Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 3838147, 10 pages
https://doi.org/10.1155/2022/3838147

Research Article

Prefix-Pruning-Based Distributed Frequent Trajectory Pattern Mining Algorithm

Jiaman Ding, Yunpeng Li, Ling Li, and Lianyin Jia

Kunming University of Science and Technology, Artificial Intelligence Key Laboratory of Yunnan Province, Kunming 650500, Yunnan, China

Correspondence should be addressed to Lianyin Jia; jlianyin@163.com

Received 6 January 2022; Revised 26 February 2022; Accepted 2 April 2022; Published 6 May 2022

Academic Editor: Ana C. Teodoro

An important problem to be solved in smart city construction is how to improve the efficiency of mining frequent patterns that can be used for location prediction and location-based services of massive trajectory datasets. Owing to uncertain personal trajectory and non-explicit trajectory items, the existing sequence mining algorithms cannot be used directly. To solve this problem, this study proposes a distributed trajectory frequent pattern mining algorithm (SparkTraj) based on prefix pruning. First, a grouping and partitioning technique is used to abstract the original trajectory data and convert them into a common time series.Then, the generation of a redundant trajectory pattern is avoided by using the path adjacency pruning method. Second, to improve mining efficiency, SparkTraj is designed and implemented in Spark, which employs cluster memory computing. Finally, experiments on common datasets show that the proposed algorithm can effectively extract frequent trajectory patterns, and, in particular, deal with the massive amounts of trajectory data. Compared with common trajectory pattern mining algorithms, the SparkTraj algorithm not only improves the overall performance but also has good scalability.

1. Introductionn

At present, academia has carried out in-depth research on trajectory data as many problems in trajectory data mining must be solved, including trajectory data collection, preprocessing, and storage management, followed by trajectory pattern mining, anomaly detection, and activity recognition. With the rapid increase in the amount of trajectory data, how to efficiently mine useful patterns (rules) from massive amounts of frequent trajectory data has been challenging. Therefore, it is particularly important to design efficient and practical trajectory frequent pattern mining algorithms. Solving the problem of trajectory pattern mining generally includes two basic steps: trajectory abstraction and frequent pattern mining [1].

Trajectory abstraction is the primary task of trajectory data mining. Owing to the high accuracy of GPS positioning, it is difficult to collect completely consistent trajectory data on the same path and thus difficult to find patterns from the original GPS trajectory data [2]. Accordingly, the abstract expression of trajectory data is gaining significance. To reduce the associated computational pressure, the general method of trajectory abstract expression is to transform the original trajectory sequence into a unified description of the public segment sequence. According to different abstract methods, they are subdivided into road network-based [3], partition-based [4], and geometry- based [5].The geometry-based method is used to cluster adjacent trajectory segments and includes line simplification technology, density clustering, and a partitioning and a grouping framework. It is more suitable for trajectory data with high uncertainty, but the choice of different clustering algorithms is more efficient. The difference in impact is clear. Moreover, through trajectory abstraction, frequent pattern mining in trajectory data is transformed into frequent sequence pattern mining in transaction data.

As distributed and parallel computing technologies become increasingly more mature, technical possibilities are provided for effectively solving trajectory mining problems with massive amounts of data. To find the trajectory pattern

from the massive amount of trajectory data, Qiao et al. proposed a parallel sequence mining algorithm (PartSpan) [6] that extends the traditional GSP algorithm to find the trajectory pattern. The algorithm uses prefix projection technology to decompose the search space to reduce the generation of candidate sequences. At the same time, a parallel formula is proposed to make the data distribution reasonable, and the candidate pruning strategy can effectively solve the I/O cost. However, since the GSP algorithm based on Apriori will generate a large number of candidate sequences, it must still be optimized in terms of time and memory consumption of a large number of scanning sequence databases. Then, Qiao et al. proposed the plute algorithm [7], which is based on the Hadoop distributed framework and uses k rounds of MapReduce jobs to find all trajectory patterns. However, the efficiency of the MapReduce computing framework in iterative operations must be optimized because in each round of iteration a new MapReduce job must read data from the Hadoop Distributed File System (HDFS) and write it back, which leads to high I/O and communication overhead. To reduce both the I/O and communication overhead, Liu et al. proposed an efficient Spark-based distributed trajectory pattern mining (DTPM) algorithm [8]. The DTPM algorithm uses memory calculations to load the trajectory sequence datasets and trajectory patterns generated in each iteration into flexible distributed datasets (RDDs) to overcome high I/O and communication overhead. Accordingly, the algorithm uses the two adjacent elements in the trajectory sequence (actually two adjacent road segments) to optimize the candidate sequence pattern generation process of the original GSP algorithm, so that each candidate trajectory sequence can only grow with its adjacent road segments, reducing the number of candidates. The generation of trajectory sequence improves performance. However, the GSP algorithm itself is based on the Apriori algorithm, and the problem of cyclic scanning of the sequence database remains.

To this end, this study proposes a trajectory frequent pattern mining algorithm (SparkTraj) that is based on prefix pruning strategies and combines distributed techniques. The original trajectory data is first grouped and partitioned for abstract expression into a common time series, and then the path adjoining pruning strategy to reduce the redundant trajectory patterns after pruning is designed. The algorithm is designed and implemented by combining the Spark cluster with memory computing advantage and using the mode growth algorithm. After verification, the algorithm improves the mining efficiency.

2. Trajectory Data Abstraction

In this study, the mining framework proposed mainly includes two parts, trajectory abstraction and distributed trajectory pattern mining. Figure 1 shows the work framework. Trajectory abstraction solves the problem of a computer implicitly representing trajectory data, so that sequence mining algorithms can be used to mine trajectory patterns. Then, the frequent sequence mining algorithm implemented in the distributed environment is used to find frequent patterns and obtain the frequent patterns of trajectories.

In a transaction database, each transaction consists of a set of items sorted by time. For example, <abc> is a sequence transaction containing three items, and these items are explicit. For trajectory data, however, each trajectory is composed of a series of coordinate numbers. Owing to the high positioning accuracy, even if the coordinates of the same location are collected, they are not completely consistent. Therefore, the trajectory data item is implicit. Therefore, when processing trajectory data, the primary task of frequent pattern mining is to identify explicit items, namely trajectory abstraction. Geometry-based methods were used in this study to abstract the trajectory data, which can maintain the activity characteristics of the individual trajectory.

2.1. Definitions and Properties

Definition 1 (track point P): $P = \{lng, lat, t\}$; the three elements represent longitude, latitude, and timestamp, respectively.

Definition 2 (trajectory T): $T = \{P_1, P_2, \ldots, P_n\}$, $P_1.t < P_2.t < \ldots < P_n.t$, which means a trajectory composed of some continuous trajectory points.

Definition 3 (path R): $R_i = \{P_i(x_{1i}, y_{1i}), P_j(x_{2i}, y_{2i})\}$, where (x, y) are the end-point coordinates of R_i. It is a scalar with no direction.

Definition 4 (common subsegment S): $S_i = \{R_i, t_{from}, t_{to}\}$, where t_{from} and t_{to} represent the start and end times, respectively, when R passes through S_i. Although different S_i may have the same R_i, each S_i may have different time attributes.

Property 1. The two adjacent elements S_i and S_{i+1} in the trajectory sequence T are two adjacent sections. For example, in Figure 2 (c), if one element in the track sequence T is S_1, then the adjacent elements of S_1 in T should be S_2 and S_7.

2.2. Trajectory Processing and Common Segment Extraction. In the framework of this study, the original trajectory data are first pre-processed, which is divided into three steps: the first step is clearing (that is, removing outliers); usually, outliers that deviate from the continuous trajectory sequence exceed the velocity threshold. The data are cleared. The velocity threshold is determined by a random sample and the Layda criterion [9]. The second step is dividing (that is, dividing the trajectory into discrete strokes with a clear start and end-point). The main purpose of dividing is to identify discontinuous points. When the distance between two data points is greater than the distance threshold, and the interval time is greater than the time threshold, the trajectory is divided into multiple trajectories, data simplification (that is, compressing the point stroke and converting it into a linear stroke). The third step uses the Douglas–Peucker (DP) line

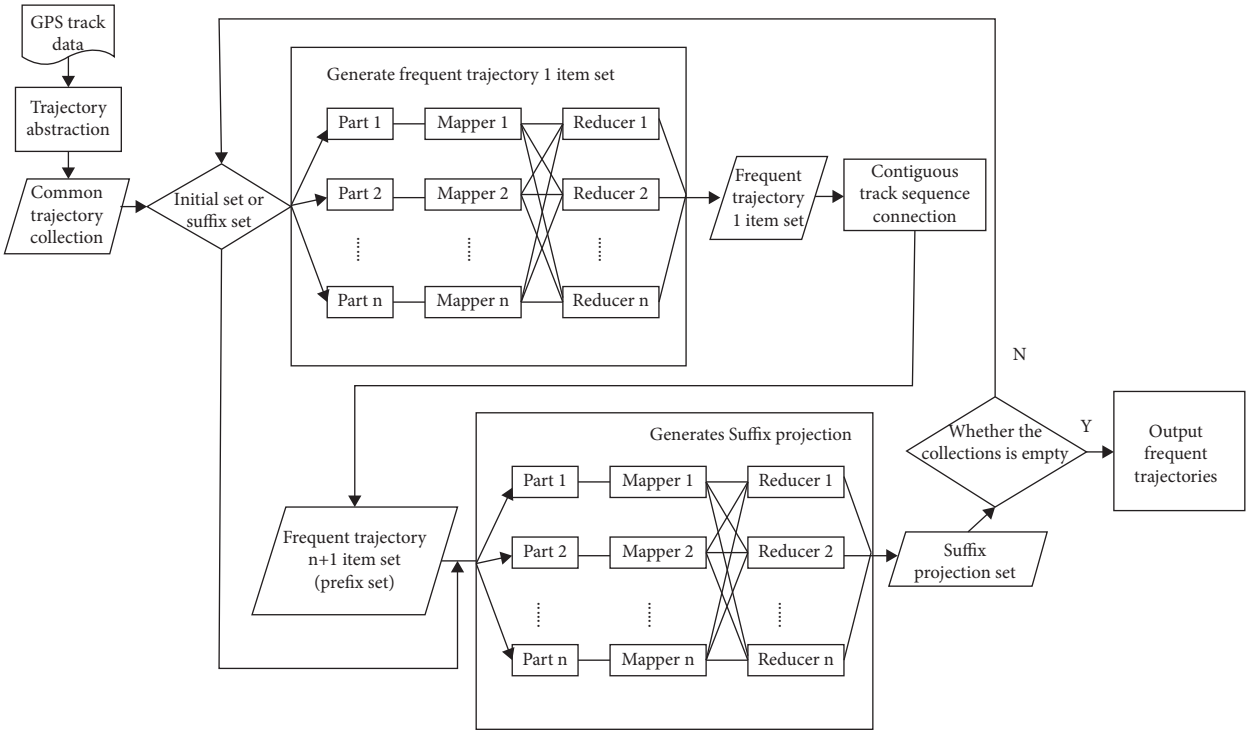

Figure 1: Work frame.

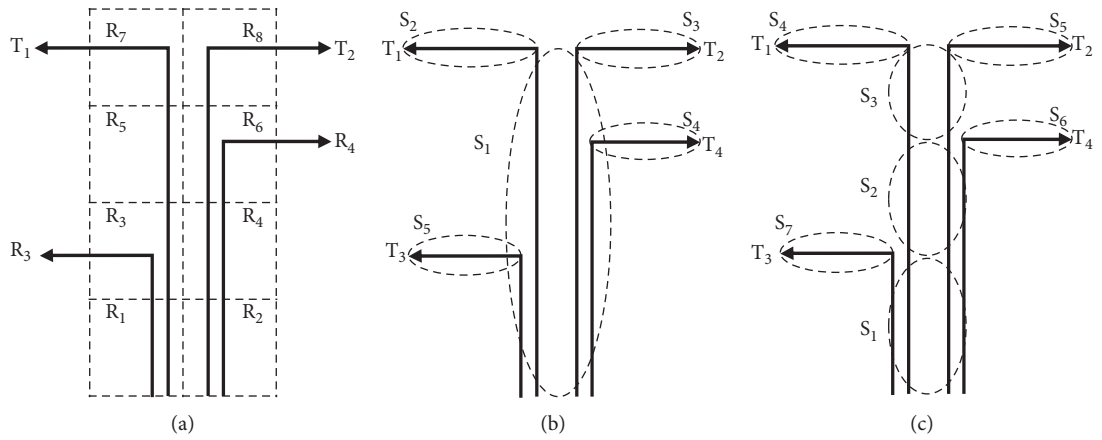

Figure 2: Schematic of abstract expression of trajectory.

simplification algorithm to compress the trajectory. After compressing the data, reducing the amount of data, and removing redundant data [10], the detection co-movement behavior between different strokes becomes possible.

For trajectory division, we divided the original trajectory into two situations: the user stays in one place or the user turns off the data recording device. For the former case, we used a dwell extraction algorithm to detect dwell points in the trajectories. The trajectory is then divided at each access point (for a stop point, its first location becomes the destination of the last journey, and the last location becomes the starting point of the current journey). For the latter case, we divided the distances based on a temporal threshold. The main purpose of trajectory division is to identify discontinuities, divide the trajectories into different trajectories, and detect stop points, consider the continuity of time in the division process besides the spatial proximity of the trajectory points. Thus, trajectories are divided when adjacent points and distance are greater than the distance threshold and the interval is greater than the time threshold.

The output of trajectory preprocessing is a set of paths R based online segments. Since the line segments in path R are not completely repeated, we use a method of grouping and partitioning to find the common movement behaviors represented by the common sub-segment (sub-segment cluster) S in all R. The boundary problem [11]in Figure 2(a) shows that directly applying the clustering algorithm to these line segments may miss some common subsegments. When

the historical trajectory data are not very sufficient, this problem will be serious because we may not find enough public movement behaviors for frequent pattern mining. The alternative we use goes through two stages (grouping and partitioning), the purpose of which is to discover all subsegment clusters from R.

In the grouping and partitioning stages, we use the line clustering algorithm to identify all candidate segment clusters that can accommodate common subsegments. Before giving the details of the algorithm, we first define the distance function, including the vertical distance and the directional distance formula as shown in Figure 3.

Given two line segments $L_1 = s_1e_1$ and $L_2 = s_2e_2$, we let L_1 be the short line segment and L_2 the long line segment. The vertical distance between L_1 and L_2 is calculated using the defined distance function [12] formula (1), where $d_{\perp 1}$ is the vertical distance from s_1 to L_2. The angular distances L_1 and L_2 are defined as their crossing angle θ. Regarding the straight line as a vector, the inner-product operation is used to calculate its angular distance, as shown in formula (2), where $||L_1||$ is the length of L_1.

$$d_{\perp}(L_1, L_2) = \frac{d_{\perp 1}^2 + d_{\perp 2}^2}{d_{\perp 1} + d_{\perp 2}}, \tag{1}$$

$$d_{\theta}(L_1, L_2) = \cos^{-1}\left(\frac{L_1 \cdot L_2}{\|L_1\|\|L_2\|}\right). \tag{2}$$

The clustering algorithm has high computational complexity because all element pairs must be compared multiple times. To accelerate this process, we improved a heuristic line clustering algorithm (Algorithm 1).

The algorithm first selects the longest line segment ll in the line segment set LS as the initial seed. Then, several row clusters are formed through two steps (filtering and clustering). The filtering step only considers the angular distance and the clustering step only the vertical distance. The reason for choosing the longest line as the seed is that it can better capture the user's main movement behavior. The filtering step can decompose the entire line segment set into several independent line segment sets, thereby reducing the size of the search space of the clustering step. After filtering, "coarse" clustering can first be performed on the data. Although the accuracy is low, it has a great speed advantage. After obtaining the k value, k means is used for further "fine" clustering to obtain the line segment set in each group. Next, the above process is repeated for the line segments without any line segment clusters. As shown in Figure 2(b), five-line segment clusters are formed after grouping and partitioning are completed. After the common sub-segment is found, R is transformed into a group of path clusters based on the subsegment by the scan line method, which is defined as S [Figure 2(c)].

3. Trajectory Frequent Pattern Mining

Frequent pattern mining of trajectory data is a special case of transaction data sequence pattern mining. The trajectory pattern mining algorithm proposed in this study is based on

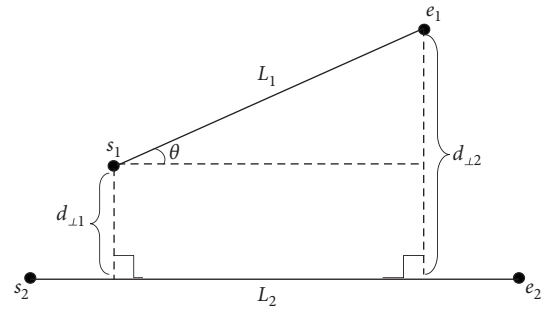

Figure 3: Schematic of the distance formula.

PrefixSpan, a sequence mining algorithm of frequent pattern projection. In this section, we first describe the improvement of frequent pattern projections that deal with trajectory datasets. Then, we describe the SparkTraj algorithm based on Spark.

3.1. SparkTraj's Prefix Pruning Strategy. The main cost of the algorithm based on frequent pattern projection is to generate suffix projection; that is, to recursively form the projection datasets [13]. If the minimum support is defined to be very low, many projection datasets will be generated, which will affect the mining efficiency. The improved algorithm modifies the generation process of the prefix trajectory pattern according to Property 1 above. The improved prefix trajectory pattern generation process is as follows: trajectory sequence s_1 and trajectory sequence s_2 can only be connected when s_1 and s_2 are adjacent road sections. Therefore, the number of generated projection trajectory sequences is reduced, thereby improving the performance of sequence mining algorithms. The improved candidate trajectory pattern generation process is shown in Algorithm 2.

To make the improved sequence connection easier to understand, an example is given in Figure 2(c), and the minimum support of the trajectory pattern to 2 is set. First, we find the frequent-item set (S_1,S_2,S_3) from T, with S_1 as the initial prefix, according to the adjacent property of the trajectory data. Then, the set of adjacent items (S_2,S_7) that meet the conditions is obtained, and two frequent items (S_2, S_3) are detected from the generated projections $(S_2S_3S_4$, $S_2S_3S_5$, $S_2S_6)$. Prefix S_1 is combined with frequent item set S_2 to generate a new prefix S_1S_2. The above process is repeated and a prefix set $(S_1,S_1S_2,S_1S_2S_3)$ is generated. These prefixes are trajectory patterns. The sequence prefix generation process is shown in Table 1.

3.2. Spark Implementation of SparkTraj Algorithm. The implementation of the proposed improved trajectory pattern mining algorithm in the Spark environment is described in this subsection. The projection datasets of each pattern are collected and processed on each machine in parallel through mapping and reduction. The SparkTraj algorithm includes two stages of mining, namely mining of frequent itemset and suffixes. In the generation of projection, a frequent trajectory sequence of length plus one is obtained through two stages each time.

```
Input.(1) A set of lines LS, (2) two parameters λperpendicular and λorientation
Output.A set of line clusters CS
(1) While LS is not empty do
(2) Find the longest line ll in LS
(3) Verified lines VS = Ø
(4) For each line li in LS do
(5)   if dθ(ll, li) < λorientation then
(6)     Append li to VS, and remove li from LS
(7)     k = Clustering(VS, λperpendicular)
(8) Line clusters CCS = k means(k, λperpendicular)
(9)   end if
(10) Append all the line cluster of CCS to CS
```

Algorithm 1: Heuristic line clustering algorithm.

```
Input.L1, L: trajectory pattern
Output.Ck+1: k+1 prefix trajectory pattern
(1) For each trajectory s1 in L do
(2) For each trajectory s2 in L1 do
(3) if k = = 1 then
(4)   if s1 and s2 are adjacent road segment
(5)     c = s1∞s2;/*Join Phase*/
(6)     add c to Ck+1;
(7)   end if
(8) else
(9)   if (s1 [2] = s2 [1])∧(s1 [3] = s2 [2])∧. . .∧(s1[k] = s2[k-1]) then
(10)    c = s1∞s2;/* Join Phase*/
(11)  if c contains non-trajectory pattern
(12)    delete c;
(13)  else
(14)    add c to Ck+1;
(15)  end if
(16) return Ck+1
```

Algorithm 2: AdjacentPrefixCon (L1,L).

Table 1: Improved sequence connection example.

Prefix	Trajectory spatio-temporal adjacency item set	Projection set	Frequent itemsets
NULL	*NULL*	S_1S_7 $S_1S_2S_3S_4$ $S_1S_2S_3S_5$ $S_1S_2S_6$	S_1,S_2,S_3
S_1	S_2,S_7	S_7 $S_2S_3S_4$ $S_2S_3S_5$ S_2S_6	S_2,S_3
S_1S_2	S_3,S_6	S_3S_4 S_3S_5 S_6	S_3
$S_1S_2S_3$	S_4,S_5	S_4 S_5	*NULL*

The SparkTraj algorithm 3 takes the trajectory sequence data D and the minimum support as the input and returns the trajectory frequent sequence pattern. L is the frequent sequence pattern in each cycle, and $D|r$ is the obtained projection datasets of the frequent sequence pattern. Each sequence in this dataset is a suffix projection set prefixed with L. In the first cycle, the value of L is empty, so trajectory sequence data D is used as the input. In the loop, the length

```
Input.D, ξ: the minimal support
Output. The complete set of frequent patternsP
(1) L = {};
(2) P = {};
(3) D|r = D;
(4) While D|r is not empty do
(5) L1 = findFrequentItems(D|r);
(6) L = L1.flatmap(.AdjacentPrefixCon (L1,L));
(7) P.append(L);
(8) D|r = gensuffix(L,D|r);
(9) Return P;
```

Algorithm 3: SparkTraj.

```
Input. D|r, ξ:the minimal support
Output. L1
(1) RDD1 ← flatMap(line offset, D|r);
(2) RDD2 ← RDD1.map<r,1>;
(3) RDD3 ← RDD2.reduceBykey(r,count);
(4) RDD4 ← RDD3.filter(count≥ ξ);
(5) Return L1(RDD4);
```

Algorithm 4: FindFrequentItems.

```
Input.D|r, L: the k trajectory pattern,
Output. D|r
(1) RDD5 ← flatMap(line offset, D|r);
(2) RDD6 ← map(L,RDD5);
(3) RDD7 ← RDD6.yeild(._project);
(4) RDD8 ← RDD7.reduceByKey(L, D|r);
(5) Return D|r(RDD8);
```

Algorithm 5: genSuffix.

of the frequent sequence pattern L is incremented by one. The findFrequentItems and genSuffix functions obtain the suffix projection sets corresponding to L_1 and L that meet the minimum support. P is increased by using L_1 to generate a frequent sequence L with a length plus one and $D|r$ is assigned to the new projection set and enters the next cycle until $D|r$ is empty.

SparkTraj's frequent one-item set generation algorithm 4:

First, the projection suffix set is loaded into Spark RDD to take advantage of the cluster memory. Then, the flatMap function is used to read all the trajectory sequences and obtain all the road section items. After the flatMap function is completed, the map function will be applied to convert each road section item into a key-value pair of <*item,1*>. Finally, the reduceBykey function is called to calculate part of the supported number of items, and an item set L_1 that is not less than the minimum support ξ is found.

Before the projection generation stage of the kth iteration, we first read L_1 from the RDD and generated a trajectory pattern L of length $k + 1$ through the modified prefix trajectory pattern connection algorithm 5.

Then, L and the suffix projection $D|r$ prefixed with L are used as the input to generate a new suffix set $D|$, and the trajectory projection sequence and prefix sequence are used as input to generate a new projection set.

3.3. Algorithmic Time Complexity. The first stage, mainly for the preprocessing of the trajectory data, extracts the common subsegments. The algorithm obtains multiple tracks by dividing the tracks, before obtaining the set of line segments of each grouping through clustering, and constantly repeating the clustering steps until the common subsegment is found. The ls represents the LS length, and the time spent in this stage is represented as the $ls \times ls$. The second stage is the first to generate a frequent item set. Second, prefix sets are generated based on the sequence of adjacent trajectories. The suffix projection set is then generated according to the prefix set, and the final prefix set (the frequency pattern) is found through the set of suffix projection. The d is the size of the trajectory sequence data D, and the time spent at this stage is represented as the $d \times (2^{(d+1)} + 2^{(d+d)})$. So the total time spent in the final algorithm is $ls \times ls + d \times (2^{(d+1)} + 2^{(d+d)})$.

4. Experiment and Evaluation

4.1. Data and Experimental Environment. The trajectory datasets were collected in the (Microsoft Research Asia) Geolife project and completed by 182 users over more than five years (April 2007 to August 2012). The trajectory of the datasets is represented by a series of timestamp points, and each timestamp point contains latitude, longitude, and altitude information. The datasets contain 17,621 trajectories, with a total distance of 1,292,951 km and a total time of 50,176 h [14].

The experimental distributed platform has a main node and six data nodes. Each host node is configured with a single core and 2 GB of memory. The environment is CentOS7, Hadoop2.7, Spark2.3, and programming is completed based on Python 3.6.

4.2. Trajectory Abstraction

4.2.1. Pre-Processing. Outliers exist in the data of most participants. Overall, approximately 4% of the original track points are deleted. Most outliers are collected inside the building in which the experiment was conducted, and the main reason is that the signal is lost inside the building.

4.2.2. Trajectory Division. To identify the continuous route and stopping point of the original trajectory data, based on the previous research [15], the distance threshold is set as $\delta_d = 60$m and the time threshold as $\delta_t = 500$ s. Figure 4 shows the results of the division. Compared with the number of original trajectories, the number of paths for each participant has increased in different proportions, with an average increase of 29.5%. This shows that the trajectory points are discrete in space and discontinuous in time. This also confirms the necessity of trajectory division.

4.2.3. Segmentation. When compressing the original data, it is necessary to find the height of the triangle based on the three-point coordinates. Since the original data are represented by latitude and longitude, the two-dimensional coordinate system formula cannot be directly used for the calculation in the compression process. To use the DP algorithm to compress the original trajectory, it was decided to follow the Helen formula $S = \sqrt{(p(p-a)\ (p-b)\ (p-c))}$ calculation, where a, b, and$c$ are the distances between the two coordinates, $p = (a+b+c)/2$, and S is the distance of the required point to the line segment, the distance threshold is set as $\lambda_{distance} = 25$ m, and the simplified point ratio is shown in Figure 5.

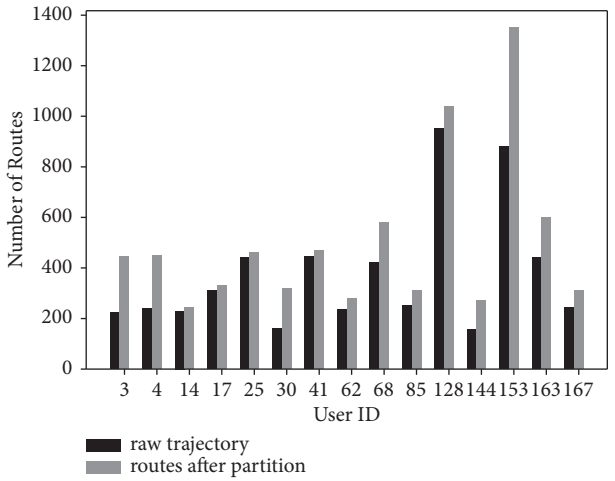

Figure 4: Trajectory division result.

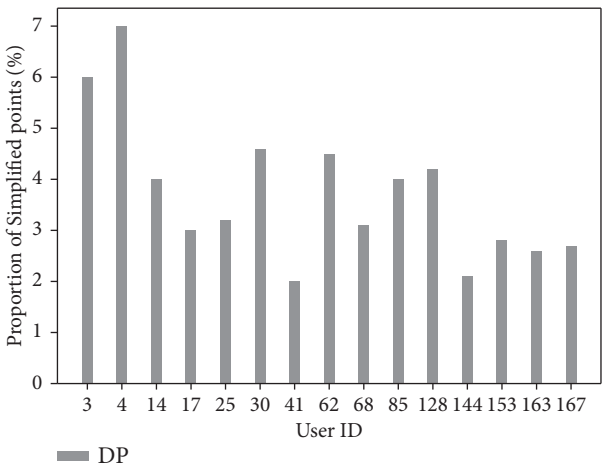

Figure 5: DP algorithm simplification of point ratio.

4.2.4. Common Subsection Discovery. With increasing distance threshold δ_d, the number of common segments gradually decreases; that is, the higher the threshold, the fewer line segments are grouped. If the threshold is too high, the mining accuracy will decrease and the time consumed will decrease. The experiment found that when $\delta_d > 100$ m, the number changes in the common segment tends to be stable, so the distance threshold $\delta_d = 100$ m is set.

As the angle threshold δ_θ increases, the clustering of line segments can withstand more interference, and longer and more complete line segment groups can be extracted. At the same time, the search range increases and the program execution time increases. The mining framework proposed in this study is suitable for the mining of personal trajectory patterns. Therefore, we selected the experimental participants with the longest data-collection time to evaluate the efficiency. It was decided to set the angle threshold $\delta_\theta = \pi/12$ in the experiment to find the common subsegments, and it was found that the number of common subsegments is positively correlated with the number of original trajectory points. In addition, the average segment lengths of different participants did not differ much. The results of some common subsegments are shown in Figure 6. The original trajectory data have high accuracy, so the data at the same location may not be the same. After trajectory abstraction processing, the data passing through the same path can be represented by the same line segment, which is frequently used. The pattern algorithm provides feasibility.

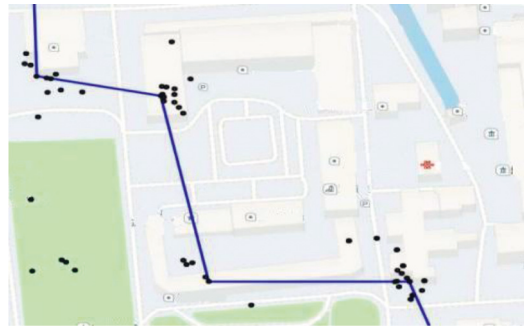

Figure 6: Public segment example.

4.3. Trajectory Mining Experiment. We use the proposed SparkTraj algorithm to mine frequent trajectories. A frequent trajectory is shown in Figure 7. It was found from the analysis that the user's mobile behavior obtained through the mining algorithm, and the user's travel destination (shopping malls, parks, public places, etc.) can be inferred. One can also obtain the user's movement route between location nodes, and further combine the actual route or semantic information to roughly infer the user's travel mode and road section conditions, which is of great help to traffic management and route inference [16-21]. Next, we conducted

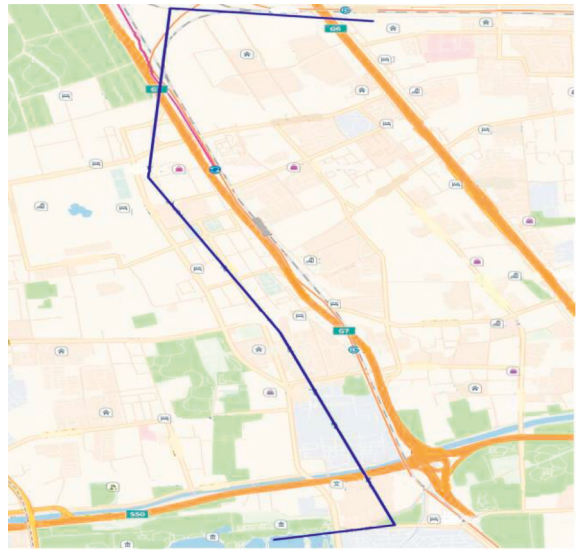

Figure 7: Frequent route.

experiments on algorithm efficiency, distributed speedup, and algorithm scalability.

4.3.1. Experiment 1. In this experiment, we analyzed the runtime of the PrefixSpan algorithm, traditional Prefixspan

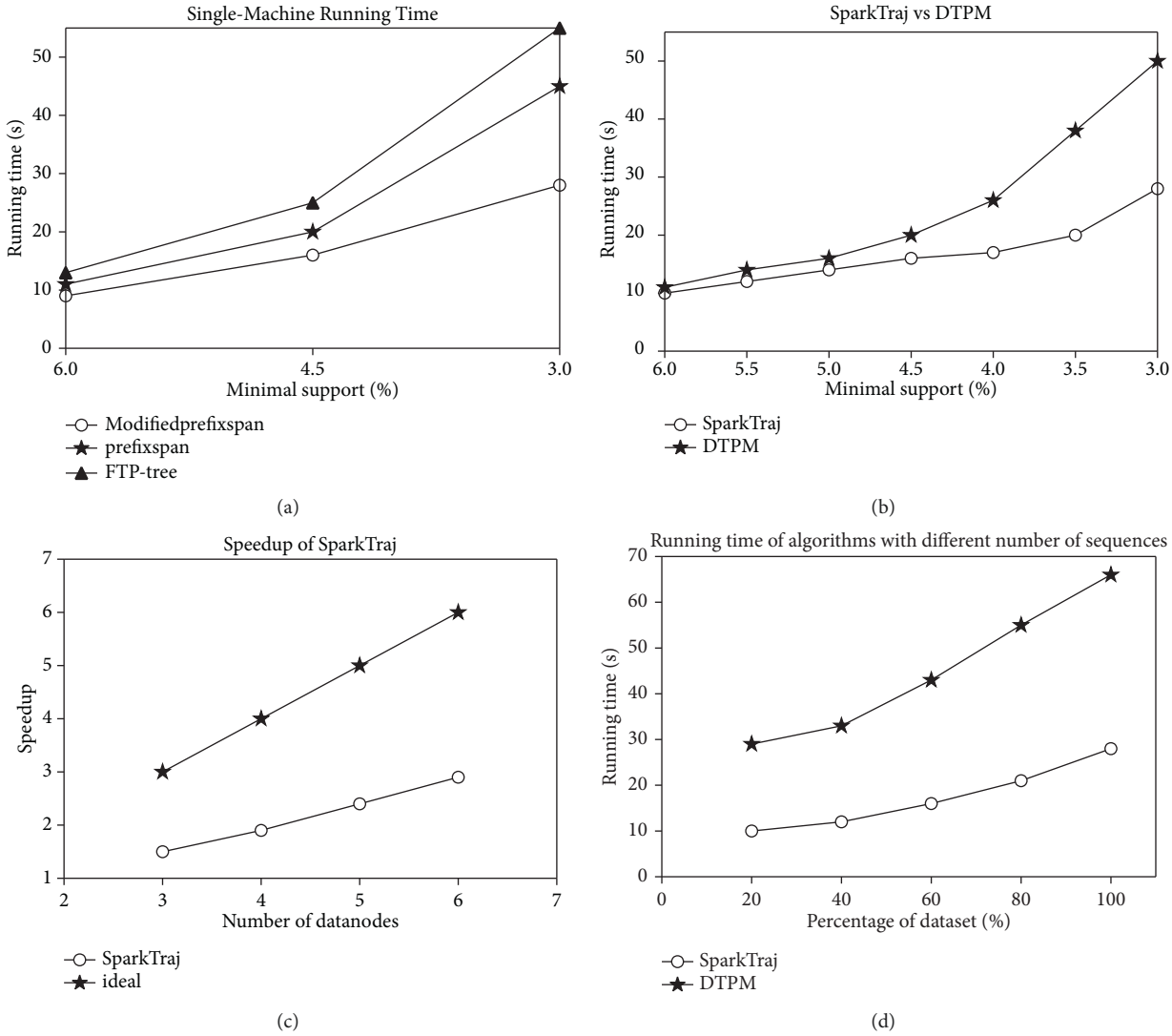

Figure 8: Experimental results (a) single machine efficiency comparison (b) distributed efficiency comparison (c) algorithm speedup (d) algorithm scalability.

algorithm, and FTP-tree-based frequent trajectory mining algorithm [22] proposed in this study based on the improved adjacency attributes under different minimum support degrees. The experimental results are shown in Figure 8(a).

(1) Results analysis. It can be seen from the figure that as the minimum support continues to decrease, the runtime of the algorithm proposed in this study and the FTP-tree-based mining algorithm increases because as the minimum support decreases, the trajectory patterns found gradually increase. Under the same minimum support degree, the PrefixSpan algorithm will perform a large number of prefix sequence connections and improve the mining efficiency, but the algorithm in this study modifies the prefix pattern connection process to avoid generating redundant trajectory-pattern projections, thereby significantly improving overall performance. However, the FTP-tree-based mining algorithm generates a large number of branches in the process of tree building due to the generation of a large number of candidates, which takes up more memory space. Therefore, as the minimum support degree decreases, the difference in efficiency between the two becomes more obvious. The PrefixSpan algorithm improved based on the adjacent pruning strategy and maintains a more efficient mining efficiency while ensuring that the mining results remain unchanged.

4.3.2. Experiment 2. In this experiment, we analyzed the runtime of the SparkTraj and DTPM algorithms [8] in the same cluster as Experiment 1 under different minimum support. The experimental results are shown in Figure 8(b).

(1) Results analysis. It can be seen from the figure that the runtime of the SparkTraj and DTPM algorithms increase as the minimum support decreases because as the minimum support decreases, more trajectory patterns can be found. However, both the SparkTraj and DTPM algorithms use

memory computing to load the trajectory sequence data into the RDD, reducing the total I/O and network transmission time, and both improve the mining efficiency. However, the SparkTraj algorithm does not generate candidate sequences, does not scan the sequence set multiple times, and modifies the prefix pattern generation process at the same time. Therefore, as the minimum support is reduced, the performance of SparkTraj is more stable.

4.3.3. Experiment 3. In this experiment, we tested the acceleration effect of different numbers of data nodes on SparkTraj. The minimum support setting is 0.5%. Speedup represents the speed at which performance is improved when additional nodes are added to the cluster. The experimental results are shown in Figure 8(c).

(1) Results analysis. It can be seen from the results that SparkTraj has good acceleration performance because the number of trajectory sequences processed by each node decreases as the number of data nodes increases. However because the runtime is affected by the data transmission between machines, the increase in speed is not completely positive considering the number of data nodes.

4.3.4. Experiment 4. In this experiment, we used five datasets, each of which contains 20%–100% of the original datasets, with a minimum support of 0.3%. We analyzed the runtime of the SparkTraj and DTPM algorithms under datasets of different scales in the same cluster (one master node and six data nodes). The experimental results are shown in Figure 8(d).

(1) Results analysis. It can be seen from the figure that the runtime of the DTPM algorithm will increase proportionally with the size of the datasets. However, for SparkTraj, unlike the DTPM algorithm, which generates a large number of candidate sequences, the runtime of SparkTraj only slightly increases as the size of the datasets increases. In addition, modifying the candidate trajectory pattern generation process also improves the overall performance of the SparkTraj algorithm.

5. Conclusions

A distribution-based frequent trajectory mining framework is proposed in this study that preprocesses the original trajectory data and converts GPS position coordinates into trajectories represented by line segments, thereby avoiding boundary problems and enhancing the applicability of the discovered trajectory patterns. The framework makes full use of the characteristics and advantages of Spark and RDD to reduce I/O and communication costs. Lesser redundant suffix datasets are generated following the prefix pruning strategy and improves the mining efficiency. Experimental results on a public dataset and the actual datasets show that the proposed SparkTraj algorithm is reasonable and effective. Planned future research directions include (1) load balancing of distributed clusters and (2) combining semantic trajectory data to provide more readable mining results.

Data Availability

Data description: This GPS trajectory dataset was collected in (Microsoft Research Asia) Geolife project by 182 users over a period of over five years (from April 2007 to August 2012). A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information on latitude, longitude, and altitude. This dataset contains 17,621 trajectories with a total distance of 1,292,951 kilometers and a total duration of 50,176 hours. These trajectories were recorded by different GPS loggers and GPS phones, and have a variety of sampling rates. About 91.5 percent of the trajectories are logged in a dense representation, e.g. every 1–5 seconds or every 5–10 meters per point. This dataset recorded a broad range of users' outdoor movements, including not only life routines like going home and going to work, but also some entertainment and sports activities, such as shopping, sightseeing, dining, hiking, and cycling. This trajectory dataset can be used in many research fields, such as mobility pattern mining, user activity recognition, location-based social networks, location privacy, and location recommendation. https://www.microsoft.com/en-us/research/project/geolife-building-social-networks-using-human-location-history/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fprojects%2Fgeolife%2Fdefault.aspx.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors thank LetPub (http://www.letpub.com) for its linguistic assistance during the preparation of this manuscript.

References

[1] Y. Zheng, "Trajectory data mining," *ACM Transactions on Intelligent Systems and Technology*, vol. 6, no. 3, pp. 1–41, 2015.

[2] H. Cao, N. Mamoulis, and D. W. Cheung, "Discovery of periodic patterns in spatiotemporal sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 4, pp. 453–467, 2007.

[3] M. Xu, J. Wu, M. Liu, Y. Xiao, H. Wang, and D. Hua, "Discovery of critical nodes in road networks through mining from vehicle trajectories," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 1–11, 2019.

[4] X. Ying, W. Ruidi, Z. Xu, and R. Wenliang, "Grid-based k-nearest neighbors query over moving trajectories under spark," *Journal of Chongqing University of Posts and Telecommunications(Natural Science Edition)*, vol. 31, no. 04, pp. 531–537, 2019.

[5] J. Zhu, C. Huang, M. Yang, and G. P. Cheong Fung, "Context-based prediction for road traffic state using trajectory pattern mining and recurrent convolutional neural networks," *Information Sciences*, vol. 473, pp. 190–201, 2019.

[6] S. Qiao, C. Tang, S. Dai, Z. Mingfang, and P. Jing, "PartSpan: parallel sequence mining of trajectory patterns," in

Proceedings of the International Conference on Fuzzy Systems & Knowledge Discovery, October 2008.

[7] S. Qiao, T. Li, J. Peng, and J. Qiu, "Parallel sequential pattern mining of massive trajectory data," *International Journal of Computational Intelligence Systems*, vol. 3, no. 3, pp. 343–356, 2010.

[8] J. Liu, X. Yu, Z. Xu, and R. C. Kim-Kwang, "A cloud-based taxi trace mining framework for smart city[J]," *Software: Practice and Experience*, vol. 47, no. 8, pp. 1081–1094, 2016.

[9] L. Shen, J. Lu, D. Geng, and D. Ling, "Peak traffic flow predictions: exploiting toll data from large expressway networks[J]," *Sustainability*, p. 13, 2020.

[10] M. Lv, Y. Li, Z. Yuan, and K. Zhou, "Route pattern mining from personal trajectory data[J]," *Journal of Information Science and Engineering*, vol. 31, no. 1, pp. 147–164, 2015.

[11] S. Karsoum, L. Gruenwald, and E. Leal, "Impact of trajectory segmentation on discovering trajectory sequential patterns," in *Proceedings of the 2018 IEEE International Conference on Big Data (Big Data)*, December 2018.

[12] J. Lee, J. Han, and K. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the international conference on management of data*, pp. 593–604, Beijing, China, June 2007.

[13] W. Gan, J. C. Wei Lin, P. Fournier-Viger, H. Chieh Chao, and P. S. Yu, "A survey of parallel sequential pattern mining," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 3, 2019.

[14] Y. Zheng, X. Xie, and W. Yingma, "GeoLife: a collaborative social networking service among user, location and trajectory [J]," *Invited paper, in IEEE Data Engineering Bulletin*, vol. 33, no. 2, pp. 32–40, 2010.

[15] Z. Fu, Z. Tian, Y. Xu, and Q. Changjian, "A two-step clustering approach to extract locations from individual GPS trajectory data[J]," *ISPRS International Journal of Geo-Information*, vol. 10, p. 5, 2016.

[16] R. W. Liu, M. Liang, J. Nie, W. Y. B. Lim, Y. Zhang, and M. Guizani, "Deep learning-powered vessel trajectory prediction for improving smart traffic services in maritime internet of things," *IEEE Transactions on Network Science and Engineering*, Article ID 3140529, 2022.

[17] R. W. Liu, J. Nie, S. Garg, Z. Xiong, Y. Zhang, and M. S. Hossain, "Data-driven trajectory quality improvement for promoting intelligent vessel traffic services in 6G-enabled maritime IoT systems," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5374–5385, Article ID 3028743, 2020.

[18] J. Chen, K. Li, Z. Tang et al., "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2017.

[19] Z. Fu, Z. Tian, Y. Xu, and C. Qiao, "A two-step clustering approach to extract locations from individual GPS trajectory data," *ISPRS International Journal of Geo-Information*, vol. 5, no. 10, p. 166, 2016.

[20] J. Chen, K. Li, H. Rong, K. Bilal, K. Li, and P. S. Yu, "A periodicity-based parallel time series prediction algorithm in cloud computing environments," *Information Sciences*, vol. 496, pp. 506–537, 2019.

[21] J. Chen, K. Li, K. Li, P. S. Yu, and Z. Zeng, "Dynamic bicycle dispatching of dockless public bicycle-sharing systems using multi-objective reinforcement learning," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 4, pp. 1–24, 2021.

[22] Q. ShaoJie, H. Nan, D. ZhiMing, J. Che-Qing, S. Wei-Wei, and S. Hong-Ping, "A multiple-motion-pattern trajectory prediction model for uncertain moving abjects[J]," *Acta Automatica Sinica*, vol. 44, no. 4, pp. 608–618, 2018.