多様化軌跡を効率検索ための統合クエリパラダイム

胡　晟[†]　馬　強[†]　肖　川[††]

[†] 京都大学情報学研究科　〒 606-8501　京都市左京区吉田本町
[††] 大阪大学情報科学研究科　〒 565-0871 大阪市吹田区山田丘
E-mail: [†]hu.sheng.4s@kyoto-u.ac.jp, [††]qiang@i.kyoto-u.ac.jp, [†††]chuanx@ist.osaka-u.ac.jp

あらまし　In this work, we present a novel query paradigm by integrating point query and range query for searching road network trajectory data. Point query asks for trajectories satisfying the spatial relationship to a set of points, while range query searches trajectories passing a given spatial region R. To our best knowledge, this is the first work to address an integrated query combining point query and range query over trajectories. However, such a complex query paradigm might cause prohibitive query processing time overhead, and even worse performances on practical applications such as detour recommendation and top-k diversified trajectory search. Thus we treat the trajectories as strings and apply an index, Nested Trie, to address the efficiency issue. We equip the Nested Trie with spatial partition information to allow it to process range queries. Furthermore, we develop matrix factorization techniques disclosing the geographical transition patterns to efficiently search for top-k diversified trajectories on the index. Such techniques are helpful for applications such as trip planning and route explorations.

キーワード　Trajectory search, top-k diversified search

1 Introduction

The prevalence of GPS devices has generated massive trajectories in various domains. Particularly, the trajectories shared for entertainment purposes are commercially beneficial to the applications such as sightseeing and route planning. As many social networking sites such as Twitter, Facebook and Foursquare begin to support trajectory sharing service, how to retrieve and display valuable answer sets from large amount of trajectories has become an essential problem. According to [19], the motivation for trajectory search is to utilize the pleasant experience obtained from previous visitors. The historical trajectories might be spent a lot of time on collecting spot information from multiple sources by the previous visitors thus being well-planned. Such carefully planned trajectories might also include some splendid “anaba” spots that the map-based navigations system such as Google Map cannot provide. Moreover, the historical trajectories may have avoided some unpleasant roads (e.g., with poor public security) although these roads are in the shortest path.

Existing query types for trajectory search include point query, range query and trajectory query [30]. Point query asks for trajectories which satisfy the spatial relationship to a set of points. Range query asks trajectories passing a given spatial region R. Trajectory query asks for similar trajectories according to some pre-defined distance-based similarity metrics. However, existing systems can only support processing a single type of query at once while we want to handle a trajectory search query consisting of points, ranges and (partial) trajectories in an integrated way. There are a lot of benefits from the proposed integrated query mechanism. First, point query can describe the exact locations that the user want to visit. Second, range query can cover places those are not single point locations but large areas containing several spatial objects. Also, for users unfamiliar with a new city, such range query provide an exploratory method to search when they cannot specify exact point locations. Third, partial trajectory query can describe the exact route (e.g., shopping arcade, seaside) the user want to pass through. In our settings, a point query selects a/multiple point(s) of interest (POI), a range query selects a/multiple region(s) in rectangular shapes and a trajectory query selects a/multiple path(s) beginning with a POI(s).

［Example 1］ In Fig. 1, a user wants to search for trajectories passing p_1, p_2, p_3 and also passing the rectangular region R_1. This user selects three points p_1, p_2, p_3 as a point query and one region R_1 as the range query. The point query and range query are issued in an integrated way. The system processes the integrated query and returns T_1, T_2 and T_3 that satisfy all the query constraints.

Search efficiency problem emerges in case of a large amount of trajectories being given. Existing works [2,5,17,21,25] for trajectory search rely too much on R-tree [3], which stores all points from the raw trajectories thus consuming huge space and incurring ineffective pruning performance. Learning a lesson from R-tree, we have to consider the other data struc-

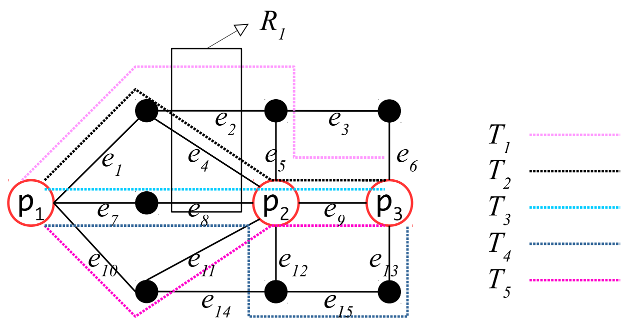

Figure 1 An Example of Map-matched Trajectories

tures for our integrated query processing. Inspired by the map-matching techniques that convert the raw trajectory into a sequence of road edges, we apply string indexing techniques [7] on the map-matched trajectories to address the above efficiency and space issues. Although several recent studies [9, 12, 22] have shown that such lightweight representations can improve query processing speed and easier for compression, none of them considers processing an integrated query rather than a single-type query.

Moreover, top-k trajectory diversification also has been a problem for search applications because displaying a large answer set containing similar trajectories just messes up the interface and confuses the users. Although a lot of efforts [4,5,14,28] have been conducted on this topic, our work is the first one to take use the map-matched trajectories and apply matrix factorization techniques on the top-k trajectory diversification problem. Matrix factorization can show transition patterns between different spatial areas and is used to rank the retrieved trajectories according to how likely the user will pass through along them. We take use of public available POI visit datasets from Foursquare and Flickr as training sets. Such novel techniques might open another direction to exploit such POI visit datasets on search applications.

To our best knowledge, our work is the first one to focus on an integrated query paradigm processing trajectory search on map-matched trajectories. We also introduce a novel way utilized matrix factorization to return top-k diversified answers.

Our main contributions are summarized as follows.

- We propose an integrated query paradigm for trajectory search (Section 2).
- We apply a string index for the map-matched trajectory search (Section 3).
- We develop matrix factorization techniques for top-k trajectory diversification tasks (Section 4).

2 PRELIMINARIES

In this section, we introduce the basic data model and query model used in this paper. Then, we provide an overview of the integrated query processing system.

2.1 Data Model

[Definition 1] (GPS points) A GPS point x is a triplet consists of $\langle x.lat, x.lng, x.time \rangle$, which represents the latitude coordinates, longitude coordinates and time stamp respectively.

[Definition 2] (Raw GPS Trajectory) A GPS trajectory is a list of time-ordered GPS points in the form of $\{x_1, x_2, \cdots, x_n\}$, where each x_i is a GPS point.

[Definition 3] (Road Network) A road network is a graph $G = (V, E)$, where V represents the set of vertices of intersections and E represents the set of road segments. For example, in Fig. 1, V includes all the intersections showed as black dots and p_1, p_2, p_3. E includes all the road segments as $\{ e_1, e_2, \cdots, e_{15} \}$.

[Definition 4] (POI) A POI is a special intersection on the road network. It usually represents a named location such as a sightseeing spot, a shopping mall, a restaurant, etc. For example, in Fig. 1, there are three POIs p_1, p_2, p_3 on the road network.

[Definition 5] (Map-matched Trajectory) A raw GPS trajectory can be converted as a sequence of road segments after being mapped on to their corresponding road network. Fig. 1 shows an example that T_1 can be converted into $e_1 \rightarrow e_2 \rightarrow e_3$.

[Definition 6] (Map-matched Trajectory with POIs) A raw GPS trajectory can be converted as a sequence consisting of both road segments and POIs according to the road network. By simply inserting the POIs appearing on the road network, such trajectories becomes Map-matched Trajectory with POIs. Fig. 1 shows an example that T_1 can be converted into $p_1 \rightarrow e_1 \rightarrow e_2 \rightarrow e_5 \rightarrow p_2 \rightarrow e_9 \rightarrow p_3$.

2.2 Query Model

[Definition 7] (Point Query) Given a list of points representing POIs must be traversed $P = p_i, p_j, \cdots, p_k$ and a map-matched trajectory dataset $TR = T_1, T_2, \cdots T_n$, we want to find all sub-trajectories of T_i, where T_i passed all the points in P with the given order.

[Definition 8] (Range Query) Given a list of rectangular regions representing regions must be traversed $R = r_i, r_j, \cdots, r_k$ and a map-matched trajectory dataset $TR = T_1, T_2, \cdots T_n$, we want to find all sub-trajectories of T_i, where T_i passed all the regions in R with the given order.

[Definition 9] (Trajectory Query) Given a list of edges

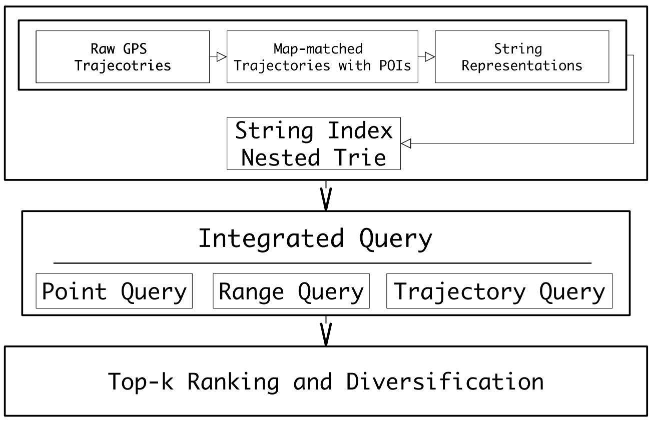

Figure 2 Overview of the Process

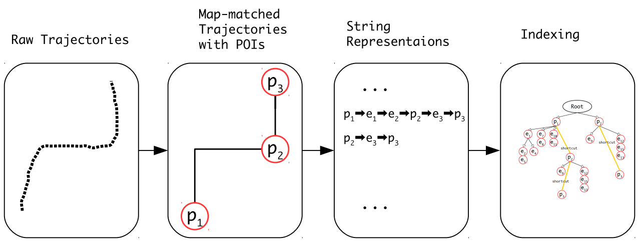

Figure 3 Processing Trajectories into Strings

starting with a POI must be traversed $J = p_0, e_i, e_j, \cdots, e_k$ and a map-matched trajectory dataset $TR = T_1, T_2, \cdots T_n$, we want to find all sub-trajectories of T_i, where T_i passed all the road segments in J with the given order.

[Definition 10] (Integrated Query) Given a list of points, rectangular regions and road segments must be traversed $Q = \{P, R, J\}$ and a map-matched trajectory dataset $TR = T_1, T_2, \cdots T_n$, we want to find all sub-trajectories of T_i, where T_i satisfied all the query constraints in Q.

2.3 System Overview

Fig. 2 shows an overview of the whole system.

- We first process the Raw GPS trajectories with map-matching techniques. We record the generated sequences as strings for post processing usage. Then we build a string index, Nested Trie [7], according to the string records, and attach spatial partition information on it.
- The constructed index will receive the integrated query consists of point query, range query and trajectory query. Efficient query processing algorithms will work out and generate a candidate answer set.
- The candidate answer set will be ranked with efficient pruning algorithms and a well top-k diversified set will be returned eventually.

3 Indexing

In this section, we show how we apply the string indexing techniques on trajectories search problems. Before that, we show the detailed processing steps that map-match a raw trajectory into a sequence consisting of edges and POIs in Section 3.1. A flowchart of the steps is shown in Fig. 3.

3.1 Processing Trajectories

We have three steps for processing the trajectories.

(1) **Step 1: Map-Matching.** In this step, our system collects all the raw trajectories and convert them into sequences of edges according to the corresponding road network. Each edge is represented by a unique integer ID. The map-matching approach [16] is adopted in this step.

(2) **Step 2: Inserting POIs.** The POI information is extracted from OpenStreetMap. Each POI is represented by a unique integer ID that is not overlapped with edge IDs. If the POI locations appear on the edge according to the road network, we insert such POIs into the edge sequence according to the visited order in the raw trajectories.

(3) **Step 3: Representing as Strings.** We record each sequence consisting of edge IDs and POI IDs as strings for the index construction latter. We also generate all the suffix strings from the gap of POI position in order to efficient search from the midterm POI position appearing in the trajectories. E.g., we record $p_2 \to e_3 \to p_3$ which is a suffix of the original sequence $p_1 \to e_1 \to e_2 \to p_2 \to e_3 \to p_3$ in Fig.3.

3.2 Nested Trie Construction

After map-matching trajectories with POIs, we collect strings that consist of two different types, edges and POIs. Inspired by the string indexing techniques Nested Trie proposed in our previous work [7] that also can process subsequent queries on two types of string characters, we apply such techniques to the integrated query processing of trajectory search.

Recall that Nested Trie consists of outer trie that indexes initial characters and inner trie that indexes non-initial characters. Hence we apply outer trie to index POIs and inner trie to index edges instead. The benefit is that Nested Trie provides efficient search performance for the queries that only traverse the outer trie. This corresponds the point query part in the trajectory search. If we want to additionally do a range query search, we only need to traverse a small part

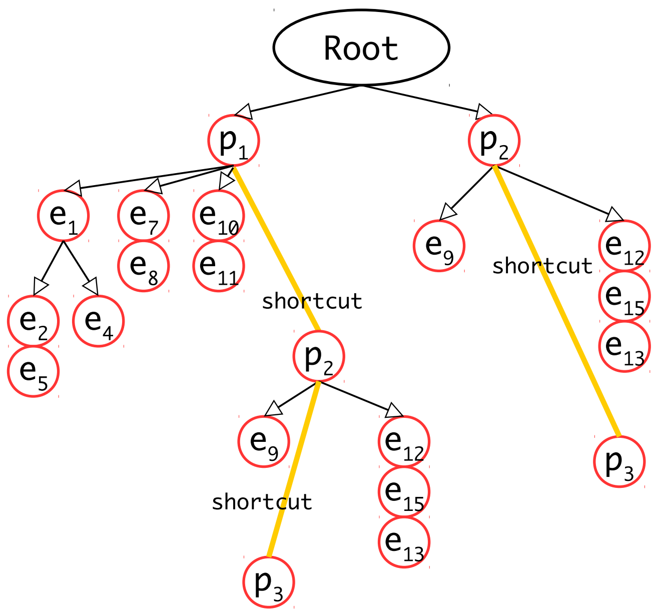

Figure 4 Build Nested Trie for Trajectories

of inner trie after the outer trie is traversed.

The construction of the Nested Trie is to only insert all the strings generated in Section 3.1. More details can be found in [7].

Table 1 Example dataset TR.

TrajID	Traj String
T_1	$p_1 \to e_1 \to e_2 \to e_5 \to p_2 \to e_9 \to p_3$
T_2	$p_1 \to e_1 \to e_4 \to p_2 \to e_9 \to p_3$
T_3	$p_1 \to e_7 \to e_8 \to p_2 \to e_9 \to p_3$
T_4	$p_1 \to e_7 \to e_8 \to p_2 \to e_{12} \to e_{15} \to e_{13} \to p_3$
T_5	$p_1 \to e_{10} \to e_{11} \to p_2 \to e_9 \to p_3$

An example trajectory set in Fig. 1 is given in Table 1 and the corresponding nested trie is shown in Fig. 4. The yellow lines represents the shortcuts connecting the outer trie.

3.3 Attach Spatial Partition Information

To support efficient search on range queries, we borrow the idea from the work [6] which attach spatial partition information on the node of Nested Trie. The experimental results in [6] show that a quadtree partition with an associated bit array on the trie node has effective pruning power on spatial range queries. In this work, we also adopt quadtree partition also it is also possible to replace it with R-tree or grid partitions.

3.4 Searching Algorithm

Below we show the searching algorithm for processing an integrated query that consists of point query and range query in Algorithm 1. As the trajectory query processing is similar with point query, we do not show it for saving spaces.

Algorithm 1: ProcessIntegratedQuery (Q, NT)

```
Input  : $Q$ is the integrated query consisting of points
         $Q.points$ and regions $Q.rng$, $NT$ is a nested trie
         built on $TR$.
Output : $\{T_i\}$, such that $T_i \in TR$ and satisfy all the
         constraints in $Q$.
1  $A \leftarrow \{$ the root of $TR\}$ ;                       /* node set */
2  $b \leftarrow$ InitRegionStatus($Q.rng$) ;                   /* bit array */
3  foreach $point_i$ in $Q.points$ do
4      $A' \leftarrow \emptyset$;
5      foreach $n \in A$ do
6          if $n$ is an outer node AND $n$ has a child n' through
            outer edge $point_i$ then
7              $A' \leftarrow A' \cup \{n'\}$ ;                 /* match $p_i$ */
8              continue;
9          if $n$ has any descendant $n'$ through outer edge $point_i$
            within a distance threshold $\tau$ then
10             $A' \leftarrow A' \cup \{n'\}$ ;         /* match distant $p_i$ */
11     $A \leftarrow A'$;
12 $R \leftarrow \emptyset$ ;                            /* returned node set */
13 foreach $n \in A$ do
14     traverse each subtree rooted at $n$;
15     foreach traversed node $n'$ do
16         $c \leftarrow$ ($b$ AND the region bit arrays on $n'$);
17         if $c = 0$ then
18             stop traversing the subtree rooted at $n'$ ;
                /* pruning */
19         if $n'$ is the leaf node in current inner tree then
20             only keep those intervals $c \neq 0$ on $n'$;
21             $R \leftarrow R \cup \{n'\}$ ;                    /* remaining */
22 foreach $n \in R$ do
23     return those trajectories $T_i$ appearing in $n.intervals$;
```

4 Top-k Diversification Ranking

4.1 Top-k Ranking

The main idea of the top-k ranking is that, the points selected by the users should represent some degrees of the user's preference. Although existing POI recommendation works [1,13,15,31] focus on predicting the next POI that user will visit, none of these works considers applying the data mining techniques on top-k trajectory search tasks. As mentioned in Section 1, results from trajectory search have many advantages over predicted POI sequences. The retrieved trajectories need to be carefully ranked by mining the associations between the user's preferences and POI correlations. We take use the point query issued by the users to provide a most preferred top-k results by training the ranker with enormous POI visit data (represent as $p_i \to p_j \to \cdots \to p_k$)

provided by Foursquare and Flickr. Moreover, we develop effective pruning algorithm to drop those trajectories with low ranks at an early stage. Such techniques and our string index works in concert with each other for efficient and effective recommendations.

As we have already separated the whole Euclidean space into quadtree partitions (see Section 3. 3), we can create the transition matrix $S \in \mathbb{R}^{|C|\times|C|}$, according to the POI visit dataset to describe the transition probability between different quadtree cells, where $|C|$ is the total leaf node numbers of quadtree. Because each POI visit record can be taken as a sequence:$c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_n$, where c_i represents a quadtree cell and $c_i \rightarrow c_j$ represents a transition between two cells. As each entry in S means a transition between two cells, after aggregating all the POI visits, we can obtain an initial transition matrix S'. S is obtained by normalizing S'. Thus we borrow the ideas from [13, 31] to apply matrix factorization techniques to disclose the latent relationships between different grid cells as below:

$$S \approx C_s M C_s^T \tag{1}$$

where $C_s \in \mathbb{R}^{|C|\times k_s}$, $M \in \mathbb{R}^{k_s\times k_s}$ represent the embedding matrix and interaction matrix, respectively. k_s means the number of latent features. More optimization details can be found in [31]. Then we obtain the transition probabilities by computing the transition matrix. Because we are not going to compute the probability of a trajectory, we take the probability as a travel gain here and aggregate the gain together for ranking purpose which is also used in [1]. When we search for a query on the index and obtain some trajectories in Algorithm 1, we use the bit array of a node to compute the a super set of trajectories to be retrieved and compute a upper bound UB for such a node. By comparing the UB gain with the k-th gain in the top-k results, we can easily prune such a node as long as the UB gain is less the the latter. The details are shown in Algorithm 2. We only need replace Line 22–23 in Algorithm 1 with Algorithm 2.

4. 2 Diversificaiton

To avoid displaying a large answer set containing similar trajectories messing up the interface, we also develop algorithms to diversify the results.

According to the property of Nested Trie, one interval will represent one same previous route. Then we can simply pick up one trajectory from each interval and compute a set with the highest diversity and also top-k gain. That means, when we successfully insert a T_i in the top-k set, we will skip to next interval on n. Such a detail is shown by Line 8 in Algorithm 2.

Algorithm 2: TopK-Diversify (R, S, k)

```
1 $Queue \leftarrow \emptyset$ ;          /* a priority queue of size $k$ */
2 foreach $n \in R$ do
3     if $n$.UB $\leqq Queue[k].score$ then
4         continue;
5     foreach $T_i$ appearing in $n.intervals$ do
6         if $|Queue| < k$ or $gain(S, T_i) > Queue[k].gain$
           then
7             $Queue.insert(T_i)$;
8             skip to next interval on n;
9 return $Queue$;
```

5 Experiments

As the experiments are being carried out hence we will provide results in the future. We describe the data preparation as below.

5. 1 Data Preparation

One dataset is used for building the Nested Trie.

- **Kyoto** is a trajectory dataset recording the raw trajectories generated from the tourists for sightseeing purposes. It contains 3,881 trajectories in total. The average number of road segments is 424 for each trajectory after map-matching.

Two datasets are collected for model training and test tasks.

- **Foursquare**[1] is a large-scale POI sequence visits dataset.
- **Flickr** is a dataset used widely for recording the check-in seqence POIs by Flickr users.

The experiments were carried out on a PC with an Intel i5 2.6GHz Processor and 32GB RAM, running Ubuntu 14.04.3. The algorithms were implemented in C++ and in a main memory fashion.

6 Related Work

Multi-Point Query on Trajectories. Searching trajectories by selecting several points as query was a very popular query paradigm and had a large body of existing works. We refer readers to a survey for a clear taxonomy over different query paradigms over trajectories [30]. In one early study, Chen *et al.* [2] first studied a problem of searching trajectories by selecting location points as a query, and call this problem k-Best Connected Trajectories (k-BCT) which best

1 : https://sites.google.com/site/yangdingqi/home/foursquare-dataset

connect the designated locations geographically with spatial distance and order constraint considered. After that, Tang *et al.* [21] proposed a new problem called k Nearest Neighboring Trajectories (k-NNT) with the minimum aggregated distance to a set of query points which has a similar target applications with [2]. Qi *et al.* [17] improved the efficiency performance of such point query by combining existing approaches to a hybrid method and also studied the practical variant of bounded distance search by considering the temporal characteristics of the searched trajectories. Yadamjav *et al.* [27] revisited the previous state-of-the-art search algorithms and proposed an optimized search method leading to further performance improvement. Recently, Shang *et al.* [19] extended the query points to query regions and proposed a novel query type called trajectory search by regions of interest (TSR query) which returns the trajectories with the highest spatial-density correlation to the query regions.

Multi-Range Query on Trajectories. The problem of retrieving trajectories passing multiple spatial regions has not been investigated too much. Most existing works [5, 25] focus on searching trajectories passing a single region and relied on traversing R-tree [3] for returning results. The baseline method [29] is to first find the answers of singe range query and then compute the intersection between these answers. However, many irrelevant trajectories will be retrieved and the intersection operations are usually computationally expensive. Yadamjav *et al.* [26] addressed this problem by proposing a two-level index structure that preserves both the co-location of trajectories and the co-location of points within trajectories for faster query processing.

Path Query on Trajectories. Path query processing relies on map-matching trajectories according to a road network. The map-matched trajectories can be seemed as sequences of edges or strings and are also called as Network Constrained Trajectories (NCTs). Sandu *et al.* [18] first studied the path query on NCTs by applying spatial filtering techniques and B^+-tree indexes for temporal filtering. Krogh *et al.* [11] extended the definition of path query and proposed strict path query where trajectories must follow all edges in the path. They presented a novel index NETTRA, which uses new path encoding scheme thus enabling efficient query processing. Song *et al.* [20] proposed a new framework PRESS, to effectively compress NCTs by representing the spatial and temporal information of trajectories separately. Krogh *et al.* [10] proposed SPNET, which is considered the first in-memory index for NCTs. Koide *et al.* [8, 9] addressed the strict path query by employing an inverse suffix array integrating the FM-index and B^+-tree with temporal information. Li *et al.* [12] handled the strict path query by modifying existing suffix tree structure to index trajectories using Microsoft Azure Table. Wang *et al.* [22, 23] proposed an integrated search engine which can answer path query, strict path query, range query and trajectory similarity query by indexing the edges and vertices of NCTs. Path query also has applications in the domain of data mining which estimates the time of a path using NCTs [24].

Trajectory Diversification. Showing the retrieved trajectories in a diversified way has aroused wide concern in recent years. Hsu *et al.* [5] applied a skyline computation to provide more diversity in their work. Liu *et al.* [14] formalized the K Shortest Paths with Diversity (KSPD) problem as finding paths that dissimilar with each other and the total length of the paths is minimized. Zhang *et al.* [28] generated routes according to visual-based diversity and facility-based diversity with information extracted from Google Street View images and FourSquare venues. He *et al.* [4] proposed Top-k Diversified Search (TkDS), which finds a set of k Origin-Destination pairs that the trajectories traversing in-between have the highest diversity.

Acknowledgment

This work is partly supported by MIC SCOPE(172307001).

Reference

[1] D. Chen, C. S. Ong, and L. Xie. Learning points and routes to recommend trajectories. In *CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 2227–2232, 2016.

[2] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: an efficiency study. In *ACM SIGMOD 2010*, pages 255–266. ACM, 2010.

[3] A. Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM SIGMOD Record, 1984.

[4] D. He, B. Ruan, B. Zheng, and X. Zhou. Origin-destination trajectory diversity analysis: efficient top-k diversified search. In *MDM 2018*, pages 135–144. IEEE, 2018.

[5] W. T. Hsu, Y. T. Wen, L. Y. Wei, and W. C. Peng. Skyline travel routes: Exploring skyline for trip planning. In *MDM 2014*, volume 2, pages 31–36. IEEE, 2014.

[6] S. Hu, C. Xiao, and Y. Ishikawa. An efficient algorithm for location-aware query autocompletion. *IEICE Transactions*, 101-D(1):181–192, 2018.

[7] S. Hu, C. Xiao, J. Qin, Y. Ishikawa, and Q. Ma. Autocompletion for prefix-abbreviated input. In *ACM SIGMOD 2019*, pages 211–228, 2019.

[8] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa. Cinct: Compression and retrieval for massive vehicular trajectories via relative movement labeling. In *ICDE 2018*, pages 1097–1108. IEEE, 2018.

[9] S. Koide, Y. Tadokoro, T. Yoshimura, C. Xiao, and Y. Ishikawa. Enhanced indexing and querying of trajectories in road networks via string algorithms. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 4(1):3, 2018.

[10] B. Krogh, C. S. Jensen, and K. Torp. Efficient in-memory indexing of network-constrained trajectories. In *ACM SIGSPATIAL 2016*, page 17. ACM, 2016.

[11] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp. Path-based queries on trajectory data. In *ACM SIGSPATIAL 2014*, pages 341–350. ACM, 2014.

[12] R. Li, S. Ruan, J. Bao, Y. Li, Y. Wu, L. Hong, and Y. Zheng. Efficient path query processing over massive trajectories on

the cloud. *IEEE Transactions on Big Data*, 2018.

[13] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui. Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 831–840, 2014.

[14] H. Liu, C. Jin, B. Yang, and A. Zhou. Finding top-k shortest paths with diversity. *IEEE TKDE 2017*, 30(3):488–502, 2017.

[15] Y. Liu, T. Pham, G. Cong, and Q. Yuan. An experimental evaluation of point-of-interest recommendation in location-based social networks. *PVLDB*, 10(10):1010–1021, 2017.

[16] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *ACM SIGSPATIAL 2009*, GIS ' 09, page 336343, New York, NY, USA, 2009. Association for Computing Machinery.

[17] S. Qi, P. Bouros, D. Sacharidis, and N. Mamoulis. Efficient point-based trajectory search. In *SSTD 2015*, pages 179–196. Springer, 2015.

[18] I. Sandu Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial. Indexing in-network trajectory flows. *The VLDBJ 2011*, 20(5):643–669, 2011.

[19] S. Shang, L. Chen, C. S. Jensen, J.-R. Wen, and P. Kalnis. Searching trajectories by regions of interest. *IEEE TKDE 2017*, 29(7):1549–1562, 2017.

[20] R. Song, W. Sun, B. Zheng, and Y. Zheng. Press: A novel framework of trajectory compression in road networks. *VLDB 2014*, 7(9):661–672, 2014.

[21] L.-A. Tang, Y. Zheng, X. Xie, J. Yuan, X. Yu, and J. Han. Retrieving k-nearest neighboring trajectories by a set of point locations. In *SSTD 2011*, pages 223–241. Springer, 2011.

[22] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu, and X. Qin. Torch: A search engine for trajectory data. In *ACM SIGIR 2018*, pages 535–544. ACM, 2018.

[23] S. Wang, M. Li, Y. Zhang, Z. Bao, D. A. Tedjopurnomo, and X. Qin. Trip planning by an integrated search paradigm. In *ACM SIGMOD 2018*, pages 1673–1676. ACM, 2018.

[24] Y. Wang, Y. Zheng, and Y. Xue. Travel time estimation of a path using sparse trajectories. In *ACM SIGKDD 2014*, pages 25–34. ACM, 2014.

[25] L.-Y. Wei, W.-C. Peng, and W.-C. Lee. Exploring pattern-aware travel routes for trajectory search. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(3):48, 2013.

[26] M.-E. Yadamjav, F. M. Choudhury, Z. Bao, and H. Samet. Efficient multi-range query processing on trajectories. In *International Conference on Conceptual Modeling*, pages 269–285. Springer, 2018.

[27] M.-E. Yadamjav, S. Wang, Z. Bao, and B. Zhang. Boosting point-based trajectory search with quad-tree. In *ADC 2017*, pages 29–41. Springer, 2017.

[28] Y. Zhang, P. Siriaraya, Y. Wang, S. Wakamiya, Y. Kawai, and A. Jatowt. Walking down a different path: route recommendation based on visual and facility based diversity. In *WWW 2018*, pages 171–174, 2018.

[29] R. Zheng, Q. Liu, H. Mao, H. Zhu, and W. Rao. Trajbase: searching trajectories in multi-region. In *Proceedings of the 2nd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, page 3. ACM, 2016.

[30] Y. Zheng and X. Zhou. *Computing with spatial trajectories*. Springer Science & Business Media, 2011.

[31] C. Zhuang, N. J. Yuan, R. Song, X. Xie, and Q. Ma. Understanding people lifestyles: Construction of urban movement knowledge graph from GPS trajectory. In *IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3616–3623, 2017.