'}) * Freq(S_i^j \rightarrow S_{i+1}^{j'})$$

where $Freq(S_i^j \rightarrow S_{i+1}^k)$ is the frequency of historical routes from S_i^j to S_{i+1}^k.

Until now, the FP-matching problem is to find a path with the largest weight, for example marked by the red line in Figure 4(b), such that the sum of all edge weights is maximized among all optional paths. Formally, the best matching path P of road segments is defined as:

$$P = \arg\max_{S_i^{best}} \sum_{i=1}^{n-1} Weight(S_i^{best_i} \rightarrow S_{i+1}^{best_{i+1}})$$

Our purpose is to find one path with the highest overall weigth in a directed acyclic graph (DAG). By the dynamic programming (DP) technique, Algorithm 6 shows the detail of finding the best path. After Algorithm 6 returns the matching road segment sequence, we check whether or not those segments inside the sequence are disconnected. If some of them are disconnected, we then have to employ the complement to recover the full path by Algorithm 5.

Time Complexity: Depending upon the input DAG, the running time of Algorithm 5 is $O(n * m^2)$, where m is the average number of candidate road segments in DAG.

Algorithm 6: **FindBestPath** (DAG: $(S_1^1 \dots S_1^k) \rightarrow \dots \rightarrow (S_n^1 \dots S_n^{k'})$, Forest F)

```
Output: Best Path $P$: $S_1^{best_1} \rightarrow \dots \rightarrow S_n^{best_n}$
Let f[] denote the highest frequency computed so far;
Let pre[] denote the previous of current candidate;
for $i = 1; i \leq k; i++$ do  $f[S_1^i] = 0$;
for $i = 2; i \leq n; i++$ do
    for $j = 1; j \leq k; j++$ do
        $max$ = -1;
        for $l = 1; l \leq k; l++$ do
            $temp = f[S_{i-1}^l]$ + getFreq($F$, $S_{i-1}^l$, $S_i^j$) * Weight($S_{i-1}^l$, $S_i^j$);
            if $max \leq temp$ then  $max = temp$; pre[$S_i^j$] = $S_{i-1}^l$;
            $f[S_i^j] = max$;
$P$ = []; $cur = argmax_{S_n^x}^{f[S_n^x]}$;
for $i = n; i \geq 2; i--$ do  { $P$.append($cur$); $cur$ = pre[$cur$]};
$P$.append($cur$); $P$.reverse();
return $P$;
```

VI. EXPERIMENTAL RESULTS

In this section, we first introduce experimental settings, including the used dataset, digital map, and key parameters, and next evaluate FP-Matching against three algorithms i.e., HMM-matching [11], ST-Matching [11] and HRIS [18].

A. Dataset

Road Network In our experiments, we use digital map of Shanghai extracted from Open Street Map. The network contains 146804 vertices and 95950 edges.

Real Taxi Data We use labeled true taxi trajectories in one day. There are total 92602 trajectories. Each GPS sample contains a timestamp, status of carrying passengers, GPS longitude and latitude coordinates, speed, and direction. We mainly use the GPS longitude and latitude coordinates to search candidate road segments. With help of the timestamp, we sample the GPS points to simulate various sampling rates.

Trajectory Division Based on the status of carrying passengers, we divide long trajectories of a taxi into multiple smaller subtrajectories, and each subtrajectory indicates one route of a passenger. Such division makes sense because the subtrajectories with passengers directly indicate the travel route from sources to destinations. Thus, we mainly use those divided subtrajectories to build FP-forest. In summary, we randomly take 155725 trajectories to build FP-forest and 337 trajectories to evaluate the matching approaches.

B. Performance metrics

We evaluate the proposed FP-matching algorithm in terms of matching accuracy and running time. The matching accuracy is measured using 1) A_N: Accuracy by Number and 2) A_L: Accuracy by Length.

$$A_N = \frac{\#correctly\,matched\,road\,segments}{\#all\,road\,segments\,of\,the\,trajectory}$$

$$A_L = \frac{\sum the\,length\,of\,matched\,road\,segments}{the\,length\,of\,the\,trajectory}$$

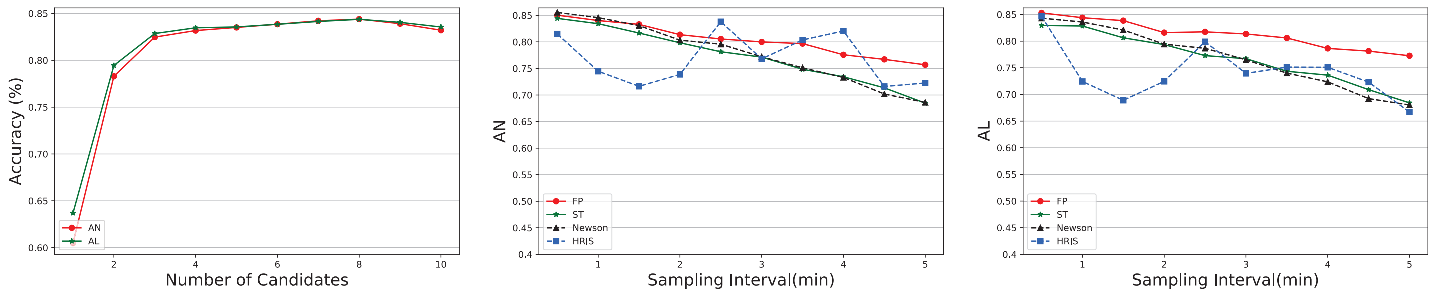

Fig. 5: From Left to Right: Effect of candidate segments, and sampling intervals

C. *Effect of candidate segments*

By varying the number of candidate segments, we measure the matching accuracy of FP-matching. As shown in Figure 5 (left), more candidates lead to higher accuracy for both A_N and A_L. When the number of candidate segments grows from 1 to 8, the accuracy grows dramatically at the number 3 and later remains relatively stable.

D. *Comparison of three algorithm*

Figure 5 (middle and right) shows the accuracy of FP-matching, ST-matching [11], HMM-matching [11] and HRIS [18] with various sampling rate. When the sampling interval becomes larger (i.e., a lower sampling rate), the accuracy of all algorithms becomes lower. It is easy to see from figures that FP-matching perform best in general. ST-matching and HMM-matching have similar the decrease trend. Note that after the interval of 3 minutes, the accuracy of FP-matching keeps smoothly lower. Such result indicates that FP-matching works very well when given very sparse GPS points.

Note that the curve of HRIS fluctuates severely. It is mainly because that HRIS works perfectly in case of smaller sampling interval than 2 min. At a relative high sampling rate, HRIS gets the worst accuracy compared to other methods.

E. *Parameter tuning of β*

In addition, we study the effect of parameter β under various sampling interval in Table I. Since the two accuracy metrics A_N and A_L share the similar trend, we thus use A_N for illustration. For a fixed β, a higher sampling interval consistently incurs lower accuracy. Yet for a fixed sampling interval, a smaller β on the overall leads to relatively higher accuracy. The behind reason is explored as follows. Taxi drivers decide the route mainly based on the current position. Thus, by the traffic situations around the current position, the drivers choose the next road segments towards the final destination. Therefore, it makes sense FP-matching works better for a small β which leads to a higher weight for the source segment in the path $S_i^j \rightarrow S_{i'}^{j'}$.

F. *Effect of Historical Data*

In this experiment, we vary the amount of historical trajectories from $3 * 10^4$ to $15 * 10^4$ and study the effect of matching accuracy in Figure 6 (left and middle). As shown in this figure, for a fixed sampling rate, more historical trajectories lead to higher accuracy in terms of both A_N and A_L. It is because more historical trajectories indicate more area coverage and stable frequent patterns.

G. *Efficiency*

Finally we evaluate the running time of four matching algorithms in Figure 6 (right). First of all, HRIS suffers from the highest running time no matter the sampling interval. Such result is consistent with the evaluation in [18]. In addition, FP-matching still outperforms the HMM-matching [11] and ST-matching [10] due to the expensive running time of shortest path computation required by HMM-matching [11] and ST-matching [10]. Note that the running time of all approaches become smaller for higher sampling intervals. That is, a larger sampling interval leads to a smaller amount of GPS samples. Thus, the running time of all approaches decreases.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we address the problem of map matching for low sampling rate trajectories, by mining the frequent patterns in huge amount of historical taxi routes. We come up with FP-matching method searching for matching result with highest sum of score, and the score is calculated by the frequency and influence of terminal road segments. The experiments results demonstrate that our FP-matching outperform typical HMM-based and historical based algorithm in terms of matching accuracy and running time.

In the future work, we plan to compress FP-forest for small space overhead. In addition, in case of history trajectories are very sparse and cannot cover the entire road map, we are going to design an effective approach to fill the missed trajectories [16]. Finally, we are interested in how to extend our algorithm in an online manner.

Acknowledgment

This work is partially supported by National Natural Science Foundation of China (Grant No. 61572365, 61503286) and sponsored by The Fundamental Research Funds for the Central Universities.

TABLE I: Accuracy (A_N) with various sampling interval and β

Interval (min) \ β	0.2	0.4	0.6	0.8	1	10	50	100	500
0.5	0.850055	0.849908	0.849626	0.849712	**0.850296**	0.846599	0.846268	0.846412	0.846238
1.0	0.840766	**0.841442**	0.840077	0.839996	0.840195	0.835229	0.832763	0.832457	0.832651
1.5	**0.834172**	0.83309	0.832415	0.832315	0.832974	0.826642	0.82434	0.82343	0.822548
2.0	0.814069	**0.815266**	0.815017	0.813064	0.81357	0.807908	0.804108	0.803813	0.80197
2.5	**0.807403**	0.80724	0.805885	0.805268	0.805494	0.800666	0.797992	0.797133	0.796604
3.0	**0.800012**	0.799681	0.799503	0.7991	0.799965	0.79674	0.794601	0.794296	0.793527
3.5	0.79655	0.797095	**0.79736**	0.796751	0.796778	0.793477	0.792043	0.791746	0.791362
4.0	0.774592	0.77569	**0.776299**	0.775794	0.775743	0.772782	0.769192	0.768666	0.767884
4.5	0.768395	0.76741	**0.768906**	0.768772	0.767098	0.762198	0.756612	0.756315	0.754372
5.0	**0.757598**	0.757363	0.756899	0.757259	0.75691	0.750908	0.746933	0.746611	0.744815

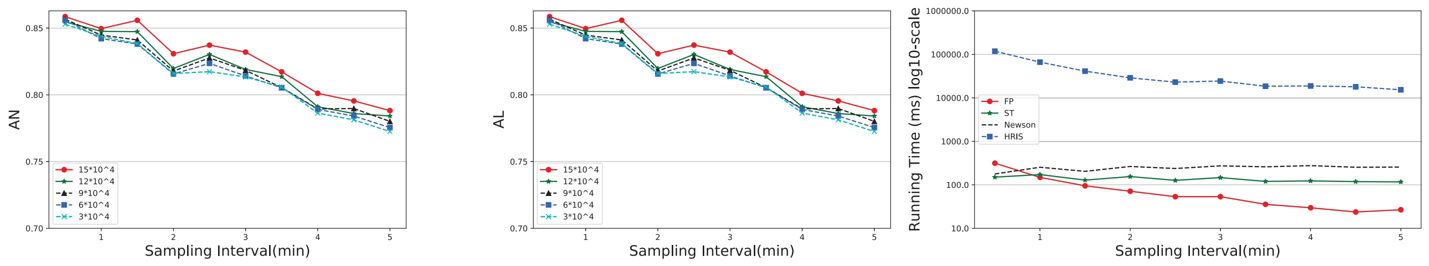

Fig. 6: From Left to Right: Effect of A_N and A_L by various sampling rate and size of historical trajectories, and running time

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 3–14, 1995.

[3] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 589–598, 2003.

[4] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 853–864, 2005.

[5] T. Eiter and H. Mannila. Computing discrete frechet distance. 05 1994.

[6] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. volume 15, pages 55–86, 2007.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 1–12, 2000.

[8] Z. Li, K. Liu, Y. Zhao, and Y. Ma. Mapit: An enhanced pending interest table for NDN with mapping bloom filter. volume 18, pages 1915–1918, 2014.

[9] Z. Li, L. Song, and H. Shi. Approaching the capacity of k-user MIMO interference channel with interference counteraction scheme. *Ad Hoc Networks*, 58:286–291, 2017.

[10] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2009, November 4-6, 2009, Seattle, Washington, USA, Proceedings*, pages 352–361, 2009.

[11] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2009, November 4-6, 2009, Seattle, Washington, USA, Proceedings*, pages 336–343, 2009.

[12] A. M. Pinto, A. P. Moreira, and P. G. Costa. A localization method based on map-matching and particle swarm optimization. volume 77, pages 313–326, 2015.

[13] M. A. Quddus, W. Y. Ochieng, L. Zhao, and R. B. Noland. A general map matching algorithm for transport telematics applications. *Gps Solutions*, 7(3):157–167, 2003.

[14] H. Wei, Y. Wang, G. Forman, Y. Zhu, and H. Guan. Fast viterbi map matching with tunable weight functions. In *SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems (formerly known as GIS), SIGSPATIAL'12, Redondo Beach, CA, USA, November 7-9, 2012*, pages 613–616, 2012.

[15] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. volume 8, pages 91–108, 2000.

[16] H. Wu, J. Mao, W. Sun, B. Zheng, H. Zhang, Z. Chen, and W. Wang. Probabilistic robust route recovery with spatio-temporal dynamics. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1915–1924, 2016.

[17] M. J. Zaki. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[18] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing uncertainty of low-sampling-rate trajectories. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 1144–1155, 2012.