Computer Science and Information Systems 19(3):1283–1304 https://doi.org/10.2298/CSIS210930031C

Nearest Close Friend Query in Road-Social Networks

Zijun Chen*, Ruoyu Jiang, and Wenyuan Liu

[1] School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
[2] The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China
zjchen@ysu.edu.cn
jiangruoyu@stumail.ysu.edu.cn
wyliu@ysu.edu.cn

Abstract. Nearest close friend query ($k\ell$NCF) in geo-social networks, aims to find the k nearest user objects from among the ℓ-hop friends of the query user. Existing efforts on $k\ell$-NCF find the user objects in the Euclidean space. In this paper, we study the problem of nearest close friend query in road-social networks. We propose two methods. One is based on Dijkstra algorithm, and the other is based on IS-Label. For the Dijkstra-based method, Dijkstra algorithm is used to traverse the user objects needed. For the label-based method, we make use of IS-Label to calculate the distance between two vertices to avoid traversing the edges that do not contain the desired user object. For each method, we propose effective termination conditon to terminate the query process early. Finally, we conduct a variety of experiments on real and synthetic datasets to verify the efficiency of the proposed methods.

Keywords: road-social networks, R-tree, IS-Label index, nearest neighbor query.

1. Introduction

With the development of location-aware smart devices, location-based applications have received extensive attention. Smart devices can allow users to obtain their own location information in location-based social networks, such as Foursquare, Facebook, Twitter, and Weibo.

The k-Nearest ℓ-Close Friends ($k\ell$-NCF) query [22] retrieves the k nearest data objects to a query point p_q from among the ℓ-hop friends of a query user u. The $k\ell$-NCF query is proposed in the Euclidean space. In real life, from the location of these returned data objects to the query point is limited by the road network. So, in this paper, we would propose the k-nearest neighbor ℓ-close friend query in road-social networks, which can also be applied in scenarios proposed in [22], such as making new friends, spatial crowdsourcing, blind dates, ridesharing, etc. We give one of the application scenarios in the following example.

Example 1. Spatial crowdsourcing. Spatial crowdsourcing is a platform, in which human workers can be assigned tasks related to a location. A requester q may issue a request to collect pictures in a specific location p_q. The worker who is assigned the task should be

* Corresponding author

close to p_q. To win the requester's trust, the worker should have acceptable social links to the requester. The 1-hop friends of the requester may be far away from p_q, or they could not accept the task. In this case, the requester may need to search the ℓ-hop friends of q.

In the road network shown in Fig. 1(a), there are four human workers B, C, D and E, who could accept the task. According to the social network shown in Fig. 1(b), q has two 1-hop friends B and F, and has 2-hop friends B, F, C and D. If only search 1-hop friends of q, F is the nearest neighbor of p_q. But F could not accept the task for some reason. B is far away from p_q. If search 2-hop friends of q, D is the nearest neighbor of p_q. In the Euclidean space, C is the nearest neighbor of p_q. But in the road network, C is farther away from p_q than D.

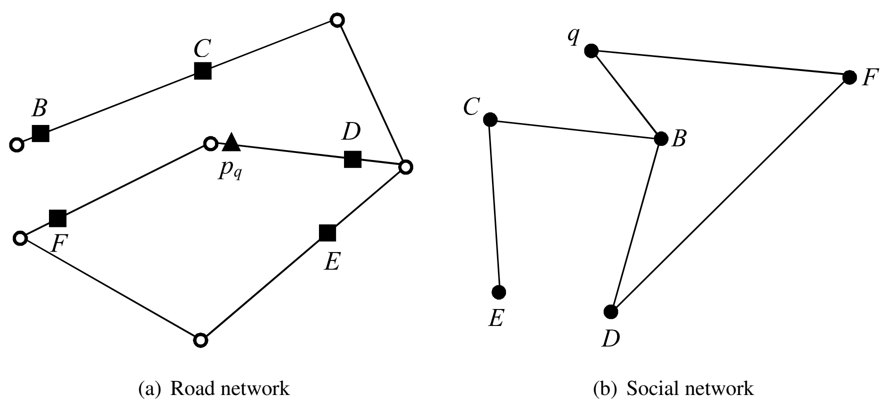


Fig. 1. Example of application scenario

Since the computation cost of the road network distance is much higher than that in Euclidean space, the methods proposed in [22] cannot be used to solve the problem of $k\ell$-NCF in road-social networks directly.

For the $k\ell$-NCF query in road-social networks, we will propose two methods. One is based on Dijkstra algorithm, and the other is based on IS-Label. For the Dijkstra-based method, Dijkstra algorithm [4] is used to traverse the user objects needed. The other is based on IS-Label, first use the R-Tree index to store the edges containing the user objects needed, then use the Best-First algorithm to search the edges. IS-Label is used to calculate the shortest distance between two verices.

Our major contributions are summarized as follows.

(1) We define the problem of $k\ell$-NCF query in road-social networks.

(2) We will propose two methods to tackle RS$k\ell$-NCF query. One is based on Dijkstra algorithm, the other is based on IS-Label.

(3) We conduct extensive experiments on different datasets to verify the efficiency of two algorithms.

The rest of the paper is organized as follows: Section 2 describes the related work and Section 3 formalizes the problem. Section 4 presents the method based on Dijkstra algorithm. Section 5 presents the method based on IS-Label. Section 6 presents the experimental results and analysis. Finally, we conclude this paper in Section 7.

2. Related work

2.1. kNN query on road network

Roussopoulos et al. [21] designed a branch-and-bound R-tree [9] traversal algorithm to find the nearest neighbor object to the query point, and then generalize it to finding the k nearest neighbors (kNN). For kNN query on road networks, there are many studies. Hu et al. [12] simplified the network by replacing the graph topology with a set of interconnected tree-based structures called SPIE's, and proposed a lightweight nd index for the SPIE. Lee et al. [16] designed a new system framework ROAD for the spatial object search on road networks. Jiang et al. [14] studied the top-k nearest keyword search problem in a massive graph and proposed algorithms that return the exact answers. Inspired by R-tree, Zhong et al.[29] proposed a height-balanced and scalable index, namely G-tree, to efficiently support three types of location-based queries on road networks. Zhao et al. [28] studied the problem of group nearest compact POI set (GNCS) query and showed that this problem is NP-hard. Ouyang et al. [20] studied the problem of top-k nearest neighbors search on road networks. They proposed an effcient and progressive query processing algorithm to output each result in well-bounded delay. He et al. [10] proposed a framework on correctness-aware kNN queries, which aims at optimizing the system throughput while guaranteeing query correctness on moving objects. Dong et al. [5] presented a direction-aware KNN (DAKNN) query covering moving objects on road networks. Kim et al. [15] proposed the moving view field nearest neighbor (MVFNN) query, which continuously retrieves the nearest object in the query's view field with the change of query location.

2.2. Geo-Social query

Geo-social queries consider the location and social relationship. Liu et al. [18] proposed a new type of query called Circle of Friend Query (CoFQ), which returns a group of friends in a Geo-Social network whose members are close to each other both socially and geographically. Emrich et al. [6] studied the problem of geo-social skyline queries. The returned users are closely connected to the query user, and close to the query location. Ahuja et al. [1] proposed geo-social keyword (GSK) search, and presented three specific GSK queries. Jiang et al. [13] proposed the top-k local user search (TkLUS) query in geo-tagged social media. Sohail et al. [24] proposed Top-k Famous Places (T_kFP) query and Socio-Spatial Skyline Query (SSSQ). For the queries, they proposed three approaches, called Social-First, Spatial-First and Hybrid. There are also researches on group queries. Zhu et al. [30] proposed a family of geo-social group queries (GSGQs) with minimum acquaintance constraints, and devised two index structures, namely SaR-tree and SaR*-tree. Sohail et al.[25] proposed the Geo-Social Group preference Top-k (SG-Top_k) query, which retrieves nearby places popular among a particular group of users based on spatial and social relevance. Ma et al. [19] proposed the personalized geo-social group (PGSG) query, which aims to retrieve a user group and a venue, where each user in the group is socially connected with at least c other users in the group and the maximum distance of all the users in the group to the venue is minimized. Ghosh et al.[8] proposed a novel Top k Flexible Socio Spatial Group Query (Top k-FSSGQ) to find the top k groups of various sizes w.r.t. multiple POIs. Shim et al. [23] proposed the ℓ-cohesive m-ridesharing group (ℓmCRG) query, which retrieves a cohesive ridesharing group by considering spatial, social, and temporal information.

2.3. Road-Social networks query

Road-Social networks query has drawn lots of attention in recent years. Zhao et al. [26] proposed the Reverse Top-k Geo-Social Keyword (RkGSK) query on road networks, and designed the GIM-tree index for the query. Attique et al.[2] proposed geo-social top-k keyword (GSTK) query and geo-social skyline keyword (GSSK) query on road networks. They proposed appropriate indexing frameworks and algorithms to efficiently process these queries. Zhao et al.[27] proposed the diversified top-k geo-social keyword (DkGSK) query on road networks, which considers not only the relevance but also the diversity of the result. Li et al.[17] studied the skyline cohesive group query problem in road-social networks.

In this paper, k-nearest neighbors ℓ-close friends ($k\ell$-NCF) query in road-social networks is closely related to the research of [22]. Shim et al.[22] studied the $k\ell$-NCF query and proposed three approaches for the query: Neighboring Cell Search, Friend-Cell Search, and Personal-Cell Search. The methods proposed in [22] cannot be directly applied to road networks. Therefore, we would study the problem of $k\ell$-NCF query on road networks.

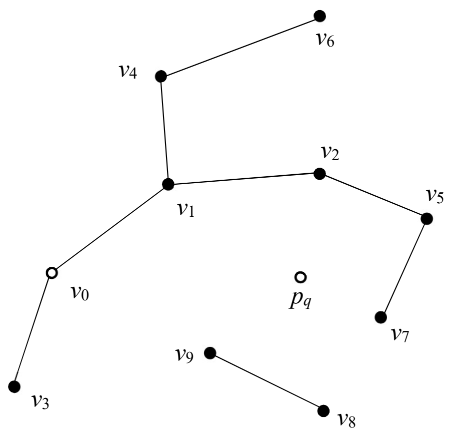
3. Problem definition

The road-social network is composed of a pair of networks, a road network G_r and a social network G_s, denoted as G = (G_r, G_s). The road network is modeled as an undirected weighted graph G_r = (V_r, E_r, W), where V_r is the vertex set, E_r is the edge set, and W is a function, such that $w(n_i, n_j)$ is the weight of edge $(n_i, n_j) \in E_r$. For $(n_i, n_j) \in E_r$, if the id of n_i is less than that of n_j, we call n_i and n_j the starting vertex and the ending vertex of (n_i, n_j) respectively, and vice versa. The social network is modeled as an undirected graph G_s = (V_s, E_s), where V_s is the vertex set (representing users), and E_s is the edge set (representing social relations). The user objects in social network are mapped to the nearest intersection or edge on the road network based on their location. We use $rdist(a, b)$ to represent the shortest path length between a and b on the road network, where a or b could be a query point, vertex or user object.

Definition 1. *(ℓ-hop friend list [22]) $V_v^\ell \subseteq V_s$ denotes the ℓ-hop friend list of v such that:*

$$V_v^\ell = \begin{cases} \{v' | \exists e(v, v') \in E_s\} & (\ell = 1) \\ V_v^{\ell-1} \cup \{v' | \exists e(v', v'') \in E_s \wedge v'' \in V_v^{\ell-1}\} \setminus \{v\} & (\ell > 1) \end{cases} \quad (1)$$

Example 2. Fig. 2 describes the social network containing the user vertex v_0. The 1-hop friend list of v_0 is v_1, v_3. The 2-hop friend list of v_0 consists of v_2, v_4, and the friends of v_0, because v_2 and v_4 are the friends of v_1. In the same way, we can get the 3-hop and 4-hop friends list of v_0, which are shown in Fig. 2.



ℓ-hop	friend list of v_0
1-hop	$\{v_1, v_3\}$
2-hop	$\{v_1, v_2, v_3, v_4\}$
3-hop	$\{v_1, v_2, v_3, v_4, v_5, v_6\}$
4-hop	$\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

Fig. 2. Social network and ℓ-hop friend lists of v_0

The $k\ell$-NCF query is defined in [22]. Based on this, we would give the definition of $k\ell$-NCF query on road networks as follows:

Definition 2. *($k\ell$-NCF on road networks) Given a road-social network G = (G_r, G_s), a query point p_q, a query user u, the number of result elements k, and a friendship degree ℓ, the k-nearest ℓ-close friends ($k\ell$-NCF) query q = (p_q, u, k, ℓ) on road networks finds a result list R = (v_1, v_2, $\ldots$, v_k), such that ($1 \leq i < k$):*

$$R \subseteq V_v^\ell \wedge rdist(p_q, v_i) \leq rdist(p_q, v_{i+1}) \wedge v' \in V_v^\ell \setminus R \wedge rdist(p_q, v_k) \leq rdist(p_q, v') \tag{2}$$

where, friendship degree represents the minimum number of edges (hops) between two data objects in the graph [22].

Example 3. After mapping, we get the road network shown in Fig. 3, where v_0, $\ldots$, v_9 are the user objects shown in Fig. 2. Given a $k\ell$-NCF query $q = (p_q, u = v_0, k = 2, \ell = 3)$ on road networks, the result list is (v_3, v_5). Although v_7 is closer to p_q than v_5, v_7 is not in the 3-hop friend list of v_0, so v_7 is not in the result list.

4. Method based on Dijkstra algorithm

In this section, we will propose a method based on Dijkstra algorithm. Before the $k\ell$-NCF query on road networks is issued, we can prepare some information to speed up query processing. We create adjacency lists for the social network and the road network respectively. For all the user objects in the social network, we create a hash table $UEHm$ with the structure $(v, (e, len))$, where e is the edge on which v locates, and len is the road network distance between v and the starting vertex of e.

In the following, we would introduce some data structures for the method.

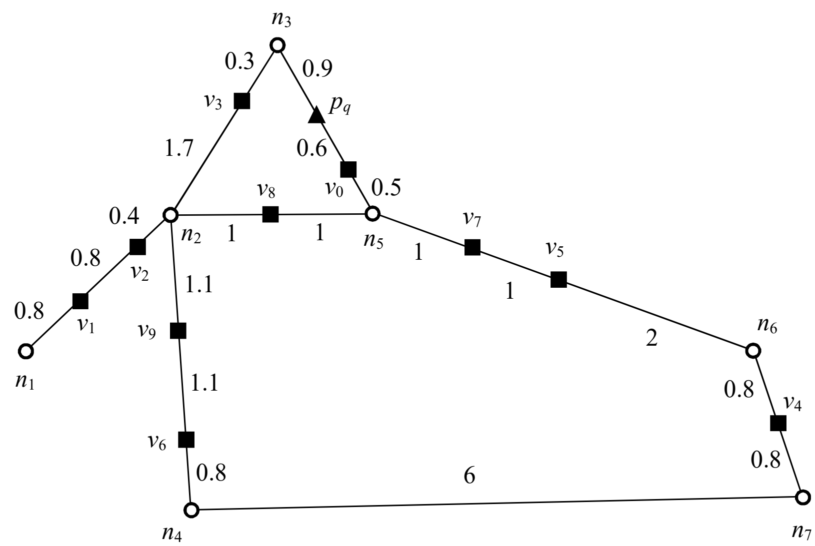


Fig. 3. Road network

$closed$: $closed$ is a hash table to store the vertices which are closed. We call a vertex closed if it has been extracted from the min heap.

$clounvis$: $clounvis$ is a hash table to store the vertices that are closed but not visited. We call a vertex visited if the user objects on all of its adjacent edges have been visited.

hE: hE is a hash table with the structure $(e, UList)$, where $UList$ is a list to store the user object, such that the user object locates on the edge e and belongs to V_u^ℓ (u represents the query user).

The overall procedure of D-RSCNF is summarized as follows:

(1) Create the ℓ-hop friend list V_u^ℓ of a query user u.

(2) Create the hash table hE.

(3) Dijkstra algorithm is used to expand from p_q.

Algorithm 1 describes the query process based on Dijkstra algorithm. Lines 1-2 are initialization. R is a max-heap with a maximum size of k, which is used to store the results. H is a min heap, in which the key is the road network distance from a vertex (or an object) to p_q. For $edgecount$, it is used to record the number of edges visited in hE. In lines 4-5, hE is created by using the user object $v \in V_u^\ell$ and the edge where the object locates.

In line 26, for the user object v on the edge (n_i, n_j), we could calculate $rdist(p_q, v)$ with Formula (3). As shown in Fig. 4, v represents the required user object, $svid$ is the starting vertex, and $tvid$ is the ending vertex. The shortest path length from p_q to v in Algorithm 1 is defined as:

$$rdist(p_q, v) = min\{rdist(p_q, svid) + d(svid, v), rdist(p_q, tvid) + d(tvid, v)\} \quad (3)$$

Algorithm 1: D-RSNCF Query

Input: RS$k\ell$-NCF query q = (p_q, u, k, ℓ), the road-social network $G = (G_r, G_s)$, $UEHm$
Output: Result list R
$R \leftarrow \emptyset$, $H \leftarrow \emptyset$, $edgecount \leftarrow 0$, $count \leftarrow 0$;
$hE \leftarrow \emptyset$, $closed \leftarrow \emptyset$, $clounvis \leftarrow 0$;
compute V_u^ℓ with the adjacency list of the social network G_s;
for *each user object* $v \in V_u^\ell$ **do**
    find the edge e on which v locates using $UEHm$ and update hE;
get the edge $(sqid, tqid)$ on which p_q locates;
insert $(rdist(sqid, p_q), sqid)$ and $(rdist(tqid, p_q), tqid)$ into H;
while $H \neq \emptyset$ **do**
    $(rdist(n_i, p_q), n_i) \leftarrow H$.delMin();
    $closed[n_i] \leftarrow n_i$;
    $clounvis[n_i] \leftarrow rdist(n_i, p_q)$;
    $flag \leftarrow 0$;
    for *each adjacent vertex* n_j *of* n_i **do**
        if n_j *is not in closed* **then**
            $flag \leftarrow 1$;
            if n_j *does not exist in* H **then**
                H.add$(rdist(n_j, p_q), n_j)$;
            else
                update $(rdist(n_j, p_q), n_j)$ in H;
        else
            if *all the adjacent vertices of* n_j *are in closed* **then**
                remove n_j from $clounvis$ table;
            if *edge* (n_i, n_j) *is in* hE **then**
                $edgecount++$;
                for *each user object* v *in* $hE[(n_i, n_j)]$ **do**
                    R.add$(rdist(p_q, v), v)$;
                    $count++$;
                if $hE.size() \leq edgecount$ **then**
                    R is sorted in ascending order by the value of $rdist()$;
                    return R;
                if $count \geq k$ **then**
                    if $R.getRoot().rdist \leq min_{rdist}(clounvis)$ **then**
                        R is sorted in ascending order by the value of $rdist()$;
                        return R;
    if $flag = 0$ **then**
        remove n_i from $clounvis$ table;
R is sorted in ascending order by the value of $rdist()$;
return R;

where $d(svid, v)$ represents the length from $svid$ to v on the edge $(svid, tvid)$, and $d(tvid, v)$ represents the length from $tvid$ to v on the edge $(svid, tvid)$.

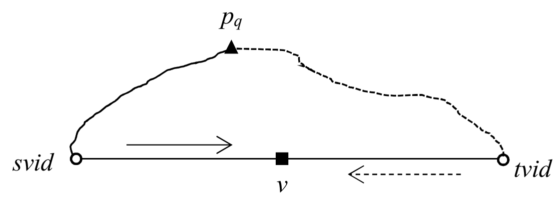


Fig. 4. Paths to be chosen for method based on Dijkstra algorithm

In lines 8-36, Dijkstra algorithm is used to expand. For an object v on the edge $(svid, tvid)$, in order to calculate $rdist(p_q, v)$, both $svid$ and $tvid$ need to be closed before calculation. If a vertex is closed, it needs to enter $closed$ table and $clounvis$ table (lines 10-11). If all adjacent vertices of a vertex have been closed, the vertex can be removed from $clounvis$ table. In line 12, we use the value of $flag$ as 0 to indicate that all adjacent vertices of n_i have been closed. If one of adjacent vertices of n_i is not closed, $flag$ is set to 1 in line 15. In line 19, the update is to find the shortest path length from p_q to n_j and store it in H.

For a vertex in $clounvis$ table, in order to remove it from $clounvis$ table, there are two cases: (1) it is closed after all its adjacent vertices; (2) it is not closed after all its adjacent vertices. For case (1), in lines 35-36, n_i is removed from $clounvis$ table, where n_i is closed after all its adjacent vertices. For case (2), in lines 21-22, n_j is removed from $clounvis$ table, where n_j is closed before its adjacent vertex n_i.

In the following theorem, we would prove the correctness of the termination condition in lines 31-32 of Algorithm 1. In the condition, $count$ is the size of R, $R.getRoot().rdist$ is the maximum distance in R and $min_{rdist}(clounvis)$ is the minimum distance in $clounvis$.

Theorem 1. *If $count \geq k$ and $R.getRoot().rdist \leq min_{rdist}(clounvis)$, then Algorithm 1 could sort and return R to terminate the query correctly.*

Proof. From Algorithm 1, we can see that clounvis is used to store the vertices that are closed but not visited. In order to terminate the query, we should focus on the unvisited user objects. For any unvisited user object v locating on edge (a, b), there are two cases: (1) a or b is closed; (2) Neither a nor b is closed.

For case (1), without loss of generality, let a be in clounvis. Since b is not closed, $rdist(p_q, b) \geq rdist(p_q, a)$. Then, we have $rdist(p_q, v) \geq rdist(p_q, a) \geq min_{rdist}(clounvis)$. Therefore, if $count \geq k$ and $R.getRoot().rdist \leq min_{rdist}(clounvis)$, we have $rdist(p_q, v) \geq R.getRoot().rdist$, which indicates that v cannot or need not replace the user object in R.

For case (2), we have $rdist(p_q, v) \geq min(rdist(p_q, a), rdist(p_q, b)) \geq max_{rdist}(clounvis) \geq min_{rdist}(clounvis)$, where $max_{rdist}(clounvis)$ is the maximum distance in clounvis. Therefore, if $count \geq k$ and $R.getRoot().rdist \leq min_{rdist}(clounvis)$, we have $rdist(p_q, v) \geq R.getRoot().rdist$. ■

Example 4. For Fig. 3, given a query $q = (p_q, u = v_0, k = 2, \ell = 2)$, we illustrate the query process of Algorithm 1. According to the social network in Fig. 2, the 2-hop friend list of v_0 is $\{v_1, v_2, v_3, v_4\}$. Then the hash table hE is created. Based on $UEHm$, we get the edges on which the 2-hop friend list of v_0 locate. The edges are $(n_1, n_2), (n_2, n_3), (n_6, n_7)$, which are stored in hE. And get the edge (n_3, n_5) where p_q locates, so $(0.9, n_3)$ and $(1.1, n_5)$ are put into the min-heap H.

(1) The first removed from H is $(0.9, n_3)$, and n_3 is put into $closed$ and $clounvis$. Then we find the adjacent vertices of n_3, and $(2.9, n_2)$ is put into H. Note that $(2.9, n_5)$ would not replace $(1.1, n_5)$ in H.

(2) The second removed from H is $(1.1, n_5)$. Similarly, n_5 is put into $closed$ and $clounvis$. Then we find the adjacent vertices of n_5 and $(5.1, n_6)$ is put into H. At the moment, n_3 and n_5 are in $closed$, and the edge (n_3, n_5) is not in hE, so we can conclude that there is no 2-hop friend of v_0 on this edge.

(3) The third removed from H is $(2.9, n_2)$, and n_2 is stored in $closed$ and $clounvis$. We find the adjacent vertices of n_2, then $(4.9, n_1)$ and $(5.9, n_4)$ are put into H. For the adjacent vertices of n_2, n_3 and n_5 are in $closed$. For the adjacent vertices of n_3, n_2 and n_5 are in $closed$, so n_3 will be removed from $clounvis$. Since the edge (n_2, n_3) is in hE, we find the user object v_3 on (n_2, n_3), and calculate $min_{rdist}(p_q, v_3)$ according to Formula (3). So $(1.2, v_3)$ is added to R.

(4) The fourth removed from H is $(4.9, n_1)$, and n_1 is put into $closed$ and $clounvis$. Since the adjacent vertex n_2 of n_1 is in $closed$, we find the user objects v_1 and v_2 on the edge (n_1, n_2). Calculate $min_{rdist}(p_q, v_1)$ and $min_{rdist}(p_q, v_2)$ according to Formula (3). So $(3.3, v_2)$ is stored in R. Because n_1 has only one adjacent vertex n_2, and n_2 is in $closed$, we can remove n_1 from $clounvis$.

(5) The fifth removed from H is $(5.1, n_6)$, and n_6 is put into $closed$ and $clounvis$. For the adjacent vertex n_7 of n_6, $(6.7, n_7)$ is put into H. Since the adjacent vertex n_5 of n_6 is in $closed$, and the edge (n_5, n_6) is not in hE, no user object is found. Because all the adjacent vertices of n_5 are in $closed$, we can remove n_5 from $clounvis$.

(6) The sixth removed from H is $(5.9, n_4)$, and n_4 is put into $closed$ and $clounvis$. Since the adjacent vertex n_2 of n_4 is in $closed$, and the edge (n_2, n_4) is not in hE, no user object is found. Because all the adjacent vertices of n_2 are in $closed$, we can remove n_2 from $clounvis$. Now, we have $count \geq 2$ and $R.getRoot().rdist \leq min(clounvis)$, where $R.getRoot().rdist = min_{rdist}(p_q, v_2) = 3.3$ and $min(clounvis) = 5.1$, so we can terminate the query. The process of this example is shown in Table 1.

Table 1. The process of Example 4

Order	H.delMin()	$closed$	$clounvis$	R
1	$(0.9, n_3)$	$\{n_3\}$	$\{(n_3, 0.9)\}$	$\emptyset$
2	$(1.1, n_5)$	$\{n_3, n_5\}$	$\{(n_3, 0.9), (n_5, 1.1)\}$	$\emptyset$
3	$(2.9, n_2)$	$\{n_3, n_5, n_2\}$	$\{(n_5, 1.1), (n_2, 2.9)\}$	$\{(1.2, v_3)\}$
4	$(4.9, n_1)$	$\{n_3, n_5, n_2, n_1\}$	$\{(n_5, 1.1), (n_2, 2.9)\}$	$\{(1.2, v_3), (3.3, v_2)\}$
5	$(5.1, n_6)$	$\{n_3, n_5, n_2, n_1, n_6\}$	$\{(n_2, 2.9), (n_6, 5.1)\}$	$\{(1.2, v_3), (3.3, v_2)\}$
6	$(5.9, n_4)$	$\{n_3, n_5, n_2, n_1, n_6, n_4\}$	$\{(n_6, 5.1), (n_4, 5.9)\}$	$\{(1.2, v_3), (3.3, v_2)\}$

Time complexity of Algorithm 1: First, create the ℓ-hop friend list V_u^ℓ for user u, which is equivalent to a breadth-first search process. So, it takes at most $O(|V_s| + |E_s|)$ to find V_u^ℓ. Next, it takes at most $O(|V_s|)$ to create hE. Then, it takes at most $O(|E_r|)$ to find the edge on which p_q locates. The top-k nearest user objects are found with the help of Dijkstra algorithm, so, the time cost is $O((|V_r| + |E_r|) \cdot log|V_r| + |V_s| \cdot t_o)$, where t_o is the time used to compute the shortest distance between p_q and the user object according to Formula (3). To sum up, the time complexity of Algorithm 1 is $O(|V_s| + |E_s| + (|V_r| + |E_r|) \cdot log|V_r|)$.

5. Method based on IS-Label

Because the method based on Dijkstra algorithm traverses the road network from near to far, it is not very advantageous that the user object found by query is far from the query point. Based on this situation, we propose a label-based method. IS-label index [7] is one of the label indexes, and it is also applicable to large graphs. So, we use IS-label index to calculate the minimum distance between p_q and the vertex.

Algorithm 2: L-RSNCF Query

Input: RSkl-NCF query $q = (p_q, u, k, \ell)$, the road-social network $G = (G_r, G_s)$, $UEHm$, IS-Label Index of G_r
Output: Result list R

$R \leftarrow \emptyset$, $hE \leftarrow \emptyset$, $count \leftarrow 0$;
$Queue \leftarrow$ NewPriorityQueue();
compute V_u^ℓ with the adjacency list of the social network G_s;
for *each user object* $v \in V_u^\ell$ **do**
    find the edge e on which v locates using $UEHm$ and update hE;
create the R-Tree $index$ for all edges in hE;
get the edge $(sqid, tqid)$ where p_q locates;
$Queue$.Enqueue$(index.Root, MinDist(p_q, index.Root))$;
while *not* $Queue.isEmpty()$ **do**
    $temp \leftarrow Queue$.Dequeue();
    if $temp$ *is an object* **then**
        for *each user object* v *on the edge* $temp$ **do**
            R.add$(rdist(p_q, v), v)$;
            $count++$;
        if $count \geq k$ **then**
            if $R.getRoot().rdist \leq Queue.getRoot().edist$ **then**
                break;
    else
        for *each child* c *of* $temp$ **do**
            $Queue$.Enqueue$(c, MinDist(p_q, c))$;
R is sorted in ascending order by the value of $rdist()$;
return R;

Algorithm 2 describes the query process based on IS-Label index, which adopts the best-first traversal [11]. In line 1, R is the same as that in Algorithm 1. In line 2, $Queue$ is a min heap, in which the key is the minimum Euclidean distance from a node or an object to p_q (Definition 3). Lines 3-5 is the same as lines 3-5 in Algorithm 1. Line 6 is to create the R-Tree $index$ with the edges in hE. In lines 8-20, best-first traversal is used to search user objects with the R-Tree $index$. In lines 11-17, if the element removed from the priority queue $Queue$ is an edge, we access the user objects on this edge, and calculate the minimum distance between p_q and user objects using the IS-Label index. In lines 15-17, we could jump out of the while loop early, which is proved in Theorem 2. In lines 19-20, we process each child c of $temp$.

We should note that for each edge (a, b) on the road network, we create an object with Minimum Bounding Rectangle (MBR) containing a and b for the R-Tree $index$.

Definition 3. *(MinDist Distance [21]) In Euclidean space of dimension n, the minimum distance between a point q and MBR $N(s, u)$ is denoted by $MinDist(q, N(s, u))$, which is defined as follows:*

$$MinDist(q, N) = \sum_{i=1}^{n} |q_i - r_i|^2, r_i = \begin{cases} s_i, & q_i < s_i \\ u_i, & q_i > u_i \\ q_i, & \text{otherwise} \end{cases} \tag{4}$$

As shown in Fig. 5, $sqid$ and $tqid$ are the starting vertex and ending vertex of the edge on which p_q locates respectively. For the user object v, $svid$ and $tvid$ are the starting vertex and ending vertex of the edge on which v locates respectively. The shortest path length from p_q to v in Algorithm 2 is defined as:

$$rdist(p_q, v) = min\{rdist1(p_q, v), rdist2(p_q, v), rdist3(p_q, v), rdist4(p_q, v)\} \tag{5}$$

where
$rdist1(p_q, v) = d(p_q, sqid) + rdist(sqid, svid) + d(svid, v)$
$rdist2(p_q, v) = d(p_q, tqid) + rdist(tqid, tvid) + d(tvid, v)$
$rdist3(p_q, v) = d(p_q, sqid) + rdist(sqid, tvid) + d(tvid, v)$
$rdist4(p_q, v) = d(p_q, tqid) + rdist(tqid, svid) + d(svid, v)$

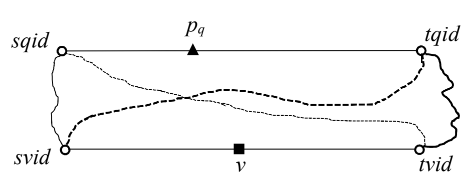


Fig. 5. Paths to be chosen for method based on IS-Label

In the following theorem, we would prove the correctness of the termination condition in lines 15-16 of Algorithm 2. In the condition, $count$ is the size of R, $R.getRoot().rdist$

is the maximum distance in R and $Queue.getRoot().edist$ is the minimum Euclidean distance in $Queue$.

Theorem 2. *If $count \geq k$ and $R.getRoot().rdist \leq Queue.getRoot().edist$, then Algorithm 2 could sort and return R to terminate the query correctly.*

Proof. R is a max-heap with a maximum size of k, so $R.getRoot().rdist$ is the maximum distance in R. $Queue$ is a min heap, $Queue.getRoot().edist$ is the minimum Euclidean distance in $Queue$. In order to terminate the query, we should focus on the unvisited user objects. For any unvisited user object v locating on the edge (a, b), we have $rdist(p_q, v) \geq min(rdist(p_q, a), rdist(p_q, b)) \geq Queue.getRoot().edist$. Therefore, if $count \geq k$ and $R.getRoot().rdist \leq Queue.getRoot().edist$, we have $rdist(p_q, v) \geq R.getRoot().rdist$, which shows that v cannot or need not replace the user object in R. ■

Example 5. For Fig. 3, given a query $q = (p_q, u = v_0, k = 2, \ell = 2)$, we illustrate the query process of Algorithm 2. The 2-hop friend list of v_0 and hE are the same as those in Example 4. Next, we create an R-Tree index for all edges in hE. The MBR of the edge in hE is shown in Fig. 6 using dashed rectangle.

(1) The first object to be removed from $Queue$ is (n_2, n_3), and we find the user object v_3 on the edge (n_2, n_3). Then we calculate $rdist(p_q, v_3)$. There are four paths from p_q to v_3, which are $p_q \rightarrow n_3 \rightarrow n_2 \rightarrow v_3$, $p_q \rightarrow n_5 \rightarrow n_3 \rightarrow v_3$, $p_q \rightarrow n_3 \rightarrow n_3 \rightarrow v_3$, $p_q \rightarrow n_5 \rightarrow n_2 \rightarrow v_3$. $rdist(n_2, n_3), rdist(n_2, n_5), rdist(n_3, n_5)$ can be calculated using the IS-Label index. According to Formula (5), we get $rdist1(p_q, v_3) = 4.6$, $rdist2(p_q, v_3) = 3.4$, $rdist3(p_q, v_3) = 1.2$, $rdist4(p_q, v_3) = 4.8$. So we get $rdist(p_q, v_3) = rdist3(p_q, v_3) = 1.2$. Now we have $R = \{(1.2, v_3)\}$.

(2) The second object to be removed from $Queue$ is (n_1, n_2), and we find v_1 and v_2 on the edge (n_2, n_3). Using the IS-Label index, we get $rdist(p_q, v_1) = 4.1$ and $rdist(p_q, v_2) = 3.3$. Now we have $R = \{(1.2, v_3), (3.3, v_2)\}$.

(3) When (n_6, n_7) is the root of $Queue$, we have $count \geq 2$ and $R.getRoot().rdist \leq Queue.getRoot().edist$. Then we can jump out of the loop early, although v_4 on the edge (n_6, n_7) has not been processed.

Time complexity of Algorithm 2: In Algorithm 2, we use the traversal of R-Tree to replace the traversal of the road network using Dijkstra algorithm. According to [11], it takes at most $O(|E_r| \cdot log|E_r|)$ to traverse R-Tree. The time of other parts are similar to that of Algorithm 1, so the time complexity of Algorithm 2 is $O(|V_s| + |E_s| + |E_r| \cdot log|E_r|)$.

6. Experiments

6.1. Datasets and setting

This experiment uses two social network datasets and three road network datasets for testing. The social network datasets are Brightkite(BR) and Gowalla(GA) [3]. They come from http://snap.stanford.edu/data/. There are three road network datasets: (1)BAY; (2)San Francisco (SF); (3)City of San Joaquin County (TG). BAY comes from http://users.diag.uniroma1.it/challenge9/download.shtml. SF and TG come from http://www.cs.utah.edu/~

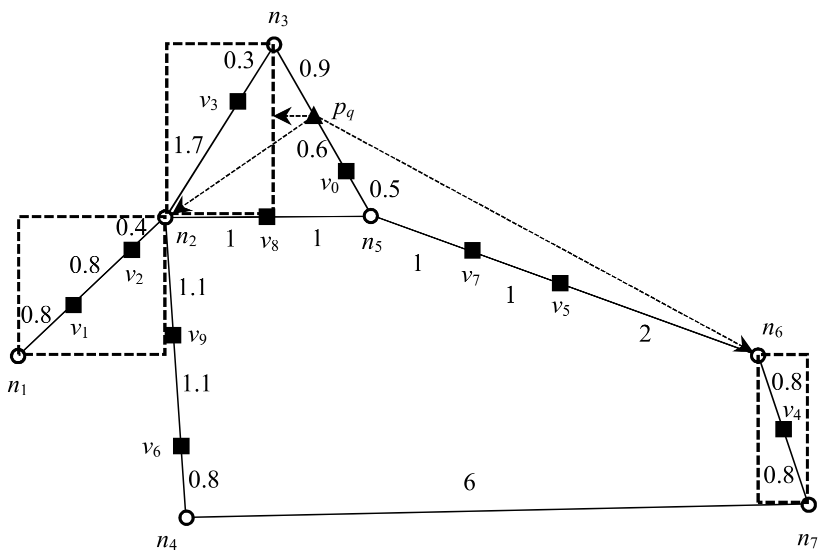
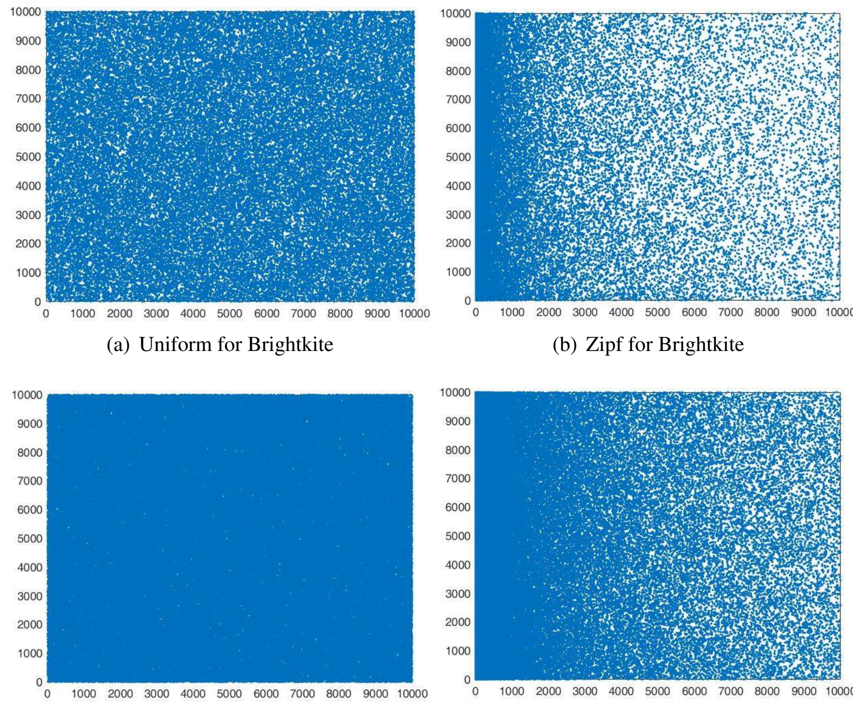


Fig. 6. Road network with MBRs

lifeifei/SpatialDataset.htm. The information of the road networks are shown in Table 2. Standardize the latitude and longitude of the user locations in the two social networks into a flat two-dimensional space, and then map the user to the nearest intersection or edge on the road network according to the coordinates. BR_rangeBAY and GA_rangeBAY represents the social network dataset of Brightkite and Gowalla within the range of BAY respectively. The information of the social networks are shown in Table 3. We use two real datasets:(1) BR_rangeBAY + BAY;(2) GA_rangeBAY + BAY. BR_rangeBAY + BAY represents the road-social network formed by BR_rangeBAY and BAY. GA_rangeBAY + BAY represents the road-social network formed by GA_range BAY and BAY. In these two real data sets, the user objects are sparse, so we also use synthetic datasets for testing.

The synthetic datasets retain the number of vertices in the two social networks and friendship relationship between users. The uniform function and the Zipf function are used to randomly allocate the location information of all user vertices, and the range of the horizontal and vertical coordinates is [0, 10000]. Uniform distribution is used for BR and GA in Fig. 11(a) and (c) respectively. Zipf distribution is used for BR and GA in Fig. 11(b) and (d) respectively. We get four synthetic datasets: (1)UBR+TG; (2)ZBR + TG; (3)UGA + SF; (4)ZGA + SF. UBR+TG represents the dataset formed by BR and TG, where uniform distribution is used for BR. ZBR+TG represents the dataset formed by BR and TG, where Zipf distribution is used for BR. UGA + SF represents the dataset formed by GA and SF, where uniform distribution is used for GA. ZGA + SF represents the dataset formed by GA and SF, where Zipf distribution is used for GA. Table 4 shows the density of road-social networks, where the density denotes the ratio of the number of vertices in the social network to the number of edges on the road network.



(a) Uniform for Brightkite (b) Zipf for Brightkite

(c) Uniform for Gowalla (d) Zipf for Gowalla

Fig. 7. Data object distributions of the synthetic datasets

Table 2. Statistics of the road network datasets

Road network	Vertices	Edges
BAY	321270	400086
SF	174956	223001
TG	18263	23874

Table 3. Statistics of the social network datasets

Social network	Vertices	Edges
BR_rangeBAY	2756	3819
GA_rangeBAY	4794	11086
Brightkite(BR)	58228	214078
Gowalla(GA)	196591	950327

Table 4. Density of the road-social network datasets

Road-social network	Density
BR_rangeBAY + BAY	0.007
GA_rangeBAY + BAY	0.012
UBR + TG	2.439
ZBR + TG	2.439
UGA + SF	0.882
ZGA + SF	0.882

Implementation: We implement all the algorithms on the Eclipse platform using Java. The experimental machine configuration is the Windows 10 operating system, Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz and 8G RAM. In this experiment, we measure the average value at each experiment performed 100 times with random query users and vertices.

Parameters setting:: Parameters setting are shown in Table 5, where ℓ is the friendship degree, k is the number of results, and rd is the Euclidian distance between the query point location p_q and the location of the query user u. The length unit of the data set BR_RangeBAY + BAY and GA_RangeBAY + BAY is kilometer. The social network datasets of the other four datasets are synthesized based on the range of the road network TG and SF. The coordinate range of TG and SF is [0, 10000], and the range of parameter rd for the synthetic datasets is [100, 200], [300, 400], [700, 800], [1500, 1600], [3100, 3200], and the default value is [1500, 1600].

Table 5. Parameters setting

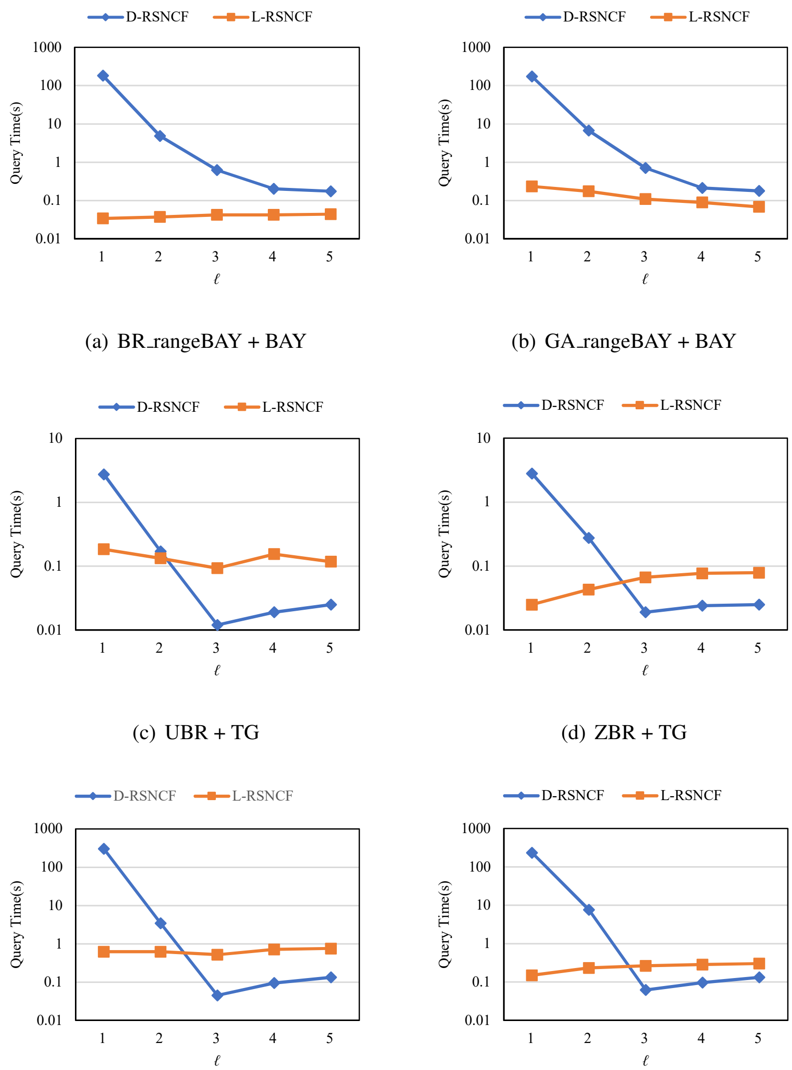
Parameter	Range	Default
ℓ	$1, 2, 3, 4, 5$	3
k	$10, 20, 30, 40, 50$	20
rd (for real dataset)	$[1, 2], [3, 4], [7, 8], [15, 16], [31, 32]$	$[15, 16]$

6.2. Performance Evaluation

In order to test the effect of the experiment, RS$k\ell$-NCF query algorithm based on Dijkstra (D-RSNCF) and RS$k\ell$-NCF query algorithm based on IS-Label index (L-RSNCF) were compared on the datasets of six road-social networks.

(1) Effect of friendship degree ℓ on query time

Fig. 8 demonstrates the effect of varying ℓ. In Fig. 8(a), we set $k = 10$. As shown in Fig. 8(a) and (b), with the increase of ℓ, the query speed of D-RSNCF algorithm is accelerated. This is because with the increase of ℓ, more user objects will be added to the ℓ-hop friend list of the query user u. Then the query range will become smaller. The query speed of L-RSNCF is much faster than that of D-RSNCF. The road network used in Fig. 8(a) and (b) is the largest in the three road network data sets, while few user objects



(a) BR_rangeBAY + BAY (b) GA_rangeBAY + BAY

(c) UBR + TG (d) ZBR + TG

(e) UGA + SF (f) ZGA + SF

Fig. 8. Effect of ℓ on query time

are mapped to the road network. This situation has a greater impact on D-RSNCF query algorithm, so D-RSNCF will be much slower in this case. From Fig. 8(a) and (b), we can see that ℓ has little effect on L-RSNCF.

As shown in Fig. 8(c) and (d), when ℓ reaches a certain value, D-RSNCF is faster than L-RSNCF. The reason may be that UBR + TG and ZBR + TG have the largest density in Table 4. Then D-RSNCF may traverse less edges for these two datasets than other datasets. For larger ℓ, there may be more user objects that meet the friendship degree. So with ℓ increasing, D-RSNCF may traverse less edges. In Fig. 8(e) and (f), the experimental effect is similar to that shown in Fig. 8(c) and (d).

(2) Effect of numbers of result k on query time

Fig. 9 demonstrates the effect of varying k. In general, the query time of both methods increases with the increase of k. The reason is that both methods need to traverse more edges with k increasing. We can see that the influence of k on L-RSNCF is not obvious. In Fig. 9(c)-(f), D-RSNCF is faster than L-RSNCF in most cases, which is similar to the case in Fig. 8(c)-(f). From Fig. 8 and 9, we can see that for the six datasets, the higher the density of the data set, the shorter the query time, and vice versa.

(3) Effect of distance rd on query time

Fig. 10 shows the efficiency of the two query algorithms by changing the parameter rd. As shown in Fig. 10(a) and (b), the query time of D-RSNCF increases with the increase of rd. With rd increasing, most of the ℓ-hop friends of the query user u may be far away from the location of p_q. D-RSNCF is based on Dijkstra algorithm, so it will traverse more edges to find the result in the ℓ-hop friends of u.

As shown in Fig. 10(c) and (e), the user objects are uniformly distributed. Then the number of user objects in a given range is relatively stable, independent of the value of rd. So the query time of D-RSNCF is relatively stable with the increase of rd. In Fig. 10(d) and (f), the user objects follow the Zipf distribution. The number of edges needs to be traversed by D-RSNCF is uncertain, so the query time of D-RSNCF is uncertain with rd increasing.

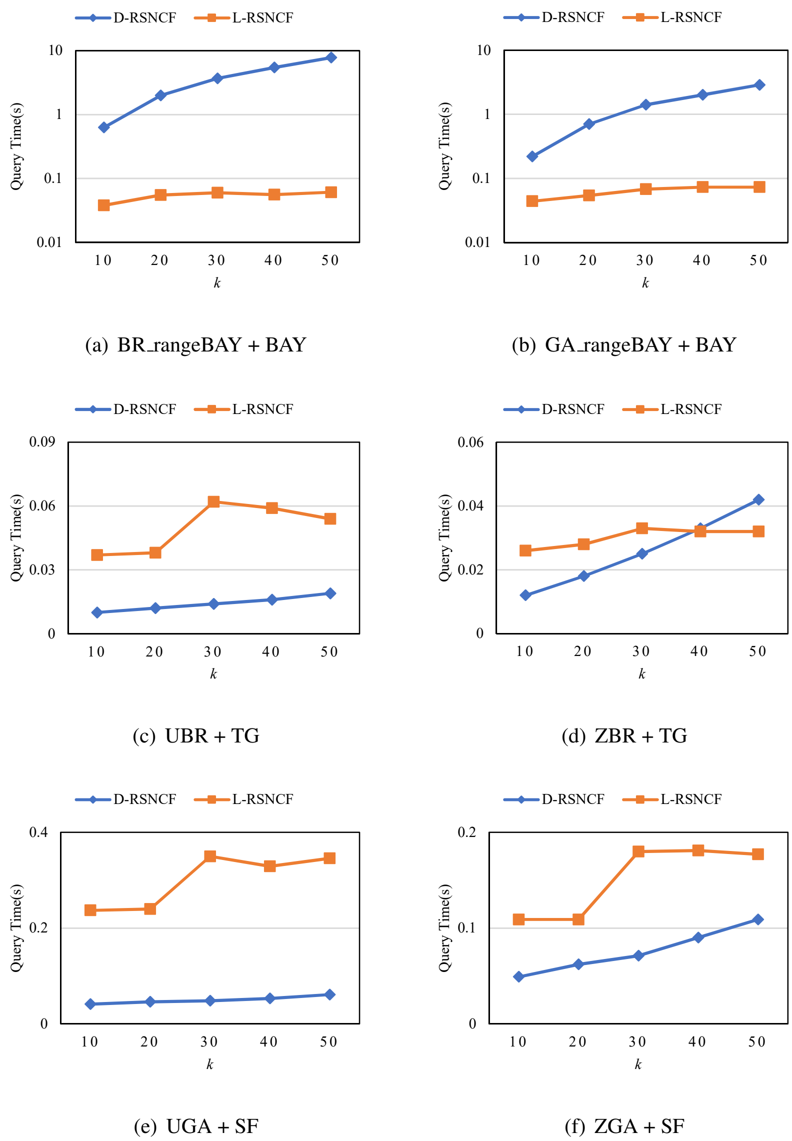
For L-RSNCF, by using R-Tree it traverses only the edges containing the ℓ-hop friends of the query user u, so the running time is uncertain, as shown in Fig. 10. With the increase of rd, the running time of L-RSNCF does not change significantly.

6.3. Discussion

In this section, 6 datasets are used to test the two algorithms. D-RSNCF is not as efficient as L-RSNCF in most cases, but D-RSNCF is more efficient than L-RSNCF on those four synthetic datasets in most cases. The reason is that these four synthetic datasets have much higher density than the two real datasets and D-RSNCF is based on Dijkstra algorithm. For high-density dataset, the search range becomes smaller, so D-RSNCF is more efficient. For dataset with lower density, the search range of D-RSNCF will become larger and the efficiency will become lower. L-RSNCF is based on IS-Label index, so the change of dataset density has no obvious impact on the running time of L-RSNCF.

7. Conclusion

This paper makes an in-depth exploration of the k-nearest ℓ-close friends ($k\ell$-NCF) query in road-social networks. The RS$k\ell$-NCF query algorithm based on Dijkstra (D-RSNCF)



(a) BR_rangeBAY + BAY

(b) GA_rangeBAY + BAY

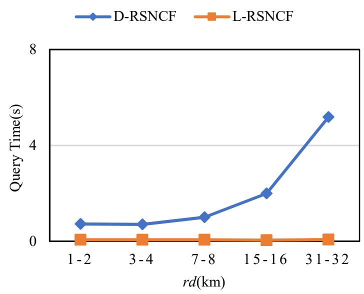
(c) UBR + TG

(d) ZBR + TG

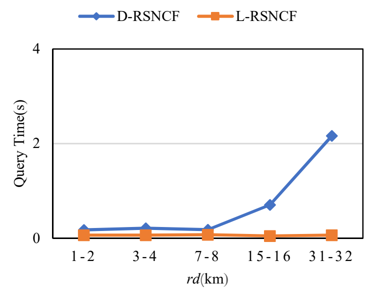
(e) UGA + SF

(f) ZGA + SF

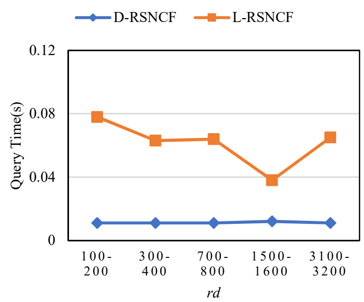
Fig. 9. Effect of k on query time



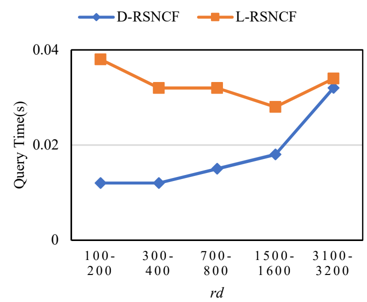
(a) BR_rangeBAY + BAY



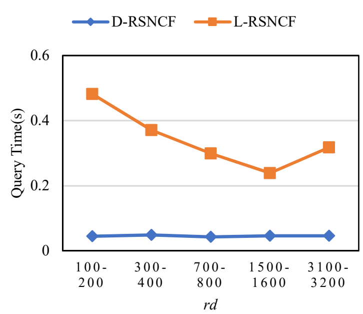
(b) GA_rangeBAY + BAY



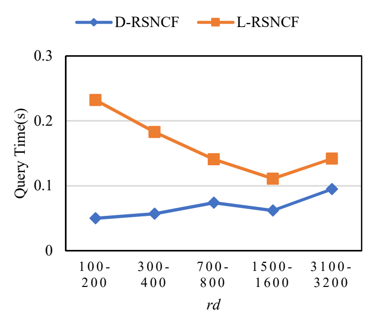
(c) UBR + TG



(d) ZBR + TG



(e) UGA + SF



(f) ZGA + SF

Fig. 10. Effect of rd on query time

and the RS$k\ell$-NCF query algorithm based on IS-Label index (L-RSNCF) are proposed. For both methods, several hash tables are used to speed the query. D-RSNCF is based on Dijkstra algorithm to traverse the user objects needed. L-RSNCF is based on IS-Label and R-Tree to traverse the user objects needed. Real datasets and synthetic datasets are used to test the two algorithms.Through experiments, we find that D-RSNCF is more suitable for dataset with high user object density, while L-RSNCF is just the opposite.

References

1. Ahuja, R., Armenatzoglou, N., Papadias, D., Fakas, G.J.: Geo-social keyword search. In: Proceedings of the 14th International Symposium on Advances in Spatial and Temporal Databases, SSTD 2015, Hong Kong, China. pp. 431–450 (2015)
2. Attique, M., Afzal, M., Ali, F., Mehmood, I., Ijaz, M.F., Cho, H.: Geo-social top-k and skyline keyword queries on road networks. Sensors 20(3), 798 (2020)
3. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: user movement in location-based social networks. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA. pp. 1082–1090 (2011)
4. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
5. Dong, T., Lulu, Y., Cheng, Q., Cao, B., Fan, J.: Direction-aware KNN queries for moving objects in a road network. World Wide Web 22(4), 1765–1797 (2019)
6. Emrich, T., Franzke, M., Mamoulis, N., Renz, M., Züfle, A.: Geo-social skyline queries. In: Proceedings of the 19th International Conference on Database Systems for Advanced Applications, DASFAA 2014, Part II, Bali, Indonesia. vol. 8422, pp. 77–91 (2014)
7. Fu, A.W., Wu, H., Cheng, J., Wong, R.C.: IS-LABEL: an independent-set based labeling scheme for point-to-point distance querying. Proceedings of the VLDB Endowment 6(6), 457–468 (2013)
8. Ghosh, B., Ali, M.E., Choudhury, F.M., Apon, S.H., Sellis, T., Li, J.: The flexible socio spatial group queries. Proceedings of the VLDB Endowment 12(2), 99–111 (2018)
9. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD Annual Meeting on Management of Data, SIGMOD 1984, Boston, Massachusetts, USA. pp. 47–57 (1984)
10. He, D., Wang, S., Zhou, X., Cheng, R.: An efficient framework for correctness-aware kNN queries on road networks. In: Proceedings of the 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China. pp. 1298–1309 (2019)
11. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Transactions on Database Systems 24(2), 265–318 (1999)
12. Hu, H., Lee, D.L., Xu, J.: Fast nearest neighbor search on road networks. In: Proceedings of the 10th International Conference on Extending Database Technology, EDBT 2006, Munich, Germany. pp. 186–203 (2006)
13. Jiang, J., Lu, H., Yang, B., Cui, B.: Finding top-k local users in geo-tagged social media data. In: Proceedings of the 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea. pp. 267–278 (2015)
14. Jiang, M., Fu, A.W., Wong, R.C.: Exact top-k nearest keyword search in large networks. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia. pp. 393–404 (2015)
15. Kim, W., Shim, C., Heo, W., Yi, S., Chung, Y.D.: Moving view field nearest neighbor queries. Data & Knowledge Engineering 119, 58–70 (2019)
16. Lee, K.C.K., Lee, W., Zheng, B., Tian, Y.: ROAD: A new spatial object search framework for road networks. IEEE Transactions on Knowledge and Data Engineering 24(3), 547–560 (2012)

17. Li, Q., Zhu, Y., Yu, J.X.: Skyline cohesive group queries in large road-social networks. In: Proceedings of the 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA. pp. 397–408 (2020)
18. Liu, W., Sun, W., Chen, C., Huang, Y., Jing, Y., Chen, K.: Circle of friend query in geo-social networks. In: Proceedings of the 17th International Conference on Database Systems for Advanced Applications, DASFAA 2012, Part II, Busan, South Korea. pp. 126–137 (2012)
19. Ma, Y., Yuan, Y., Wang, G., Bi, X., Wang, Y.: Personalized geo-social group queries in location-based social networks. In: Proceedings of the 23rd International Conference on Database Systems for Advanced Applications, DASFAA 2018, Part I, Gold Coast, QLD, Australia. pp. 388–405 (2018)
20. Ouyang, D., Wen, D., Qin, L., Chang, L., Zhang, Y., Lin, X.: Progressive top-k nearest neighbors search in large road networks. In: Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA]. pp. 1781–1795 (2020)
21. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA. pp. 71–79 (1995)
22. Shim, C., Kim, W., Heo, W., Yi, S., Chung, Y.D.: Nearest close friend search in geo-social networks. Information Sciences 423, 235–256 (2018)
23. Shim, C., Sim, G., Chung, Y.D.: Cohesive ridesharing group queries in geo-social networks. IEEE Access 8, 97418–97436 (2020)
24. Sohail, A., Cheema, M.A., Taniar, D.: Social-aware spatial top-k and skyline queries. The Computer Journal 61(11), 1620–1638 (2018)
25. Sohail, A., Hidayat, A., Cheema, M.A., Taniar, D.: Location-aware group preference queries in social-networks. In: Proceedings of the 29th Australasian Database Conference on Databases Theory and Applications, ADC 2018, Gold Coast, QLD, Australia. pp. 53–67 (2018)
26. Zhao, J., Gao, Y., Chen, G., Jensen, C.S., Chen, R., Cai, D.: Reverse top-k geo-social keyword queries in road networks. In: Proceedings of the 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA. pp. 387–398 (2017)
27. Zhao, J., Gao, Y., Ma, C., Jin, P., Wen, S.: On efficiently diversified top-*k* geo-social keyword query processing in road networks. Information Sciences 512, 813–829 (2020)
28. Zhao, S., Xiong, L.: Group nearest compact POI set queries in road networks. In: Proceedings of the 20th IEEE International Conference on Mobile Data Management, MDM 2019, Hong Kong, SAR, China. pp. 106–111 (2019)
29. Zhong, R., Li, G., Tan, K., Zhou, L., Gong, Z.: G-tree: An efficient and scalable index for spatial search on road networks. IEEE Transactions on Knowledge and Data Engineering 27(8), 2175–2189 (2015)
30. Zhu, Q., Hu, H., Xu, C., Xu, J., Lee, W.: Geo-social group queries with minimum acquaintance constraints. The VLDB Journal 26(5), 709–727 (2017)

Zijun Chen received the bachelor's degree from the Northeast Heavy Machinery Institute, China, the master's degree from Yanshan University, and the PhD degree from Fudan University in 2002, all in computer science. Since 1995, he has been with the School of Information Science and Engineering, Yanshan University, Qinhuangdao, China, where he is currently a professor. His research interests include moving object databases, spatio-temporal databases and graph databases.

Ruoyu Jiang received the bachelor's degree in software engineering from Hebei Normal University, China, in 2018. She received the master's degree in computer technology from Yanshan University, China, in 2021. Her research interest includes geo-social networks.

Wenyuan Liu received the bachelor's and master's degrees from the Northeast Heavy Machinery Institute, China, and the PhD degree from the Harbin Institute of Technology in 2000, all in computer science. Since 1996, he has been with the School of Information Science and Engineering, Yanshan University, Qinhuangdao, China, where he is currently a professor. His research interests include wireless sensor networks and mobile networks.

Received: August 30, 2021; Accepted: July 30, 2022.