Contact Query Processing Based on Spatiotemporal Trajectory

Shuchang Zhang and Zhiming Ding(✉)

Beijing University of Technology, Beijing 100022, China
zmding@bjut.edu.cn

Abstract. Due to the prevalence of location-based devices, user trajectories are widely available in daily life, and when an infectious disease outbreak occurs, contact tracking can be achieved by examining the trajectories of confirmed patients to identify other trajectories of direct or indirect contact. In this paper, we propose a generalized trajectory contact search (TCS) query that models the contact tracking problem and other similar trajectory-based problems. In addition, we propose a new method for building spatio-temporal indexes and an algorithm for DBSCAN clustering based on spatio-temporal lattices to find all contact trajectories, which iteratively performs a distance-based contact search to find all contact trajectories. The algorithm, which is able to downscale the location and time of trajectories into a one-dimensional data and maintain the spatio-temporal proximity of the data, reduces the dimensionality of the search and improves the time and space efficiency. Extensive experiments on large-scale real-world data demonstrate the effectiveness of our proposed solution compared to the baseline algorithm.

Keywords: Spatiotemporal index · Trajectory Contact Query · Trajectory data

1 Introduction

Our daily movements are collected in trajectory data by various devices such as GPS, Bluetooth, cellular towers, etc. Among the different types of trajectory queries, there is a lack of research on the use of trajectory contact features, which occur when two objects are in close proximity to each other over a period of time, allowing information (infectious diseases, chemical/radiation spills, airborne materials, etc.) to be transmitted from one to the other. A direct application is the tracking of contacts during the New Coronary Pneumonia pandemic. A large body of literature highlights the importance of contact tracking, and applications for tracking contact events via Bluetooth connections between devices have been established in some countries. However, this approach is hardly practical because it requires strict user involvement and it cannot detect contact events with different radii due to hardware limitations. In contrast, contact

Supported by organization x.

tracking via traces is more applicable because the trace distance is measurable and the data is widely available.

Therefore, in this paper, we propose a new type of trajectory query, termed as Trajectory Contact Search (TCS), which finds all trajectories that directly and indirectly contact the query trajectory. Specifically, when two trajectories appear within a distance over a time period, we say they make a contact and one can influence the other. Then, given a query trajectory Tr_q, a distance ϵ, and the time step threshold k, we aim to find not only all trajectories R' it contacts, but also all the trajectories contacted by the influenced results R' subsequently. In fact, trajectory contact tracing is non-trivial. As pre-computing all contact events is not viable due to the flexibility of contact definition (ϵ and k), the query can only be answered by searching direct contacts to the influenced trajectory recursively. Besides, as a contact event requires both spatial and temporal continuity, new index and scanning algorithm are required to store and retrieve timestamp-level trajectories efficiently. Overall, our contributions are as follows:

- We propose a trajectory contact search query for the contact tracking problem.
- We propose a solution based on spatio-temporal grid partitioning to answer the contact search query without redundancy. In addition, we propose an algorithm based on DBSCAN clustering of spatio-temporal lattices to find all contact trajectories to further improve the temporal and spatial efficiency.
- Extensive experiments on large-scale real-world datasets show that our approach can answer TCS queries more efficiently than existing methods

2 Related Works

To the best of our knowledge, the travelling group discovery problem is closely related to TCS. Itfinds all groups of objects that move together over a period of time. Depending on how to define proximity (distance-based [1,2,4], or density-based [3,7,9]) and whether the time period is required to be consecutive [2,3,7] or not [5,9], various group patterns are identified. To discover the groups, the trajectories arefirst sliced into temporal snapshots, then a clustering algorithm or predefined criteria is applied to each snapshot tofind groups. Finally, the clusters from adjacent snapshots are intersected and concatenated until forming a long time sequence satisfying travel requirements. Besides, to enable distance comparison in every timestamp, linear interpolation is introduced to ensure an object to appear in every snapshot it crosses, which greatly inflates the input size and the processing cost. To reduce the cost, [3] uses trajectory segments instead of points, and it further simplifies trajectories using Douglas-Peucker algorithm. Meanwhile, [7] proposes a travelling buddy structure to capture the minimal groups of objects and perform intersection on buddies instead.

Geohash effectively defines an implicit, recursive quadtree over the worldwide longitude-latitude rectangle and divides this geographic rectangle into a hierarchical structure. The division continues along the longitude and latitude directions alternately until the desired resolution is achieved. During each division, if the target coordinate value is greater than the division point, a ‘1’ bit is appended to the overall set of bits; otherwise, a '0' bit will be appended. So each node of the recursive quadtree can represent a fixed spatial bounding box. Finally, GeoHash uses a 1D string to represent a 2D rectangle from a given quadtree node. The GeoHash string is derived by interleaving bits obtained from latitude and longitude pairs and then converting the bits to a string using a Base 32 character map. For example, the point with coordinates of $45.557, 18.675$ falls within the GeoHash bounding box of "u2j70vx29gfu". GeoHash has been widely implemented in many geographic information systems (e.g. PostGIS), and also used as a spatial indexing method in some NoSQL databases (e.g. MongoDB).

3 Problem Statement

Definition 1 (Trajectory). A trajectory is a series of chronologically ordered points $Tr_o = \langle p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n \rangle$ representing the historical trace of an object o. Each point $p_i = \langle x, y, t \rangle$ indicates the location of o at time $p_i.t$.

As for a contact event, two objects are defined as contacted if their trajectories (1) are close to each other at a certain point in time, and (2) such proximity is kept for a continuous period of time, formally defined as follows:

Definition 2 (Contact Event). Given a distance threshold ϵ and a duration k, objects a and b are directly contacted during $[t_u, t_v]$ if $\forall t_i \in [t_u, t_v], dist(a, b, t_i) \leq \epsilon$ and $t_v - t_u \geq k * \Delta t$, denoted as a contact event $C_{\epsilon,k}(a, b, [t_u, t_v])$.

Subsequently, we define the direct contact search problem below:

Definition 3 (Direct Contact Search (DCS)). Given a trajectory set R, a query trajectory Tr_q, a starting time t, a distance threshold ϵ and a duration k, a direct contact search $DCS(Tr_q, t, \epsilon, k)$ returns all trajectories T_o that satisfies: $\exists C_{\epsilon,k}(q, o, [t_u, t_v])$ where $t_u \geq t$ (direct contact).

Note that, if not specified, the query starting time t is assumed to set to the starting time of Tr_q. Now we are ready to define the trajectory contact search which further capture the indirect contacts:

Definition 4 (Trajectory Contact Search (TCS)). Given a trajectory set R, a query trajectory Tr_q, a distance threshold ϵ and a duration k, the trajectory contact search $TCS(Tr_q, \epsilon, k)$ returns all trajectories Tr_a which satisfy: there exists a sequence of trajectories $\langle Tr_0, Tr_1, \ldots, Tr_n \rangle$ where (1) $Tr_0 = Tr_q, Tr_n = Tr_a$, (2) $\forall i \in [1, n], Tr_i$ and Tr_{i-1} are contacted directly as $C_{\epsilon,k}(i-1, i, [c_i, c_i + k * \Delta t])$ and (3) $\forall i \in [2, n], ct_i \geq ct_{i-1} + k * \Delta t$.

4 Iteration-Based Trajectory Contact Search

A direct solution to address TCS follows the same routine of the disease transmission process. Starting from the query trajectory, it performs a DCS on a contacted trajectory in each iteration. Then trajectories retrieved by DCS are regarded as the newly contacted trajectories. The algorithm terminates when all contacted trajectories are examined. Intuitively, the iteration process may follow either Breath-First Search (BFS) or Depth-First Search (DFS) order. However, both can retrieve the result correctly but they may incur redundant computation, as one trajectory may contact multiple reported trajectories.We use a distance-based clustering algorithm to search for nearby trajectories to reduce the number of searches.

5 3DGeoHash-DB Contact Search Algorithm

5.1 3DGeoHash

GeoHash has been employed as an efficient indexing solution for massive 2D location data . But GeoHash only encodes the information of latitude and longitude. Our proposed 3DGeoHash method extends the idea of 3DGeoHash, and it includes the temporal dimension besides spatial dimensions. The core of the 3DGeoHash method is the encoding process that convert the items from the augmented 3D data structure into a sequence of characters, i.e. 1D string. Since the 3DGeoHash method defines a recursive octree on the temporally augmented world-wide geographic space, the maximum bounding box for this octree should be first established. The 3DGeoHash uses the following spatial and temporal extents:

Latitude and longitude: Since 3DGeoHash uses the WGS84 as the spatial references coordinate systems, the scope of longitude is $[-180°, 180°]$ while the scope of latitude is $[-90°,\ 90°]$

Time: Time is infinite and endless, while the 2D space has definite limits. Thus, a single year is divided in the 3DGeoHash encoding. There are two kinds of a single year, common year and leap year. The leap year contains 527040 min ($366 \times 24 \times 60$) while the common year has 525600 min ($365 \times 24 \times 60$). So the scope of a common year is $\left[\,0,\ 525600\,\right]$, and the scope of a leap year is $[0, 527040]$.

In Fig. 1, the longitude of one input point is $-40°$ and the division is carried out four times. As the total longitude scope is $[-180°, 180°]$, the longitude is split into two parts ($[-180°, 0°]$ and $[0°, 180°]$) in the first division. If the points belong to the left part $[-180°, 0°]$, then the points are marked as '0', or marked as '1'. Hence, $-40°$ belongs to $[-180°, 0°]$, so the first binary bit is '0'. In the second division, $[-180°, 0°]$ is split into $[-180°, -90°]$ and $[-90°, 0°]$. Because $-40°$ belongs to $[-90°, 0°]$, the second binary bit is '1'. The third and the forth divisions continue in the same way. Thus, the final binary code is "0110".

Since each child node has an extent equal to half of the interval of the parent node, a more direct calculation is formulated as the following. Given that the total scope in one dimension is $[X_{\min}, X_{\max}]$ and the total height of the final binary tree is h, the resolution of the leaf node will be r :

$$r = \frac{X_{\max} - X_{\min}}{2^h}$$

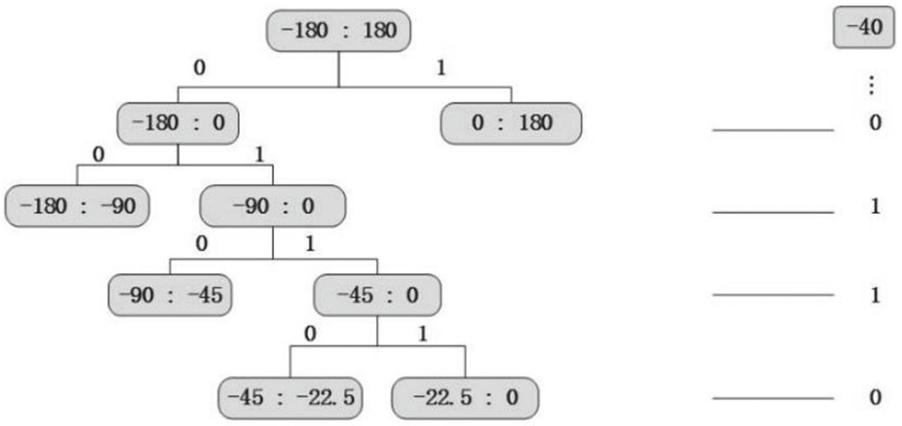


Fig. 1. The binary tree built by four divisions.

If the input value is X_i, the decimal code C_d can be derived as:

$$C_d = \left\lceil \frac{X_i - X_{\min}}{r} \right\rceil$$

In the above-mentioned example, if the longitude Xi is $-40°$, $X_{\max} = 180°, X_{\min} = -180°$, and $h = 4$. According to the Eq. 1, $r = 22.5°$ is obtained. Through the Eq. 2, the decimal code C_d is equal to 6. Finally, C_d can be transformed to the binary code $C_b = 0110$.

This encoding process is applied to all three dimensions, and yields three bit sequences, where the number of bits in each corresponds to the number of levels in the tree. For example, if the input data is $(-140°, 20°, 2015 - 6 - 1\ 00{:}00{:}00)$ and h is 10, then the input value of time '2015-6-1 00:00:00' will be transformed into a decimal value, 217440. The three derived bit sequences are:

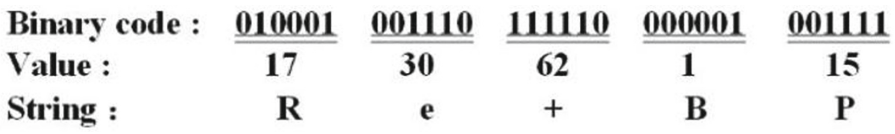
$$\begin{array}{ll} \textit{longitude}: & 0001110001 \\ \textit{latitude}: & 1001110001 \\ \textit{time}: & 0110100111 \end{array}$$

Finally, the three binary codes of the input trajectory point along the longitude, latitude, time dimensions are interleaved into one long binary code. The three bit sequences in this example are interleaved, and the complete binary code of the given trajectory point is listed as the following:

$$010001001110111110000001001111$$

B. The Base64 string of a trajectory point

The complete binary code of the given trajectory point is too long and cannot be directly stored in the target database. So the whole binary code is transformed to a Base64 string in a binary-to-text encoding schema to make it more convenient for database storage. During this encoding, six bits in the binary code are grouped into a corresponding character according to the Base64 map table. Thus the above-mentioned binary code in the Section 3.1 is transformed to a string "Re+BP", shown in Fig. 2. The detailed code-to-string transformation is illustrated as follows.



Binary code :	**010001**	**001110**	**111110**	**000001**	**001111**
Value :	**17**	**30**	**62**	**1**	**15**
String :	**R**	**e**	**+**	**B**	**P**

Fig. 2. The binary-code-to-string transformation process.

If the length of the transformed string is l and the height of binary tree is h, then the relationship between the l and h will be

$$h = 2l$$

The Eq. 3 indicates that each character in the 3DGeoHash string represents two levels in the octree. The resolution r of the octree leaf node in one dimension is

$$r == \frac{X_{\max} - X_{\min}}{2^{2l}}$$

If the spatiotemporal resolution r is known, the height h of the octree is obtained through the Eq. 5:

At last, C_d is transformed to a binary code C_b.

$$h = \left\lceil \frac{\ln\left(\left(X_{\max} - X_{\min}\right)/(r/2)\right)}{\ln(4)} \right\rceil$$

The string does not yet include the year information, so a prefix representing the year is appended to the string. Since the input data is $(-140°, 20°, 2015-6-100:00:00)$, the prefix is "2015-". Finally, the complete 3DGeoHash string is "2015Re+BP"

5.2 DBscanclustering Algorithm Based on Spatio-Temporal Grid

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a more representative density-based clustering algorithm. Unlike division and hierarchical clustering methods, it defines clusters as the maximum set of density-connected points, is able to divide regions with sufficient density into clusters, and can discover clusters of arbitrary shapes in a spatial database of noise.

We use a set of neighborhoods to describe the closeness of the sample set, and the parameters $(\epsilon, MinPts)$ are used to describe the closeness of the sample distribution in the neighborhood. Where,ϵ describes the neighborhood distance threshold of a sample, and MinPts describes the threshold of the number of samples in the neighborhood of a sample with distance ϵ.

After generating the corresponding HASH codes for each trajectory point according to the 3DGEOHASH algorithm, for each HASH code corresponding to a different trajectory point, according to the distance given by ϵ, set the threshold of the number of samples in the neighborhood as 2, and perform DBSCAN clustering, and then generate different population classifications, the specific generated effect is shown in Fig.

Within a divided 3DGEOHASH spatio-temporal grid, all trajectory points satisfy the definition condition of contact, i.e., between trajectories, the distances are less than a given threshold, and the time interval is within the definition time of contact. Since the scale of the number of trajectory points in the same spatio-temporal grid is smaller, the trajectories are more uniformly distributed at each spatio-temporal scale, so the time required to perform the clustering algorithm is shorter.

After the DBSCAN method groups the people in the same spatio-temporal grid, all people in the same group are marked as contact as long as they are in contact or indirect contact when performing contact query, so that only one lookup is done for each group in query. Therefore, this method is easy to query, there will be no missing queries, and the number of queries is only once, so there is no need for multiple recursive queries, saving time and space.

6 Experiments

6.1 Experiment Settings

The experimental setup lacks public trajectory datasets with sufficient scale and density, especially for pedestrian trajectories, so we conducted 500 efficient trajectory contact query processing experiments on a real commercial dataset of cab trajectories in Beijing, China. We also experimented on an existing user location check-in dataset, Gowalla, a location-based social networking site where users can share their location by checking in. This dataset contains 6,442,890 check-ins from these users between February 2009 and October 2010. To ensure the correctness of the evaluation results, we randomly select 10, 20, 30, 40, 50 trajectories from the dataset as query trajectories, and evaluate the total runtime as well as evaluate the average query efficiency i.e., the number of contacts searched for a single trajectory query.

Our algorithms are implemented in Java and all experiments are performed on a single server. Our chosen solutions include 3DR-Tree [10] and R-Tree indexes + temporal indexes and HR-Tree [11], and our proposed algorithm for 3DGeoHash-DB.

6.2 Effectiveness Study

Figures 3 and 4 depict the total running time of TCS search using different algorithms and indexes on the two datasets, respectively. From the figure we are

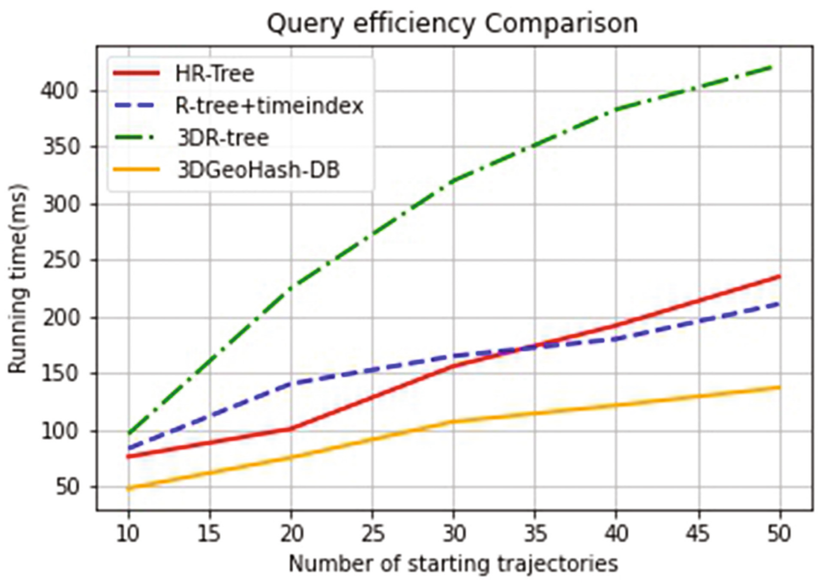


Fig. 3. Running time on T-Drive.

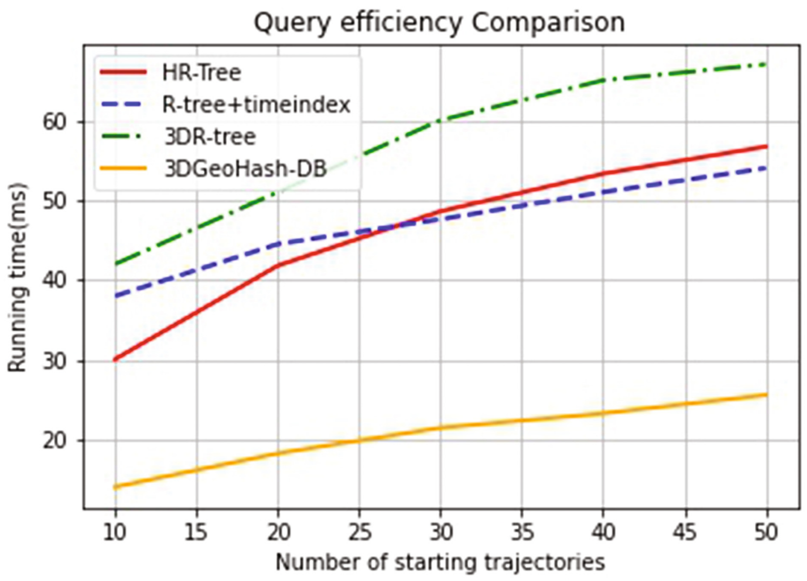


Fig. 4. Running time on Gowalla.

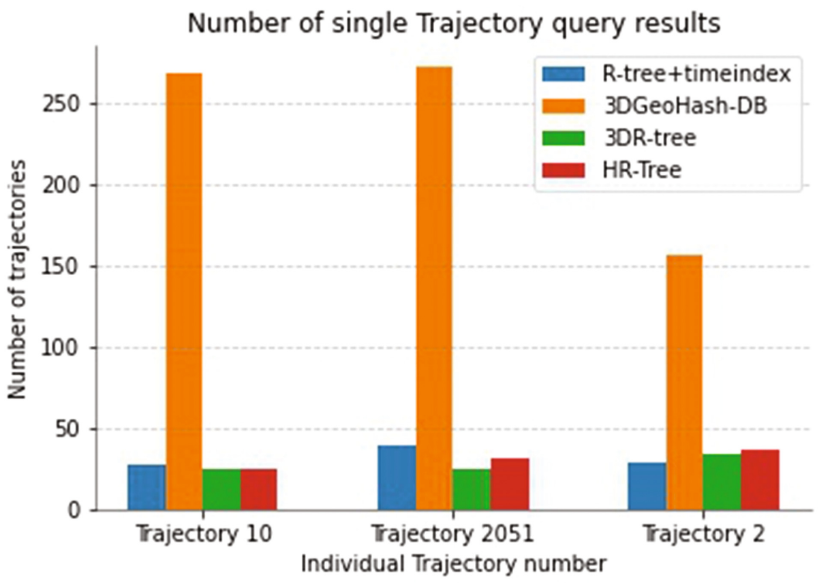


Fig. 5. Number of single track query results on T-Drive.

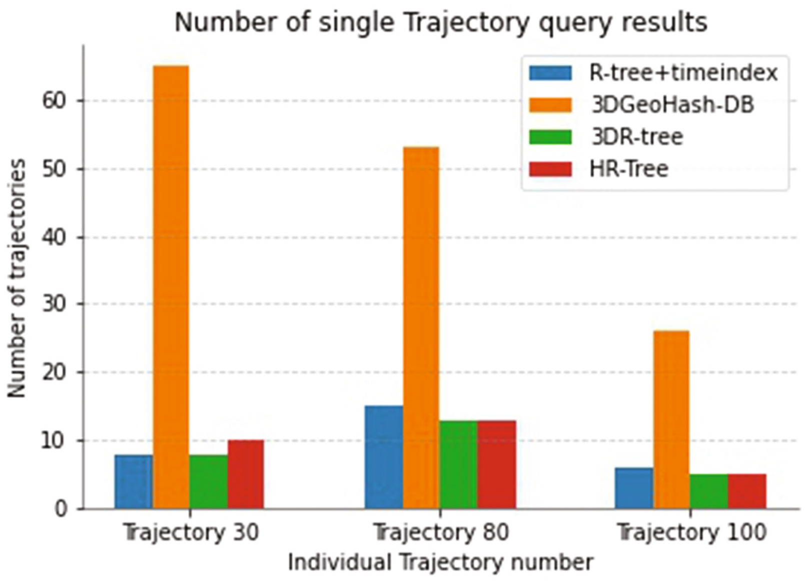


Fig. 6. Number of single track query results on Gowalla.

able to see that our proposed 3DGeoHash-DB algorithm consistently outperforms other traditional solutions, and the gap is larger as the number of query trajectories increases, which is in line with our expectation because more trajectories are queried to have more intersections in the contact result set, and our method only traverses the trajectory search results in the intersection once, preventing duplicate searches and saving time.

Figure 5 and 6 depicts the number of contacts that can be searched for a trajectory query using different algorithms and indexes. From the figure, we can see that our proposed 3DGeoHash-DB algorithm clearly outperforms the other two solutions, which is in line with our expectation, because the other methods search in a circle with a certain distance radius when querying, i.e., only direct contacts can be queried, and trajectories of indirect contacts cannot be queried. The single trajectory query will suffer from result omission. In contrast, our method uses distance-based clustering for trajectory grouping, and each query adds all trajectories within the same group to the results, which makes the average query efficiency improve.

7 Conclusion

In this paper, we introduce a new trajectory contact search query to model the trajectory contact problem. We propose a new spatio-temporal indexing method and a DBSCAN clustering algorithm based on a spatio-temporal grid to find all contact trajectories, which iteratively performs a distance-based contact search to find all contact trajectories. The algorithm, which can reduce the location and time of trajectories into a one-dimensional data and maintain the spatio-temporal proximity of the data, reduces the dimensionality of the search and is efficient in performing spatio-temporal proximity search. Experiments show that our scheme achieves faster query speed and less space consumption.

References

1. Buchin, K., Buchin, M., van Kreveld, M., Speckmann, B., Staals, F.: Trajectory grouping structure. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 219–230. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40104-6_19
2. Gudmundsson, J., van Kreveld, M., Speckmann, B.: Efficient detection of motion patterns in spatio-temporal data sets. In: Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems, pp. 250–257 (2004)
3. Jeung, H., Shen, H.T., Zhou, X.: Convoy queries in spatio-temporal databases. In: ICDE, pp. 1457–1459. IEEE (2008)
4. van Kreveld, M., Löffler, M., Staals, F., Wiratma, L.: A refined definition for groups of moving entities and its computation. Int. J. Comput. Geom. Appl. **28**(02), 181–196 (2018)
5. Li, Z., Ding, B., Han, J., Kays, R.: Swarm: mining relaxed temporal moving object clusters. PVLDB **3**(1–2), 723–734 (2010)

6. Schmidt, J.M.: Interval stabbing problems in small integer ranges. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 163–172. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_18
7. Tang, L.A., et al.: On discovery of traveling companions from streaming trajectories. In: ICDE, pp. 186–197. IEEE (2012)
8. Xu, J., Lu, H., Bao, Z.: IMO: a toolbox for simulating and querying "infected" moving objects. PVLDB **13**(12), 2825–2828 (2020)
9. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S., Zhou, X.: Online discovery of gathering patterns over trajectories. TKDE **26**(8), 1974–1988 (2013)
10. Pfoser, D., Jensen, C., Theodoridis, Y.: Novel approaches to the indexing of moving object trajectories. In: Proceedings of VLDB, pp. 395–406 (2000)
11. van der Spek, S., van Schaick, J., de Bois, P., de Haan, R.: Sensing human activity: GPS tracking. Sensors **9**(4), 3033–3055 (2009)