traj2bits: Indexing Trajectory Data for Efficient Query

Rui Zhang[1], Jiming Guo[1], Yueqi Zhou[1], Hongbo Jiang[2,*], Chen Wang[3]
[1] Hubei Key Laboratory of Transportation Internet of Things, School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China, 430070, {zhangrui@whut.edu.cn, jimingguo@whut.edu.cn, zhouyueqi91@gmail.com}
[2] Sclool of Information Science and Technology, Hunan University, Changsha China, 410012, hongbojiang2004@gmail.com
[3] Sclool of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan China, 430074, chenwang@hust.edu.cn

ABSTRACT

With the popularity of mobile devices and the rapid development of position acquisition technology, the amount of trajectory data has soared dramatically. It is time-consuming to manage and mine massive trajectory data, because we need to access different trajectory samples or different parts of a trajectory for multiple times. Therefore, it is necessary to devise an efficient data management technology to fast retrieve the desired trajectories. In general, building indexes is a basic step for solving query problems. However, traditional spatial indexing technologies are mostly designed for moving objects, and thus are unable to achieve fast trajectory data query and efficient computing analysis. In this regard, we propose traj2bits, a bitmap-based trajectory data encoding schema, to convert trajectories into binary strings. Based on traj2bits, we also design a trajectory query method. Experiments on two real datasets have shown that traj2bits improves the spatio-temporal efficiency of trajectory query. Compared with other schemes, traj2bitsuery encoding occupies less than 1/10 of the disk space, its encoding efficiency is at least four times faster and its range query time is reduced by at least 65%.

KEYWORDS

trajectory data, query, index, bitmap

ACM Reference Format:
Rui Zhang[1], Jiming Guo[1], Yueqi Zhou[1], Hongbo Jiang[2,*], Chen Wang[3]. 2019. traj2bits: Indexing Trajectory Data for Efficient Query. In *ACM Turing Celebration Conference - China (ACM TURC 2019) (ACM TURC 2019), May 17–19, 2019, Chengdu, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3321408.3321578

*This is the corresponding author

ACM TURC 2019, May 17–19, 2019, Chengdu, China

ACM ISBN 978-1-4503-7158-2/19/05…$15.00
https://doi.org/10.1145/3321408.3321578

1 INTRODUCTION

Along with the growth of mobile computing and position-based services, the growth of positioning data and more accurate positioning information has been increased and attracted the research on positioning data as well as accumulated massive amounts of trajectory data. For example, Moll et al. point out that even a small research team working on autonomous driving can accumulate trillions of pieces of sensor data through a few trips and multiple vehicles [3]. Zhang et al. also state that Didi, China's largest ride-hailing platform, generates more than 10 million trajectories every day[7]. Many types of valuable information, such as behavior pattern and traffic pattern, can be obtained by mining these trajectories. However, it is very time-consuming to mine a large amount of trajectories, because we need to access different trajectories or the different parts of a trajectory for multiple times [8]. It requires effective data management techniques that can quickly retrieve the desired trajectories. Traditional databases often use spatial query plug-ins to manage spatial data. For example, Sheng et al. use PostGIS to compress data size for ease of storage[5]. However, performing query analysis on trajectory data needs databases that can support spatio-temporal data management. Meanwhile, it is difficult for relational databases to fast query billions of lines of trajectory sampling points. In addition, since the distribution of trajectory data is different from that of general spatial data, the universal spatial index R-tree is unsuitable for processing trajectory data. Therefore, STR-tree, TB-tree [4] and other solutions have been proposed according to trajectory characteristics. On the other hand, many models specially designed for trajectory query only solve specific query problems, such as popular-routes[6]. Therefore, their indexing scheme and query operation are inefficient and less universal for trajectory data.

Moreover, index-based query methods suffer from disk space usage problem. As traditional relational databases do not involve data compression, the index data could be excessively large. For example, although PostGIS successfully implements the R-tree spatial index, yet it could only build indexes for sampling points or geometric figures due to the fact that it is unable to process trajectory data in a customized manner. Consequently, trajectory query solutions based on these databases suffer from various problems, such as slow query response, and excessive data size. Although spatio-temporal databases provide more comprehensive support for spatial relation computation, the computational costs and other issues could be very prominent in case of massive data. Therefore, it

is a challenging problem to implement a trajectory-oriented range query operation under low resource occupancy.

To solve the above-mentioned problems, we propose traj2bits and our main contributions are as follows:

1. We have proposed traj2bits, a bitmap-based trajectory encoding scheme. Different from the traditional indexing sampling points schemes, our encoding method converts trajectories to binary strings to support spatial filtering (intersect) operation. Its also occupied space and conversion time do not increase as the the number of sampling points.

2. Based on traj2bits, we have put forward a method for querying trajectory range. During range query, it first filters the spatial range before filtering temporal range.The scheme has good performance, because it makes full uses of the high efficiency of bit operation.

3. We have evaluated our scheme on two real datasets, and the results show that traj2bits is superior to other schemes in terms of encoding spatio-temporal efficiency and range query.

The rest of this paper is organized as follows. Section 2 introduces the bitmap-based trajectory coding schema. In Section 3, the method for range querying is presented. Section 4 gives an evaluation of our proposed method on two real datasets. Finally, the whole paper is summarized in Section 5.

2 BITMAP-BASED TRAJECTORY ENCODING

This section gives the definitions and mathematical description of trajectory encoding method, presents the implementation of traj2bits, and explains how to use our method to compute spatio-temporal trajectory data, including the query operations oriented to spatial relations: Intersect and Contains.

2.1 Trajectory Model

The trajectory shows a continuous motion process of a moving object. Moreover, it also stores as a series of sampling spatio-temporal points. Therefore, we define the trajectory as follows:

Definition 1:Trajectory . Trajectory is a continuous sequence made up of (x, y, t) , where (x, y) are the points under space coordinates and t represents sampling time. (x_i, y_i, t_i) means that the moving target is in position (x_i, y_i) at the time of t_i.Trajectory can be represented as $Trajectory = [(x_1, y_1, t_1) \cdots (x_i, y_i, t_i) \cdots (x_n, y_n, t_n)]$,where $t_1 < t_i < t_n$.

A trajectory represents a long period of motion process. If we look at a part of a trajectory, then in a certain time period $[t_i, t_j]$, the relationship between this part of the trajectory and the overall motion process can be denoted by a sub-trajectory or trajectory segment. Hence, we define the sub-trajectory or trajectory segment as follows:

Definition 2: sub-trajectory (SubTraj). Within a specified time period $[t_i, t_j]$, a part of the motion process made up of sampling points belonging to a trajectory is called a sub-trajectory. If the total number of sampling points in a trajectory is n , then a sub-trajectory can be denoted by $SubTraj(i, j) = [(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}) \cdots (x_j, y_j, t_j)]$,where $1 \le i \le j \le n$.

Definition 3: Trajectory segment. The line segment between any two neighboring sampling points is called a trajectory segment, which is denoted by TS. If the total number of sampling points in a trajectory is n , then $TS = Trajectory(i, i + 1)$, where $1 \le i < n$ .TS is the shortest trajectory segment. Since trajectories and sub-trajectories often span a wide space, we need to divide a space (geographical positions that a trajectory crosses through) into a number of smaller regions in order to fully describe each sampling point of the trajectory. For ease of description, we define *Grid* and *GID* as follows:

Definition 4: Grid and Grid ID (GID).*Grid* is a region within a fixed region under under the space rectangular coordinate system. The whole space can be covered by a finite number of identical *Grid* , and none of *Grid* are overlapped. The side length of *Grid* is denoted by δ . Each *Grid* has a unique ID, which is named as *GID*.

We use GeoHash algorithm [1] to divide the space so that it can maintain a unified division granularity under global coordinate system (longitude range$[-180, 180]$, and latitude range $[-90, 90]$) with simple parameters. The sampling frequencies and intervals of the trajectory sampling points are different. When the distance and sampling time between two neighboring sampling points are greater than critical condition, the trajectory recognition algorithm will divide the sampling point sequence into independent trajectories [2] according to this critical condition. When sampling interval and other factors all meet this critical condition, then traj2bits has to process the relationship between sampling point distance and *Grid* so that the *GID* passed through is adjacent and continuous. Traj2bits assumes that the motion in the middle of trajectory segment (TS) is rectilinear motion.

2.2 Trajectory Encoding

In this section, we describe the method for implementing trajectory encoding. First, the problem formation is as follows:

Problem 1: Trajectory encoding (Grid Encoding). Given a trajectory $Traj$, we need to convert all of its sampling points into a set of bitmap-structure data, where the elements of the set is the *GID* of each *Grid* . Let the set be *GE* , then after computation, *GE* can be expressed as $GE(Traj) = [GID_i, GID_{i+1} \cdots GID_j]$, where $1 \le i \le j \le n$.

The first step of trajectory encoding is to ensure the continuity of trajectory sequence on *Grid* . Generally, the range of *Grid* is greater than the distance between two neighboring sampling points. However, exceptions may also exist. In this case, the *Grid* corresponding to the two neighboring sampling points is discontinuous, which will affect trajectory encoding. Therefore, we use $TSP(\delta)$ to solve this problem by adjusting the parameter δ and enlarging the region of *Grid* , where δ is the side length of *Grid* . In other words, by adjusting the *Grid* according to δ , we can ensure that the distance between any two neighboring points is smaller than δ . After ensuring the continuity of the trajectory sequence on *Grid* , we need to find out the region division corresponding to each sampling points. In other words, we need to find the *Grid* where each sampling point resides. We directly use GeoHash algorithm to compute the character code corresponding to each sampling point, and denote it by a unique integer. Each integer is the *Grid* corresponding to each sampling point. The detailed algorithm steps are as algorithm 1.

After the character code corresponding to each point is obtained, Bitmap can be used for compression storage. Therefore, the complete solution is as follows: First, the trajectory is divided into

Algorithm 1 Grid_contains_TS

Input: Trajectory segment TS , $Grid$ side length δ
Output: List made up of $Grid$ codes

```
glist = list()
for point in TSP(δ) do
    hash = GeoHash(point,δ))
    gid = GID_MAP(hash)
    glish.add(gid)
end for
return glist
```

continuous trajectory segments and traversed one by one. Second, the ID of the region corresponding to each trajectory segment is computed and stored into the set. Finally, the set of region IDs are aggregated and de-duplicated and stored into the Bitmap. The specific implementation of trajectory encoding algorithm is as algorithm 2.

Algorithm 2 Grid_Encoding

Input: Trajectory T , $Grid$ side length δ
Output: $GE(T)$

```
ge = list()
for each ts in T do
    grid_ids = Grid_contains_TS(ts,δ)
    ge.add(grid_ids)
    glish.add(gid)
end for
return bitmap(ge.removeDuplicate)
```

Finally, after trajectory encoding is complete, we can design some basic operations based on this encoding strategy, such as *Intersect* , *Contain* and so on. *Intersect* computes the number of overlapped *Grid* between GE_A and GE_B corresponding to the two trajectories. If the number is 0, then the two trajectories do not overlap at all. Otherwise, the two trajectories intersect. *Contain* computes the intersection between GE_A and GE_B , GE_B completely belongs to GE_A , which means *Contain* meets $Intersect(GE_A, GE_B) = len(GE_A)$. Hence, *Contain* need only to add one more judgment based on *Intersect* . For ease of understanding, we give the detailed implementation steps of *Intersect* as algorithm 3.

Algorithm 3 Intersect

Input: GE_A , GE_B
Output: intersect_length

```
length=len(GE_A & GE_B )
return length
```

3 QUERY BASED ON TRAJECTORY ENCODING:

This section describes traj2bit-based range query as well as its implementation.

Range query is a basic method of trajectory data analysis. As shown in Figure 1, the red box indicates the range query region R . It can be seen that the trajectories T_1 and T_3 reside in R thus it can be queried while the trajectory T_2 is not in R so it cannot.

Here are the followings notations used in this research paper:

Definition 5 Query range (QueryBox). Given a query condition, we need to determine the maximum range $[X_1, X_2]$ and $[Y_1, Y_2]$ and time range $[t_1, t_2]$. Then the query range is $QueryBox = (x, y, t) \mid x \in [X_1, X_2], y \in [Y_1, Y_2], t \in [t_1, t_2]$.

Definition 6: Search range (CoverageBox). Given a query range, there is a mapping function H and a set containing non-repetitive codes so that $CoverageBox = Set(\{H(x, y) \mid (x, y) \in QueryBox\})$, where H is GeoHash algorithm, and the items in the set are the calculated values of the mapping function.

By definition, there are not duplicate items in CoverageBox, and it only contains spatial range conditions. It inclusively processes the regions corresponding to QueryBox of range query according to the preset region division. Since traj2bits has determined the spatial division regions when processing the trajectory data, the query conditions need to be processed with the same parameters. The diagram of the two ranges is shown in Figure 2. Traj2bits first processes spatial conditions and converts $QueryBox$ to $CoverageBox$.

The input parameters of the algorithm Traj2bits-Range-Scan are the spatio-temporal range query condition QueryBox. The returned data is in the format of trajectory chain table. The steps are as follows. First, we compute the intersection between $QueryBox$ and all $Grid$ to obtain $CoverageBox$. Second, the set of Grid_Encoding storing all trajectories is traversed. Intersect computation is performed on trajectories and CoverageBox. If a trajectory has an intersection with CoverageBox, then it will be added to the set of candidate results. Finally, the trajectories in the candidate set are expanded into a sequence of sampling points. For each sampling point, the inclusion relation between QueryBox and sampling points is checked, including time range. The detailed steps of the algorithm are as algorithm 4.

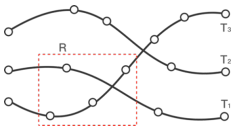


Figure 1: Query range of trajectories.

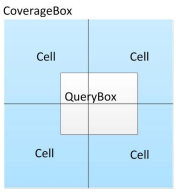


Figure 2: $QueryBox$ **and** $CoverageBox$

According to the algorithms, we can see that Intersect is based on bit operation with the time complexity is O(1). Then, it narrow the time complexity search range in O(n), where n is the number of trajectories. Last, the total time complexity depends on the number of sampling points in the trajectory m, so the total time complexity is O(m).

4 EXPERIMENT

This section evaluates the performance of traj2bits on several real datasets. First, we briefly describe the datasets, then evaluate the performance of trajectory encoding and trajectory range query.

Algorithm 4 traj2bits_range_scan

Input: *QueryBox*
Output: List(T)
1: tList=list()
2: *CoverageBox* = *H*(*QueryBox*)
3: **for** each ge in GE_Store **do**
4: **if** Intersect(CoverageBox, ge) > 0 **then**
5: tList.add(ge.T)
6: **end if**
7: **end for**
8: tList.filter(t ⇒ queryBox.contains(t.points))
9: **return** tList

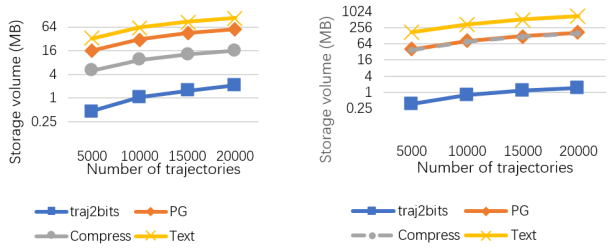
4.1 Datasets

Traj2bits is evaluated on two real datasets. The first is derived from GeoLife [9–11] , publicly available Beijing taxi data downloaded from the Internet. The dataset has 5484 taxies and 4375068 instances, which is abbreviated to TAXI. In this dataset, an independent text file is established according to taxi IDs, storing the trajectory information about each taxi. Each instance in the file records a sampling position point. In the experiment, day is used as the interval, and sampling points are combined as the trajectories of taxies. The second dataset is non-public maritime AIS data, including a total of 16371055 instances generated by 8,506 ships, which is abbreviated to SHIP. The dataset records the AIS information by day. Each instance in the file contains a sampling position point and velocity, sailing angle, etc. All instances are recorded in chronological order, so the extraction of trajectories is mainly to sort and combine the records. TAXI data is processed in the same way, where the extraction of trajectories is done on a daily basis. The average trajectory lengths of TAXI dataset and SHIP dataset are 150 km and 30 km, respectively. On average, each TAXI trajectory and each SHIP trajectory are made up of 138 and 836 sampling position points, respectively. Therefore, the average distance of TAXI trajectories is larger than the average distance of SHIP.

4.2 Trajectory encoding efficiency

The primary performance measurements are disk space usage and execution time. These measurements can reflect the space and time costs as well as the response speed of traj2bits-based schemes. The comparison scheme is designed based on PostGIS spatio-temporal database. In traj2bits scheme, the parameter of the control range set by Grid_Encoding is 5, which means that the code length generated by GeoHash is 5. The 10,000-square-kilometer rectangular region is divided into 667 subregions. In TAXI dataset, there are 486 subregions. All experiments are conducted on a laptop with 2.6 GHz Intel i5 dual-core CPU , 8GB RAM, and 256GB SSD.

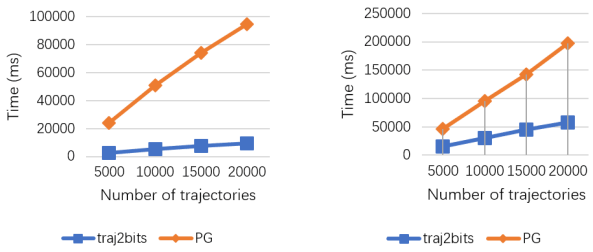
First, we compare the disk space usage of each method. Figure 3 shows the disk space usage of trajectory encoding implemented by each method. As is shown in Figure 3(a), in TAXI dataset, traj2bits takes up only 1/20 the storage volume in the PostGIS (PG) scheme. In Figure 3(b), under SHIP dataset, the performance difference between traj2bits and PostGIS-based schemes is more obvious. The disk space usage of traj2bits is only 1% that of PostGIS. In addition, according to the characteristics of datasets, the average trajectory length in TAXI dataset is greater than that in SHIP dataset. The experimental results show that the traj2bits creates greater data volume under TAXI dataset than SHIP dataset. Given the same number of trajectories, the main influencing factor for traj2bits disk space usage is trajectory length. Figure 3 also shows the relationship between query scheme and data volume. Trajectory data can be compressed and stored in Apache Parquet compression format so that the data size can be reduced to 1/4 or even less of the original data size. The ratio of the volume occupied by traj2bits to the original data in the compression format is about 1:10 in TAXI dataset and nearly 1:100 in SHIP dataset.



(a) TAXI (b) SHIP

Figure 3: Occupied Space

Second, we compare the computation time of each method to reflect the computational complexity required by the scheme. Figure 4 shows the time cost of the traj2bits-based Grid_Encoding process. As the number of trajectories increases, the time difference between traj2bits computation time and PostGIS computation time becomes greater. Under TAXI and SHIP, traj2bits requires only 1/9 and 1/4 the computation time of PostGIS scheme, respectively. Given the same number of trajectories, it takes longer for traj2bits to process SHIP dataset than TAXI dataset. Moreover, as the number of trajectory sampling points in SHIP dataset is greater than TAXI dataset, the main influencing factors for traj2bits conversion time is the number of trajectory position points rather than trajectory length. Under the same dataset, traj2bits computation time shows a linear relationship with the number of trajectories.



(a) TAXI (b) SHIP

Figure 4: Traj2bits conversion time

Finally, we use Intersect operation to compare the computing efficiency of each method. Figure 5 shows the time it takes for each method to perform Intersect operation under TAXI dataset. It can be seen that the traj2bits-based Intersect operation is at least

20 times faster than the PostGIS-based scheme, and increase the number of trajectories. Tra2jbits is based on bitmap computation thus the actual computation efficiency is very high. Through the experiments above, we can see that traj2bits has obvious advantages in time and space than PostGIS-based solutions, and outstanding performance in basic operations.

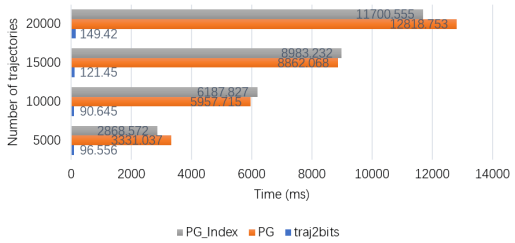
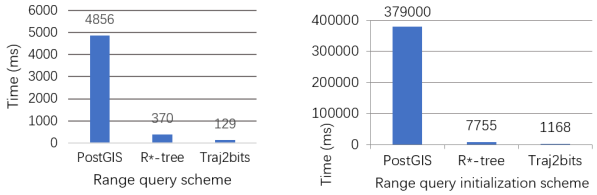


Figure 5: Intersect operation under TAXI dataset

4.3 Range query efficiency

We evaluate the performance and availability of traj2bits Range Query algorithm. The test dataset contains 1056,474 trajectory points. After pre-processing, we can obtain 7524 trajectories. Two other schemes are used for comparison with the detail environment as follows: 16GB RAM, 2.60GHz quad-core CPU, and a mechanical hard disk as storage medium. The first is v2.2.1 PostGIS. Trajectory points are stored in lines. PostGIS's range query statement is implemented based on ST_Cover. The second scheme is based on R * -tree of full memory storage to implement range query. Data and memory are both stored in RAM, and there is no disk I/O during range query. The relevant query method of traj2bits is implemented based on the distributed data interfaces of Spark . However, in our experiment, Spark's standalone node deployment mode is configured. The operating environment is a 2.60GHz quad-core CPU, 10GB RAM and a 200GB mechanical hard disk. The driver-memory for configuring Spark is 5GB.

Figure 6 shows the time of range query and initialization. For the query of the same data, although PostGIS also uses R-tree index, but the internal database system may be affected by the disk I/O during range query. Therefore, the R * -tree scheme can significantly reduce the query time. However, for the same computing task in memory, traj2bits in Figure 6(a) provides a more suitable query structure for analyzing trajectory data, which can complete trajectory range query faster. Traj2bits query time is 65% less than R * -tree and 97% less than PostGIS. As can be seen from Figure 6(b), the query speeds of all three query schemes meet the common query requirements. However, high initialization cost would hamper the availability of query schemes. The operating environment for PostGIS scheme in last section uses the SSD as storage medium, but the experiment in this section uses a mechanical hard disk. For initialization, it takes PostGIS scheme 379s to import 33MB data. As R * - tree and traj2bits store all data in RAM, their RAM-based initialization is nearly 50 times faster than PostGIS scheme. Traj2bits initialization time is 84% that of R * -tree.



(a) Range query (b) Initialization

Figure 6: Time of range query and initialization

5 CONCLUSION

This paper proposes a trajectory binary coding schema named traj2bits and design trajectory query methods based on it. Our proposed encoding and query schemes achieved good spatio-temporal efficiency. Experiments are conducted to evaluate the performance of traj2bits on two real trajectory data.We hope to apply traj2bits to the big data environment and implement a distributed version of traj2bits in future. Meanwhile, we will extend traj2bits for indexing multi-dimensional trajectory information to improve trajectory data denoising under more scenarios.

REFERENCES

[1] Anthony D. Fox, Christopher N. Eichelberger, James N. Hughes, and Skylar Lyon. 2013. Spatio-temporal indexing in non-relational distributed databases. In *BigData*. IEEE, 291–299.

[2] Luca Leonardi, Gerasimos Marketos, Elias Frentzos, Nikos Giatrakos, Salvatore Orlando, Nikos Pelekis, Alessandra Raffaetà, Alessandro Roncato, Claudio Silvestri, and Yannis Theodoridis. 2010. T-Warehouse: Visual OLAP analysis on trajectory data. In *ICDE*. IEEE Computer Society, 1141–1144.

[3] Oscar Moll, Aaron Zalewski, Sudeep Pillai, Sam Madden, Michael Stonebraker, and Vijay Gadepally. 2017. Exploring Big Volume Sensor Data with Vroom. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1973–1976. https://doi.org/10.14778/3137765.3137822

[4] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. 2000. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 395–406. http://dl.acm.org/citation.cfm?id=645926.672019

[5] Kai Sheng, Zefang Li, and Dechao Zhou. 2015. A Storage Method for Large Scale Moving Objects Based on PostGIS. In *ITITS (1) (Advances in Intelligent Systems and Computing)*, Vol. 454. 623–632.

[6] Ling-Yin Wei, Wen-Chih Peng, and Wang-Chien Lee. 2013. Exploring Pattern-aware Travel Routes for Trajectory Search. *ACM Trans. Intell. Syst. Technol.* 4, 3, Article 48 (July 2013), 25 pages. https://doi.org/10.1145/2483669.2483681

[7] Zhigang Zhang, Cheqing Jin, Jiali Mao, Xiaolin Yang, and Aoying Zhou. 2017. TrajSpark: A Scalable and Efficient In-Memory Management System for Big Trajectory Data. In *APWeb/WAIM (1) (Lecture Notes in Computer Science)*, Vol. 10366. Springer, 11–26.

[8] Yu Zheng. 2015. Trajectory Data Mining: An Overview. *ACM Trans. Intell. Syst. Technol.* 6, 3, Article 29 (May 2015), 41 pages. https://doi.org/10.1145/2743025

[9] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding Mobility Based on GPS Data. In *Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp '08)*. ACM, New York, NY, USA, 312–321. https://doi.org/10.1145/1409635.1409677

[10] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.

[11] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining Interesting Locations and Travel Sequences from GPS Trajectories. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. ACM, New York, NY, USA, 791–800. https://doi.org/10.1145/1526709.1526816