Detect Tracking Behavior Among Trajectory Data

Jianqiu Xu(✉) and Jiangang Zhou

Nanjing University of Aeronautics and Astronautics, Nanjing, China
{jianqiu,jiangangzhou}@nuaa.edu.cn

Abstract. Due to the continuing improvements in location acquisition technology, a large population of GPS-equipped moving objects are tracked in a server. In emergency applications, users may want to detect whether a target is tracked by another object. We formulate the tracking behavior by continuous distance queries in trajectory databases. Index structures are developed to improve the query performance. Using real trajectories, we demonstrate answering continuous distance queries in a database system and animating moving objects fulfilling the distance condition in the user interface. The result benefits mining the interesting behavior among trajectory data and answering distance join queries.

1 Introduction

The rise of GPS-equipped mobile devices has led to the emergence of big trajectory data. A large amount of historical trajectory data has become available for emerging applications such as traffic monitor and location-based services. Real time tracking has been studied to efficiently track an object's trajectory in real-time [12]. However, so far little attention has been paid to detect the tracking behavior over historical movement. That is, given a query trajectory o_q and a distance threshold d, we aim to find whether there is any object keeping some distance to o_q at each time point of o_q. We formulate the problem by continuous distance queries over trajectory data in which users define a threshold to return objects within a distance (e.g., 500 m) to the query.

Consider an example in Fig. 1. There are five trajectories $\{o_1, o_2, o_3, o_4, o_5\}$ and the task is to find whether o_3 is tracked by another object during $[t_1, t_2]$. Suppose that a short distance d is defined, e.g., 500 m. In the example, we will find o_4 following o_3. Other objects are within the distance to o_3 but only for a short time period. This will not be treated as tracking behavior. To find the *tracker*, we need to compute the time-dependent distance between trajectories and determine whether the period fulfilling the distance condition contains each time point of the query trajectory. This is not a simple task because the precise distance needs to be determined. One may think of using the continuous nearest neighbor query to find the tracking behavior, but such a query will return objects with small distances at each piece of time. We need to figure out the precise distance and find whether there are objects that always keep a certain distance to the target rather than only at a short period.

G. Cong et al. (Eds.): ADMA 2017, LNAI 10604, pp. 872–878, 2017.
https://doi.org/10.1007/978-3-319-69179-4_64

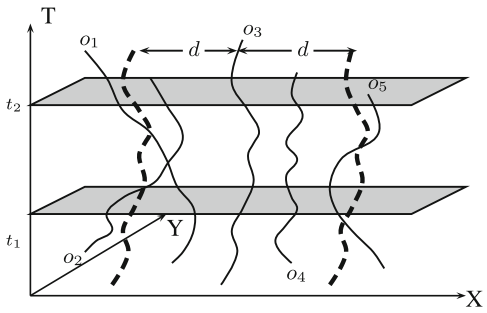


Fig. 1. Detect tracking behavior

In the literature, a lot of effort has gone to process k nearest neighbor queries in which a distance function is defined to return k objects with the smallest distance to the target. Due to diverse spatial environments, e.g., free space [8], road network [13] and obstacle space [5], different distance functions are used. The distance is a static value if a time point is considered and becomes a time-dependent function if the continuous query over a time period is considered. The latter is non-trivial because the distance varies over time such that returned objects change at certain points. In some special applications, the tracking behavior is helpful to find interesting objects that cannot be discovered by nearest neighbor queries. For example, to look for the tracker or travelers with common routes, we would like to search the object that always keeps a certain distance to the target rather than the nearest object at certain time points. On one hand, the nearest object to the target may change at some points or places. On the other hand, the nearest object is usually not the tracker because of being easily exposed.

In the demo, we detect the tracking behavior by using continuous distance queries. To be general, the query is of three forms in terms of distance parameters. We are able to search objects within or out of a certain distance to the target. To efficiently answer the query, several trajectory indexes are provided in the system such as 3D R-tree and TB-tree. We propose a novel index structure that combines the grid index and R-tree. Given a query trajectory, one quickly calculates the cells within the query distance. During the R-tree traversal, we make use of the index structure to prune the space that cannot contribute to the result. Real datasets are used in the demonstration scenario for continuous distance queries and distance join queries.

The rest of the paper is organized as follows. We review the related work in Sect. 2, formulate the problem and introduce the solution in Sect. 3, demonstrate continuous distance queries in Sect. 4, followed by conclusions in Sect. 5.

2 Related Work

In the literature, tremendous efforts have been made on querying trajectories including nearest neighbors [6,8], similarity search [4,17] and pattern

discovery [10,14]. Those works focus on searching some targets *close* to the query, but do not consider a precise distance function to evaluate the query. *Calibrating* trajectory data is studied in [17], the goal of which is to transform heterogeneous trajectories to one with unified sampling strategies. Discoverying convoys in trajectory databases is to return groups of objects such that each group consists of a set of density-connected objects which should satisfy the distance requirement over a certain time period. Trajectory clustering [9,11,15] also aims to form groups such that (i) points within the same group are close to each other, and (ii) points from different groups are far apart. However, the distance in convoy and cluster queries is calculated at each time point but not a time-dependent function as we use for processing continuous queries. In addition, discovery of convoys and clusters in trajectory database shows the behavior for a group of objects. In contrast, we aim to find whether a target is tracked by another object, which is an *individual* behavior.

The *closest-point-of-approach join* is studied in [2,18], the task of which is to return all object pairs. Each pair of trajectories has the distance less than a threshold at some point in time. This is not a continuous query and cannot be used to detect the tracking behavior. Two objects close to each other at some point in time do not mean that one is tracked by another. Probably this is treated as *passby* or *overtaken*. We generalize the distance query by considering a time interval such that the distance between two trajectories is less than a certain value at each time point during the query.

3 The Framework

3.1 Problem Definition

Let the database $\mathcal{O}$ be a set of trajectories, each of which consists of a sequence of temporal units. Each unit represents the movement over a time interval by recording start and end locations. Locations between them are estimated by interpolation. Given two trajectories $o_1, o_2 \in \mathcal{O}$, we let $dist(o_1, o_2, t)$ return the distance between o_1 and o_2 at time point t.

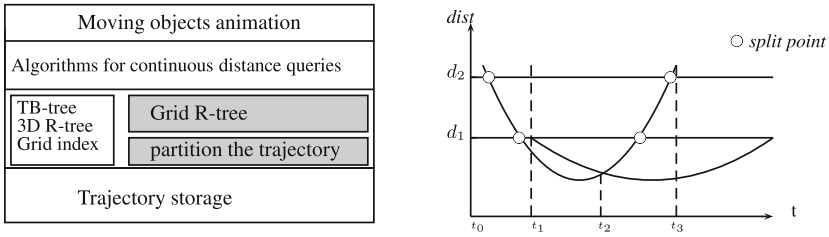
Definition 1 *(Tracking behavior). Let $T(o)(o \in \mathcal{O})$ return the time period of a trajectory. Given a query trajectory $o_q \in \mathcal{O}$ and a distance d, we define o_q's tracker by*

$$Tracker(o_q) \in \mathcal{O} : \forall t \in T(o_q), dist(o_q, Tracker(o_q), t) < d$$

We generalize the query by defining a distance range, i.e., $d = [d_1, d_2]$. If $d_1 = 0$ and $d_2 > 0$, this is used to find the tracking behavior. If $0 < d_1 < d_2$, we can return objects between d_1 and d_2 to the target which may also be trackers. If $d_1 > 0$ and $d_2 = \infty$, objects that are further than a certain distance to the target are found and can be used to exclude suspicious objects.

3.2 The Solution

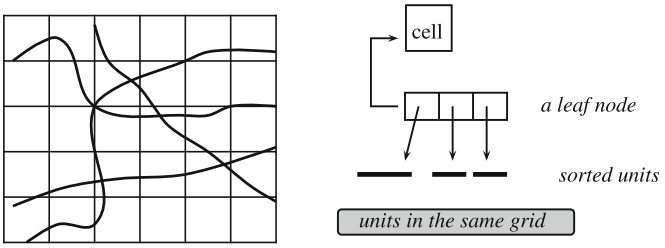
An outline of the solution is provided in Fig. 2. To efficiently find trajectories within a certain distance to the target, an index structure is essentially needed. We have implemented well established trajectory indexes including TB-tree [16], 3D R-tree and Grid index [3]. We perform a traversal on the indexes to retrieve trajectories that approximately belong to the result and then do the exact distance computation. The distance function over a time period is represented by a parabola function. We will receive pieces of distance functions and split trajectories at certain points to restrict the movement fulling the query condition. The method is able to find the tracking behavior but can be further optimized.



(a) The architecture (b) Distance curve

Fig. 2. An overview

To enhance the performance, we propose an index structure that combines the grid index and R-tree, as shown in Fig. 3. The 2D space is partitioned into a set of equal-size cells and trajectories are decomposed according to cells. That is, each trajectory is split into pieces and each piece is limited to a cell. We sort trajectory pieces by cell id, time and the spatial box. Each leaf R-tree node only contains trajectories from the same cell. The query procedure will make use of the cells to prune the search space. We first determine the cells in which the query trajectory is located. Since cells have the same size, we can quickly calculate *target* cells which are within the specified distance to the query trajectory. *Target* cells will be used to



(a) Grid (b) Grid R-tree nodes

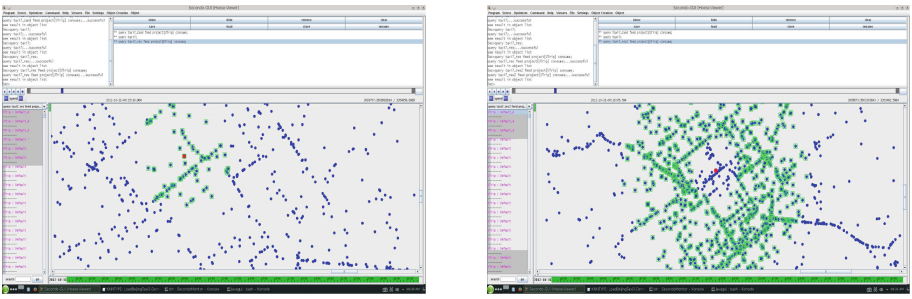
Fig. 3. Grid R-tree

prune R-tree nodes that do not cover *target* cells because only trajectories located in *target* cells will contribute to the result.

4 Demonstration

The implementation is developed in an extensible database system SECONDO [7] and program in C/C++. We use real datasets from a data company DataTang [1] including 1,675,667 GPS records of Beijing taxis. There are 22,269 trajectories in total.

In the demonstration, we randomly select a trajectory from the dataset and search trajectories keeping the specified distance to the query during its whole life time. Two distance parameters are defined: (i) $d = [0\,\text{m}, 5000\,\text{m}]$; and (ii) $d = [5000\,\text{m}, 20000\,\text{m}]$ and the results colored by green are displayed in Fig. 4. We are able to animate moving objects in the java interface such that one can observe how returned objects change over time. If an object always keeps the distance to the query, then it is the *tracker*.



(a) within 5km (b) between 5km and 20km

Fig. 4. Continuous distance queries (Color figure online)

Distance join queries. We also demonstrate distance join queries over trajectories, as illustrated in Fig. 5. The result is a set of pairs of trajectories such that each pair colored by green and red has the period of the intersection time more than a certain value (e.g., 15 min) and the distance always less than a threshold during the period. This would help finding common traveling routes among different travelers and discovering potential relationships such as neighbor and colleague.

Index performance. By scaling the distance parameter $d = \{[0, 5\,\text{km}], [0, 10\,\text{km}], [0, 20\,\text{km}]\}$, we compare the query performance of methods using different index structures, as reported in Fig. 6. One can see that 3D R-tree and TB-tree outperform the grid index up to an order of magnitude.

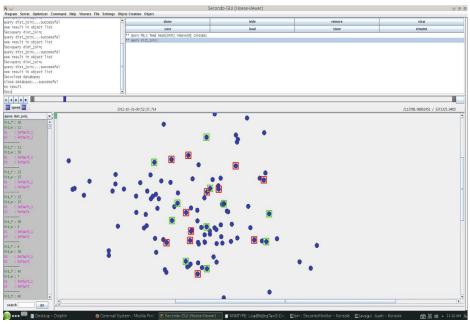


Fig. 5. Distance join queries (Color figure online)

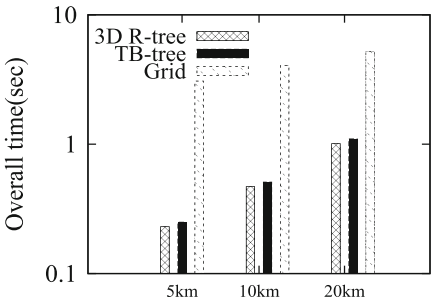


Fig. 6. Performance comparison

5 Conclusions

We study detecting the tracking behavior over trajectory data and formulate the problem by continuous distance queries. The framework of processing the queries is introduced including partitioning the data, building index structures, developing query algorithms and animating moving objects in the user interface. We demonstrate answering continuous distance and distance join queries in a prototype database system.

The future work is to evaluate the system performance by using big trajectory data and flexibly visualize a large number of trajectories in the user interface.

Acknowledgment. This work is supported by NSFC under grant numbers 61300052 and the Fundamental Research Funds for the Central Universities NO. NZ2013306.

References

1. http://factory.datatang.com/en/ (2006)
2. Arumugam, S., Jermaine, C.: Closest-point-of-approach join for moving object histories. In: ICDE, p. 86 (2006)
3. Chakka, V.P., Everspaugh, A., Patel, J.M.: Indexing large trajectory data sets with SETI. In: CIDR (2003)
4. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: SIGMOD, pp. 491–502 (2005)
5. Gao, Y., Zheng, B.: Continuous obstructed nearest neighbor queries in spatial databases. In: ACM SIGMOD, pp. 577–590 (2009)
6. Gao, Y., Zheng, B., Chen, G., Li, Q.: Algorithms for constrained k-nearest neighbor queries over moving object trajectories. GeoInformatica **14**(2), 241–276 (2010)
7. Güting, R.H., Behr, T., Düntgen, C.: SECONDO: a platform for moving objects database research and for publishing and integrating research implementations. IEEE Data Eng. Bull. **33**(2), 56–63 (2010)
8. Güting, R.H., Behr, T., Xu, J.: Efficient k-nearest neighbor search on moving object trajectories. VLDB J. **19**(5), 687–714 (2010)

9. Hung, C.C., Peng, W.C., Lee, W.C.: Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. VLDB J. **24**(2), 169–192 (2015)
10. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. PVLDB **1**(1), 1068–1080 (2008)
11. Kalnis, P., Mamoulis, N., Bakiras, S.: On discovering moving clusters in spatio-temporal data. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 364–381. Springer, Heidelberg (2005). doi:10.1007/11535331_21
12. Lange, R., Dürr, F., Rothermel, K.: Efficient real-time trajectory tracking. VLDB J. **20**(5), 671–694 (2011)
13. Li, Y., Li, J., Shu, L., Li, Q., Li, G., Yang, F.: Searching continuous nearest neighbors in road networks on the air. Inf. Syst. **42**, 177–194 (2014)
14. Li, Z., Ding, B., Han, J., Kays, R.: Swarm: mining relaxed temporal moving object clusters. PVLDB **3**(1), 723–734 (2010)
15. Pelekis, N., Tampakis, P., Vodas, M., Panagiotakis, C., Theodoridis, Y.: In-DBMS sampling-based sub-trajectory clustering. In: EDBT 2017, pp. 632–643 (2017)
16. Pfoser, D., Jensen, C.S.: Novel approaches in query processing for moving object trajectories. In: VLDB, pp. 395–406 (2000)
17. Su, H., Zheng, K., Wang, H., Huang, J., Zhou, X.: Calibrating trajectory data for similarity-based analysis. In: SIGMOD, pp. 833–844 (2013)
18. Zhou, P., Zhang, D., Salzberg, B., Cooperman, G., Kollios, G.: Close pair queries in moving object databases. In: ACM-GIS, pp. 2–11 (2005)