Finding Frequent Sub-trajectories with Time Constraints *

Xin Huang
Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences
Shenzhen, China
xin.huang@siat.ac.cn

Jun Luo
Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences
Shenzhen, China
Huawei Noah's Ark Lab
jun.luo@siat.ac.cn

Xin Wang
Department of Geomatics Engineering
University of Calgary
Canada
xcwang@ucalgary.ca

ABSTRACT

With the advent of location-based social media and location-acquisition technologies, trajectory data are becoming more and more ubiquitous in the real world. Trajectory pattern mining has received a lot of attention in recent years. Frequent sub-trajectories, in particular, might contain very usable knowledge. In this paper, we define a new trajectory pattern called frequent sub-trajectories with time constraints (FSTTC) that requires not only the same continuous location sequence but also the similar staying time in each location. We present a two-phase approach to find FSTTCs based on suffix tree. Firstly, we select the spatial information from the trajectories and generate location sequences. Then the suffix tree is adopted to mine out the frequent location sequences. Secondly, we cluster all sub-trajectories with the same frequent location sequence with respect to the staying time using modified DBSCAN algorithm to find the densest clusters. Accordingly, the frequent sub-trajectories with time constraints, represented by the clusters, are identified. Experimental results show that our approach is efficient and can find useful and interesting information from the spatio-temporal trajectories.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: GIS; D.2.8 [**Database Management**]: Spatio-temporal Database—*Data Mining, Trajectory*

General Terms

Algorithms

1. INTRODUCTION

With the advent of location-based social media and location-acquisition technologies, large amount of trajectory data sets have been collected. Trajectories represent the sequences of spatio-temporal records of moving objects. Varieties of trajectory data are collected from mobile sensors like GPS on the taxis, LBS APPs etc. Accordingly, effective analysis of trajectory data has been a crucial topic aiming at finding useful knowledge from the big data sets. A trajectory is a sequence of spatio-temporal records of a moving object [14]. Frequent sub-trajectories (FST), namely the frequent subsequences of spatio-temporal records of moving objects, are the sub-trajectories contained by many different trajectories. In this paper, we address the frequent sub-trajectories as the spatio-temporal sequences which could be employed in finding or predicting the behaviors of moving objects.

The location information of each record in a trajectory could be represented as the numerical values or symbols. For example, the location information of hurricane trajectories and taxi trajectories is represented as longitude and latitude [12]. While the location information of trajectories collected from social media might be the city name or region's name[1],or preprocessed towards that format [25]. If the location information are the numerical values, we could use geometric shape matching techniques such as Fréchet distance to find FST [2]. However, the computation cost is very expensive in this way. If the location information are symbols, one trajectory can be treated as an alphabetical sequence (or a string). On the one hand, this kind of representation makes it easy to apply efficient string-matching algorithms instead of expensive shape-matching algorithms and previous work proved the effectiveness of these algorithms in the data mining community [4] [5]. On the other hand, the extendability to other data sets could be guaranteed since previous work showed that numerical representation could be transformed into string representation efficiently [24]. In terms of FST, we can use suffix tree to find it in linear time. In this paper, the location information is the city name which can be represented by a symbol.

There is also temporal information attached to each record. The temporal information could be the time stamp or the time interval for staying at each location. In this paper, we use the **staying time** for each city which is defined as the time interval from the first microblog appeared in this city to the first microblog appeard in the next city. If we consider temporal information in finding FST, the results could be more interesting. For example, we have two trajectories

*This research has been partially funded by NSF of China under poject 11271351, Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant 355996-2013 and the International Science & Technology Cooperation Program of China (2010DFA92720).

UrbComp '13, August 11-14, 2013, Chicago, Illinois, USA.

[1]For example, Tencent Microblog, t.qq.com

with staying time information:

$$Beijing \xrightarrow{15hours} Shanghai \xrightarrow{3.5hours} Jiaxing \quad (a)$$

$$Nanjing \xrightarrow{28hours} Huangshan \xrightarrow{23hours} Hangzhou \quad (b)$$

Here, sequence (a) shows the staying time at Shanghai is only 3.5 hours and it may be interpreted as a typical behavior of businessman taking a business trip from Beijing to Jiaxing by interchanging at Shanghai. Sequence (b), instead, may highlight the behavior of tourists travelling at the major scenic spots around the Yangtze River Delta. Therefore, we need to find FST not only with the same location sequence but also with the similar staying time in each location. We call those kind of FST as frequent sub-trajectories with time constraints (FSTTC). It seems that we can map the staying time to symbol so that we can use suffix tree to find FSTTC. For example, we can map staying time 1-2 hours to letter "A" and 2-3 hours to letter "B". However, in this way, the staying time 1.9 hours and 2.1 hours are different. But in reality, we know they are very close and should be treated as one cluster. Hence in this paper we propose a two-phase approach to find FSTTC. Firstly, we project the trajectories to location sequences and exploit suffix tree algorithm to find the FST. Secondly, we load all sub-trajectories which belong to one FST with the staying time information as the candidate sub-trajectories. Then we cluster all the candidate sub-trajectories to find the densest clusters with respect to the staying time. The final FSTTCs are those on the center points of the dense clusters.

To sum up, the contributions presented in this paper are:

1. The definition of a new trajectory pattern, a.k.a. FSTTC.
2. We exploit suffix tree to find FST.
3. We proposed a two-phase approach to find FSTTC. Since clustering (the second phase) is relatively more expensive, we prune all sub-trajectories of which the location sequences are not frequent after the first phase.
4. When calculating the distance between different trajectories in the second phase, we regard the locations as the coordinate axes and the stay time as the values of each axis, and then use the Euclidean distance to measure the distance.

The rest of the paper is organized as follows. In the next section we review the related works. In Section 3 we give some preliminary definitions. The two phases of our algorithm will be introduced in Section 4 and Section 5, respectively. In Section 6 we report the data set and the experiment results and in Section 7 we conclude our work.

2. RELATED WORK

In this section we summarize some relevant work within the context of trajectories analysis and string matching algorithms.

2.1 Trajectories Analysis

Trajectory Clustering Clustering is a substantial research topic in the data mining community. Influential clustering algorithms include DBSCAN [6], OPTICS [1], k-means [15] etc. Specifically to trajectory clustering, recently there has been considerable research in the area of analyzing and modeling spatio-temporal data. Gaffney et al. proposed a model-based algorithm clustering trajectories [7] [8]. They adopted a regression mixture model to represent the trajectories and conducted unsupervised learning using the maximum likelihood principle. Their work showed how EM algorithms could be adopted in trajectory clustering. The application of their framework in climate trajectories indicate that their approach outweigh traditional vector-based approaches. Their work was one of the earliest attempt in trajectory clustering. However they clustered the whole trajectories instead of sub-trajectories.

Lee et al. observed clustering the whole trajectory might miss some similar portions of the trajectories [12] . Therefore, the common sub-trajectories play a substantial role in many real world applications. Based on this observation they proposed the partition-and-group framework and developed a density-based line-segment clustering algorithm TRACLUS [12] .In TRACLUS, a trajectory is partitioned into a set of line segments at characteristic points, and then, similar line segments in a dense region are grouped into a cluster. The main advantage of TRACLUS is the discovery of common sub-trajectories from a trajectory database. Van Kreveld and Luo [20] considered the problem of sub-trajectory similarity measurement with time shift. The work in [3] improved the similarity measurement from the perspective of runtime. Buchin et al. addressed the problem of detecting commuting patterns with a sub-trajectory clustering based approach [2]. They proposed a Fréchet distance based algorithm to measure the distance between sub-trajectories. Their experimental results show that the problem of finding the longest sub-trajectory cluster is as hard as MaxClique problem to compute and approximate.

Trajectory Pattern Mining Vautier et al. [21] proposed a method to extract temporal information in the form of patterns named chronicles which contain numerical temporal constraints on events. The work in [11] extended conventional sequential pattern mining considering not only the orders of events indicated by the time stamps but also the time intervals. They defined projection levels and extended sequences with the time intervals as prefixes and postfixes to handle the temporal information. Giannotti et al. introduced trajectory patterns as concise descriptions of frequent behaviors, in terms of both space and time [9]. They used the neighborhood functions to model the Regions-of-Interest and proposed algorithms based on this and proposed various algorithms to mine out the patterns. One major feature of these works is that they considered the problem of trajectory pattern mining as an extension of traditional sequential pattern mining and association rule mining. Though very similar, we still need to emphasize that FSTTC is different from these patterns since FSTTC requires the locations sequences to be continuous in corresponding original trajectories while other patterns do not have this requirement.

The work in [24] considered the problem of trajectory pattern mining from the perspective of users. They proposed a two-phase framework called LP-Mine. In the modeling phase, they preprocess the trajectories to spatio-temporal sequences and in the mining phase they applied some traditional association rule mining algorithms to find the patterns corresponding to users behaviors and habits. Mamoulis et al. considered the problem of mining periodic patterns patterns in spatio-temporal data sets [16]. They argued that these patterns could facilitate data management significantly.

Mining Location History Multiple users' location history data sets convey a lot of knowledge. Using GPS trajectories generated by multiple users, Zheng et al. proposed a HITS-based inference model to mine the interesting locations and travel sequences. They constructed a graph to connect individuals and locations with weighted links [27]. In [26], the concept of LBSN (Location-Based Social Network) was defined. LBSN bridges the gap between physical network and the internet of things. Location-based social media plays a substantial role in people's daily life and the trajectory data sets behind it convey a lot of knowledge. [26] also argued the importance of modeling the locations histories effectively. For instance, the work of [23] [13] conducted user similarity analysis based on location history.

2.2 String-Matching Algorithms

Suffix tree is a popular string algorithm, widely used in finding common substrings, repeated substrings, substring matching etc. It was originally proposed in [22]. Ukkonen proposed an online algorithm which constructs a suffix tree in linear time [19]. When comparing with KMP (Knuth-Morris-Pratt Algorithm), another widely-used string algorithm, suffix tree outweighs KMP [18, 17] when we need to utilize the tree to find substrings for more than one time. Within the context of trajectory mining, string-matching algorithms provide an efficient solution since it can speed up the computation significantly compared to shape-matching algorithms and traditional clustering algorithms (ie. DBSCAN [6]). The work in [4] argued the effectiveness of string-matching algorithms utilized in trajectory mining. However, suffix tree is always adopted in exact matching. When it comes to the numerical values like the staying time at each place, exact matching can hardly be exploited.

Note that we need to handle multiple strings in this paper instead of just one string. In this case we can build a generalized suffix tree (GST) which is a suffix tree for a set of strings [10]. Given a set of strings $D = \{S_1, S_2, \ldots, S_d\}$ of total length n, a GST can be built in $O(n)$ time adopting Ukkonen's approach while the naive approach for constructing a suffix tree needs $O(n^3)$ time. Ukkonen's algorithm is an online algorithm and it speeds up the construction to $O(n)$ by introducing two indices and suffix link [19]. The number of nodes in a suffix tree is $O(n)$. Once a suffix tree is successfully constructed, finding the specific pattern only needs $O(p)$ where p is the length of the pattern which is usually much smaller than n.

3. PRELIMINARY DEFINITIONS

A trajectory is a spatio-temporal sequence of records for moving objects [14]. In this paper, we define a trajectory as follows:

DEFINITION 1. *Trajectory* $\tau = \{(s_1,t_1), (s_2,t_2), (s_3,t_3), \ldots, (s_{n-1},t_{n-1}), (s_n)\}$, *where* $t_i > 0$. *Here* s_i *represents a location and* t_i *represents the staying time at location* s_i.

A location sequence (LS) is a sequence of locations without time information.

DEFINITION 2. *Location Sequence* $LS = \{s_1, s_2, s_3, \ldots, s_{n-1}, s_n\}$. $LS(\tau)$ *is the location sequences contained in trajectory* τ.

4. COMPUTING LOCATION SEQUENCES

In our two-phase approach, we should firstly compute the location sequences to generate frequent sub-LSs so that candidate sub-trajectories for the second phase could be identified. A major advantage of this phase is to reduce the clustering workload in the second phase by pruning the non-competitive sub-trajectories using solutions in the first phase.

4.1 Location Sequences(LS) Generation

If we cluster the trajectories directly (using DBSCAN as an example), the time complexity is at least $O(k^2)$ where k is the number of trajectories. When it comes to sub-trajectory clustering, since the number of sub-trajectories could be $O(n^2)$ where n is the total length of all trajectories, the time complexity could be $O(n^4)$, which is definitely too expensive. Fortunately, a sub-trajectory which does not belong to FST can not be the candidate of FSTTC. Therefore we can compute FST using suffix tree and then prune the rest.

Since in this phase, we only need the location sequence, we project the original trajectories into a location sequence. In the projection, we get a replication of the original trajectory which does not contain the temporal information. For instance, the projection of trajectory (a) in the introduction is :

$$(Beijing \xrightarrow{15hours} Shanghai \xrightarrow{3.5hours} Jiaxing)$$

$$\downarrow Projection$$

$$(Beijing, Shanghai, Jiaxing)$$

Namely,

$$\tau = ((s_1,t_1),(s_2,t_2),...,(s_n)) \Longrightarrow LS(\tau) = (s_1, s_2, ..., s_n)$$

4.2 Mining Frequent Sub-LSs

First of all, we consider the location sequences as strings, put them to the GST one by one, and construct a GST using the Ukkonen's algorithm. We also maintain a counter for each node which represents the frequency of the substring from the root to this node.

DEFINITION 3. $Count = \sum_{i=1}^{f} |INDEX(i)|$ *where* $INDEX(i)$ *is the index contained in an offspring leave of a particular node, and* f *is the number of offspring leaves of this node.*

Fig. 1 illustrates the generalized suffix tree for a set of strings {'banana', 'ana'} with the Count values.

DEFINITION 4. $Length_{min}$*: the minimal length of sub-LSs. We assume a sequence which is too short (i.e., length = 1) is not useful in our work. This parameter is an empirical value and is specified manually in implementation.*

DEFINITION 5. $LS_{minsupport}$*: the minimal number of frequency of a sub-LS. This parameter is also an empirical value and is specified manually in implementation.*

After successfully constructed the GST . We traverse the nodes in the GST with the length to the root greater than or equal to $Length_{min}$. For each node, if the Count value is greater or equals to $LS_{minsupport}$, the sub-LS represented

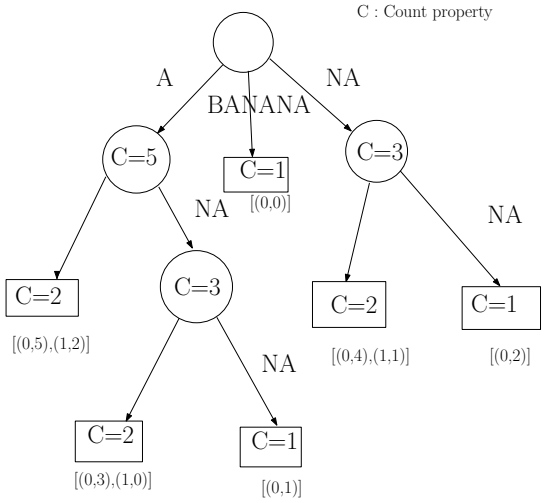


Figure 1: Example of a generalized suffix tree for a set of strings {'banana', 'ana'}, where the numbers in nodes are the Count values.

by this node is useful. Hence we load the original sub-trajectories corresponding to this sub-LS. The sets of trajectories are the expected results of the first phase. A typical example of the candidate set for the second phase is:
[(Beijing, 25.0 hours),(Shanghai, 27.0 hours), (Suzhou)],
[(Beijing, 15.0 hours),(Shanghai, 87.0 hours), (Suzhou)],
[(Beijing, 85.0 hours),(Shanghai, 27.0 hours), (Suzhou)],
[(Beijing, 26.0 hours),(Shanghai, 27.2 hours), (Suzhou)],
[(Beijing, 25.1 hours),(Shanghai, 27.1 hours), (Suzhou)].

Algorithm 1 summarizes the overall algorithm in this phase. More specifically, in line 3 we project the trajectories to location sequences by keeping a replication with temporal information removed. In line 5, the online linear algorithm constructing a suffix tree proposed by Ukkonen, namely the Ukkonen algorithm is adopted. There are at most $2n$ nodes in Ukkonen's approach representing a suffix tree where n is the total length of all strings involved. Then we traverse the constructed generalized suffix tree. Note in lines 9∼11, we prune the whole sub-tree of which the Count value of the nodes does not meet the required frequency. For all the nodes that meet the requirements on length and frequency, we get a set storing the two-dimension indices in its offspring leaves (see the numbers under the squares in Figure 1). The 2-dimension index (stringIndex, locationIndex) convey these facts : I. stringIndex is the index indicating which trajectory to load the sub-trajectory information. II. locationIndex is the index indicating where the sub-trajectory begins in the trajectory. Note that we should convert this index to $2locationIndex - 1$ when loading the data.

To sum up, in this section we generated a set of candidate sub-trajectories sets. Both the tree construction and traverse could be carried out with $O(n)$ complexity in both space and time by adopting Ukkonen's algorithm. Additionally, with the help of the index set we maintained, we can visit the required sub-trajectories in the original trajectories directly instead of long time searching. The number of candidate sub-trajectories generated in this step is generally much smaller than the number of existing sub-trajectories. In the next phase, we are going to cluster all the candidate sub-trajectories aiming at clustering sub-trajectories with similar staying time.

Algorithm 1 Algorithm Generating Candidate Sub-Trajectories.

Require:
A data set of trajectories which are time annotated, **Tr**;
The minimal length of a useful sequence, $Length_{min}$;
The minimal support of frequent sub-LSs, $LS_{minsupport}$;
Ensure:
A set of sets of candidate sub-trajectories, candidate sub-trajectories in one particular subset correspond to the same sub-LS, **C**;
1: **C** = ∅, GST = ∅ ,stringIndex = 0, locationIndex = 0, Index = ∅;
2: **for** each τ in **Tr** **do**
3: sequence = τ.project(); //project the trajectory to location sequence according to (1)
4: convert sequence to a string; //each unique char represents a location name
5: put sequenceInStringFormat to GST;
6: **end for**
7: node = GST.root
8: **for** each node in GST **do**
9: **if** node.Count < $LS_{minsupport}$ **then**
10: Prune the subtree rooted by this node;
11: Continue;
12: **end if**
13: length = node.getsubString().length;
14: **if** length < $length_{min}$ **then**
15: Continue;// each node represents a substring
16: **end if**
17: Index = node.getindices();// return all indices in the offspring leaves nodes
18: current = ∅;//current is the set of sub-trajectories having this sub-LS in projection
19: **for** each (stringIndex, locationIndex) in Index **do**
20: locationIndex = $2locationIndex - 1$;
21: length = $2length - 1$;
22: subtrajectory=loadData(stringIndex,locationIndex, length);//load data from **Tr**
23: current = current∪ subtrajectory;
24: **end for**
25: **C** = current ∪ **C**;
26: **end for**
27: **return** **C**;

5. CLUSTERING CANDIDATE SUB-TRAJECTORIES

5.1 Preliminaries

Unlike locations, the staying time between different locations can hardly be matched exactly. The consequence is that we can not calculate the frequency of a sub-trajectory by counting the occurrences directly even we generalize the staying time into symbol as we showed in introduction. Therefore, in this phase, we still use the traditional clustering idea to find FSTTC. We define the distance between two trajectories $\tau a, \tau b$ as follows.

Definition 6. *Distance*$(\tau a, \tau b)$ =

$$\begin{cases} \sqrt{\sum_{i=1}^{n-1}(\tau a_i.t - \tau b_i.t)^2} & LS(\tau a) = LS(\tau b) \\ \infty & LS(\tau a)! = LS(\tau b) \end{cases}$$

Where $LS(\tau a)$ and $LS(\tau b)$ are the location sequences contained in the two trajectories as defined in definition 2. $\tau a_i.t$

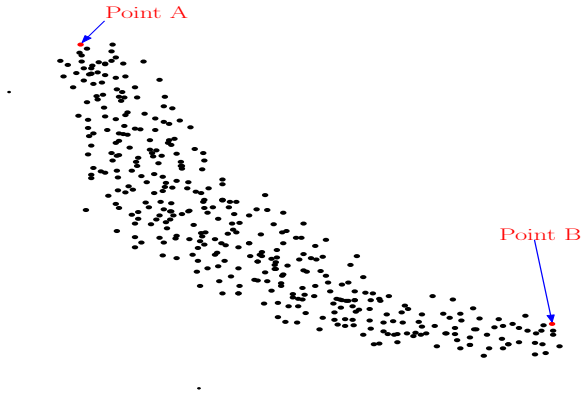


Figure 2: A typical cluster by DBSCAN.

and $\tau b_i.t$ are the staying time contained in ith location of τa and τb. Note that we consider the distance between two trajectories with different location sequences as infinity since in our work the locations are represented as symbols instead of numeric values.

5.2 Revised DBSCAN Clustering Algorithm

DBSCAN [6] is a density-based clustering algorithm. One vital advantage of DBSCAN is it can handle noise effectively. The high-level process of DBSCAN goes like this: if there are enough points in a point's eps-neighborhood, these points form a cluster, and if a neighborhood point has enough points in its eps-neighborhood, merge these points with the cluster, carry out the process recursively. One property of DBSCAN is that if a point is density-connected to any point of the cluster, it is part of the cluster as well. DBSCAN can find arbitrarily shaped clusters. Hence the distance of two boundary points could be very large. As is illustrated in Fig. 2, the distance between point A and B is very large. In our scenario, we need to find FSTTC such that the staying time should also be very similar between two points. Therefore, we need to modify DBSCAN to find FSTTC.

With respect to our application scenario where each location is represented as an axis and the staying time is treated as value of corresponding location, one parameter MaxDis is applied to original DBSCAN algorithm. More specifically speaking, when expanding a cluster if the distance between this point and the central point P is greater than MaxDis, we stop adding points around this point even though more density-connected points might be found around P. Hence in our approach a cluster like Fig. 2 will be divided into several clusters. Besides we also define the minimal size of a result cluster. Algorithm 2 illustrates our detailed modification on DBSCAN. More specifically, the modification is on the expandingCluster method of original DBSCAN. Namely we added one constraint on the distance between current point and the center point. As for the rest part of RevisedDBSCAN, we reused the corresponding parts of original DBSCAN.

5.3 Mining frequent sub-trajectories

Previously we generated candidate sub-trajectories. The candidate sub-trajectories are represented as sets of candidate sub-trajectories. Each inner set contains the candidates corresponding to one location sequence. We already

Algorithm 2 expandingCluster method in RevisedDBSCAN.

Require:
A data set D of points, the radius of a points neighborhood, **eps**; The minimum number of points required to form a cluster, **MinPts**; The maximum distance from the central point, **MaxDis**; The point currently evaluating in the expanding procedure, Point; //as to the rest input parameters check the original DBSCAN [6] ;
Ensure:
A set of points denoting a cluster, **Cl**;
1: add Point to cluster **Cl** ;
2: **for** each **P** in **NeighborPts** **do**
3: **if P** not visited **then**
4: mark **P** as visited;
5: **PNeighborPts** = return all points within **P**'s **eps**-neighborhood;
6: **if** sizeof(**PNeighborPts**) >= **MinPts** and distance(**P**, Point) < **MaxDis then**
7: **NeighborPts** = **NeighborPts** joined with **PNeighborPts**;
8: **end if**
9: **end if**
10: **if P** is not yet member of any cluster **then**
11: add **P** to cluster **Cl**;
12: **end if**
13: **end for**
14: **return** **Cl**;

defined that points with different location sequences have an extremely long distance. Hence we conduct clustering on every individual set instead of all candidates using RevisedDBSCAN. Algorithm 3 illustrates the implementation. In line 3, we adopt RevisedDBSCAN to find clusters in each inner set and in line 4-7 clusters the size of which are not big enough will be removed. Finally in line 9, we represent each cluster using the average value of all points in it.

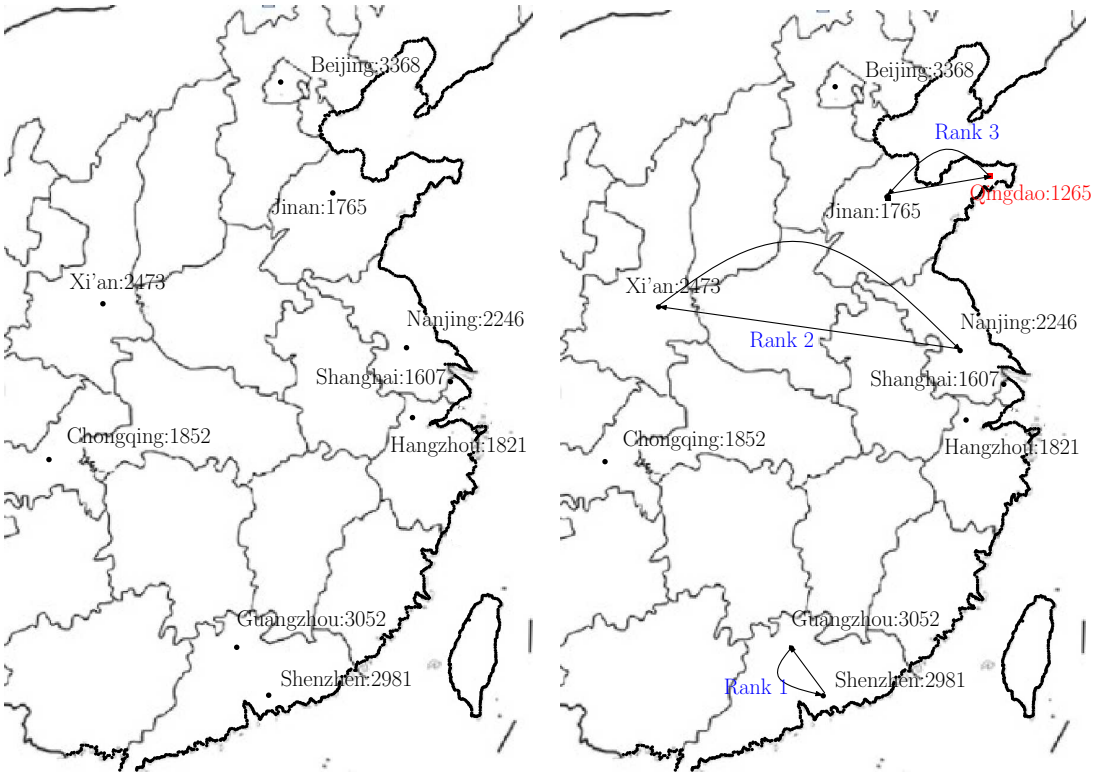
6. EXPERIMENTS

In this paper we adopted Tencent Microblog[2] data sets collected from 2010.10 to 2011.5 for the experiments. It contains 2606 users, 1007882 tweets and 337 locations. And for all experiments, parameters including minimal cluster size, maximum trajectory distance and the eps are constant. Since the data set contains information like location, time etc, we generate one trajectory for each user in the preprocessing procedure. More specifically, we remove the subsequent part of the continuous microblogs from one same user at the same location since this static part contains no movements. Hence after preprocessing 67280 microblogs were actually used in the experiments and the average length of the location sequences is 25.8.

Fig. 3 is the illustration of top 9 locations which appeared most frequently in the trajectories as well as the top 3 location sequences without considering staying time. The second frequent location sequence contains a location Qingdao which does not show up in the top 9 frequent locations. On the other hand, the most frequent location Beijing does not appear in any of the top 3 location sequences. Hence, though frequent locations are more likely to be part of frequent locations sequences, no necessary condition could be asserted.

When the temporal information is introduced, more in-

[2]t.qq.com



(a) Top 9 frequent cities in the trajectories. (b) Top 3 location sequences

Figure 3: Most popular cities and location sequences. The number after the city name is the frequency of the city.

teresting facts could be identified. In the experiments, the value we assigned to $length_{min}$ is 33, namely locations sequences which appear less than 33 times are excluded in the candidates generation procedure. We also assumed the minimum length of a useful sub-trajectory is 5, namely the minimum length of the location sequence is 3. The top 5 frequent sub-trajectories are listed in Table 1. The two parameters eps and MaxDis in revised DBSCAN algorithm are set as 66666(ms) and 101288(ms), respectively. The comparison between Fig. 3 and Table 1 successfully demonstrates that by considering the temporal information the frequent sequences are different.

From the top 3 frequent sub-trajectories, we can infer many people are on business trips between Shenzhen and Guangzhou and these business trips always take less than 3 days. Nanjing and Xi'an are 2 cities with a relatively long distance. Hence it is surprising to find that people from Nanjing travel to Xi'an so often. Another interesting fact is that many sub-trajectories in Table 1 contain repeating locations, namely users travel from Location A to Location B and come back to Location A. However sometimes we want sub-trajectories with distinct locations and these sub-trajectories might contain other kinds of interesting knowledge. Table 2 shows the frequent top 5 sub-trajectories consisting of distinct locations. From the first sub-trajectory, we can infer that a lot of people travel from Shenzhen to Chongqing by interchanging at Guangzhou. Other sub-trajectories also contain meaningful information.

The time complexity of phase 1 is $O(n)$ where n is the total length of all trajectories, including the cost of building the suffix tree and traversing the suffix tree. Whereas the time complexity of phase 2 is $O(\sum(m_i^2))$ where m_i is the size of a particular set containing all candidate sub-trajectories consisting of a particular location sequence. However, in real world applications, the total number of candidate sub-trajectories generated in phase 1 is much smaller than that of original trajectories. Hence the overall time complexity is still linear. Fig. 4 illustrates the overall runtime as well as the runtime of phase 1 in our approach are all linear.

Fig. 5 is a comparison about time cost of phase 1 between our algorithm with suffix tree (the black line) and the naive string-matching without suffix tree (the blue dashed line). This figure proved that suffix tree speeded up frequent sub-trajectory mining significantly.

Table 2: Top 5 Frequent Sub-trajectories with Time Constriants and Distinct Locations

Sub-trajectory	Frequency
Shenzhen $\xrightarrow{21.00hours}$ Guangzhou $\xrightarrow{21.00hours}$ Chongqing	28
Xi'an $\xrightarrow{16.30hours}$ Changzhou $\xrightarrow{23.67hours}$ Nantong	27
Chongqing $\xrightarrow{1days9.15hours}$ Guangzhou $\xrightarrow{2days13.90hours}$ Shenzhen	26
Shenzhen $\xrightarrow{3days0.55hours}$ Chongqing $\xrightarrow{3days20.71hours}$ Guangzhou	23
Changzhou $\xrightarrow{3days20.71hours}$ Zhenjiang $\xrightarrow{1days5.19hours}$ Xi'an	21

7. CONCLUSIONS

Table 1: Top 5 Frequent Sub-trajectories with Time Constriants

Sub-trajectory	Frequency
Shenzhen $\xrightarrow{1days23.02hours}$ Guangzhou $\xrightarrow{2days8.21hours}$ Shenzhen $\xrightarrow{2days8.21hours}$ Guangzhou $\xrightarrow{2days12.98hours}$ Shenzhen	416
Guangzhou $\xrightarrow{2days17.43hours}$ Shenzhen $\xrightarrow{3days15.09hours}$ Guangzhou	230
Shenzhen $\xrightarrow{2days17.29hours}$ Guangzhou $\xrightarrow{1days22.95hours}$ Shenzhen	225
Jinan $\xrightarrow{1days20.72hours}$ Qingdao $\xrightarrow{days16.31hours}$ Jinan $\xrightarrow{2days8.23hours}$ Qingdao	200
Nanjing $\xrightarrow{2days12.05hours}$ Xi'an $\xrightarrow{2days12.13hours}$ Nanjing	193

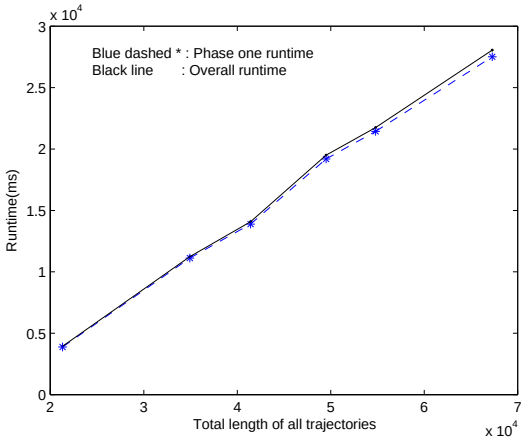


Figure 4: Overall runtime versus the total length of all trajectories.

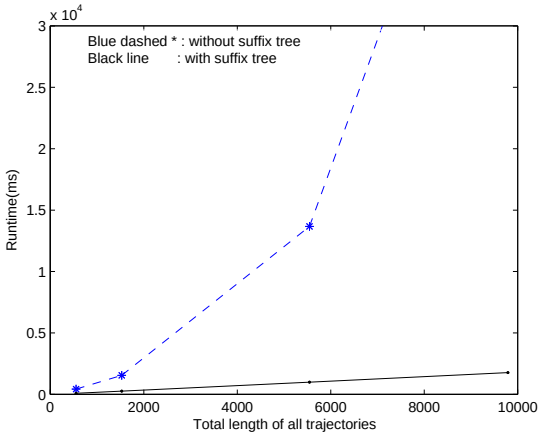


Figure 5: Runtime of phase 1 with(out) suffix tree versus the total length of all trajectories.

Algorithm 3 Mining frequent sub-trajectories.

Require:
A set of sets of candidate sub-trajectories, candidate sub-trajectories in the same subset correspond to the same sub-LS, **C**;
The radius of a point's neighborhood, **eps**;
The minimum number of points required to form a cluster, **MinPts**;
The maximum distance from the central point, **MaxDis**;
The minimal size of a valid cluster representing a frequent sub-trajectory, **MinSize**;

Ensure:
A set of all frequent sub-trajectories, **STr**;

```
1: STr = ∅, Sets = ∅ ;
2: for each Cd in C do
3:    Sets = RevisedDBSCAN(Cd, eps, MinPts, MaxDis);
4:    for each set temp in Sets do
5:       if sizeof(temp) < MinSize then
6:          Remove temp;
7:       end if
8:    end for
9:    Represents each subset of Sets using the average values of
      the points in this subset;
10:   STr = STr ∪ Sets;
11: end for
12: return STr;
```

In this paper, we define a new trajectory pattern called frequent sub-trajectories with time constraints (FSTTC) that requires not only the same location sequence but also the similar staying time in each location. We present a two-phase approach to find FSTTCs based on suffix trees. Experiments with the Tencent Microblog data set successfully demonstrate that our approach could discover frequent sub-trajectories with time constraints which contain interesting knowledge. The overall computing time is linear since the running time for the first phase is the dominating step. In the future, we would like to apply our algorithm to various data sets with longer repeat patterns to see whether our algorithms can achieve the same performance. Moreover, can we handle temporal information directly with suffix tree so that we do not need the second phase of clustering?

8. REFERENCES

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. *ACM SIGMOD Record*, 28(2):49–60, 1999.

[2] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of*

Computational Geometry & Applications, 21(03):253–282, 2011.

[3] K. Buchin, M. Buchin, M. Van Kreveld, and J. Luo. Finding long and similar parts of trajectories. *Computational Geometry*, 44(9):465–476, 2011.

[4] S. Dodge, P. Laube, and R. Weibel. Movement similarity assessment using symbolic representation of trajectories. *International Journal of Geographical Information Science*, 26(9):1563–1588, 2012.

[5] C. Du Mouza, P. Rigaux, and M. Scholl. Parameterized pattern queries. *Data & Knowledge Engineering*, 63(2):433–456, 2007.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Data Mining*, pages 226–231. AAAI, 1996.

[7] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 63–72. ACM, 1999.

[8] S. J. Gaffney, A. W. Robertson, P. Smyth, S. J. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. *Climate dynamics*, 29(4):423–440, 2007.

[9] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339. ACM, 2007.

[10] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[11] Y. Hirate and H. Yamana. Sequential pattern mining with time intervals. In *Proceedings of 10th Pacific-Asia ConferenceAdvances in Knowledge Discovery and Data Mining*, pages 775–779. LNCS, 2006.

[12] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[13] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[14] X. Li, J. Han, S. Kim, and H. Gonzalez. Roam: Rule-and motif-based anomaly detection in massive moving object data sets. In *Proc. SIAM International Conference on Data Mining*. SIAM, 2007.

[15] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[16] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245. ACM, 2004.

[17] M. Regnier. Knuth-morris-pratt algorithm: an analysis. In *Mathematical Foundations of Computer Science 1989*, pages 431–444. Springer, 1989.

[18] E. Ukkonen. Finding approximate patterns in strings. *Journal of algorithms*, 6(1):132–137, 1985.

[19] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[20] M. van Kreveld and J. Luo. The definition and computation of trajectory and subtrajectory similarity. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, page 44. ACM, 2007.

[21] A. Vautier, M.-O. Cordier, and R. Quiniou. An inductive database for mining temporal patterns in event sequences. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, volume 19, page 1640. LAWRENCE ERLBAUM ASSOCIATES LTD, 2005.

[22] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory*, pages 1–11. IEEE, 1973.

[23] X. Xiao, Y. Zheng, Q. Luo, and X. Xie. Finding similar users using category-based location history. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 442–445. ACM, 2010.

[24] Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie. Mining individual life pattern based on location history. In *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware*, pages 1–10. IEEE, 2009.

[25] Z. Yin, L. Cao, J. Han, J. Luo, and T. S. Huang. Diversified trajectory pattern ranking in geo-tagged social media. In *Proc. SIAM International Conference on Data Mining*, pages 980–991. SIAM, 2011.

[26] Y. Zheng. Location-based social networks: Users. In *Computing with Spatial Trajectories*, pages 243–276. Springer, 2011.

[27] Y. Zheng and X. Xie. Learning travel recommendations from user-generated gps traces. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(1):2, 2011.