Xu XJ, Bao JS, Yao B *et al.* Reverse furthest neighbors query in road networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(1): 155–167 Jan. 2017. DOI 10.1007/s11390-017-1711-5

Reverse Furthest Neighbors Query in Road Networks

Xiao-Jun Xu[1,2], Jin-Song Bao[3,*], Bin Yao[4,5,*], *Member, CCF, ACM, IEEE*, Jing-Yu Zhou[4,5]
Fei-Long Tang[4,5], *Member, CCF, ACM, IEEE*, Min-Yi Guo[4,5], *Senior Member, IEEE, Member, CCF, ACM* and Jian-Qiu Xu[6]

[1] *School of Software, Beijing Institute of Technology, Beijing 100081, China*

[2] *First Research Institute of Ministry of Public Security, Beijing 100048, China*

[3] *College of Mechanical Engineering, Donghua University, Shanghai 200051, China*

[4] *Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai 200240, China*

[5] *Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*

[6] *College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China*

E-mail: 18611288159@163.com; bao@dhu.edu.cn; {yaobin, zhou-jy, tang-fl, guo-my}@cs.sjtu.edu.cn
jianqiu@nuaa.edu.cn

Received February 26, 2016; revised August 12, 2016.

Abstract Given a road network $G = (V, E)$, where $V(E)$ denotes the set of vertices(edges) in G, a set of points of interest P and a query point q residing in G, the reverse furthest neighbors (RFN_R) query in road networks fetches a set of points $p \in P$ that take q as their furthest neighbor compared with all points in $P \cup \{q\}$. This is the monochromatic RFN_R (MRFN_R) query. Another interesting version of RFN_R query is the bichromatic reverse furthest neighbor (BRFN_R) query. Given two sets of points P and Q, and a query point $q \in Q$, a BRFN_R query fetches a set of points $p \in P$ that take q as their furthest neighbor compared with all points in Q. This paper presents efficient algorithms for both MRFN_R and BRFN_R queries, which utilize landmarks and partitioning-based techniques. Experiments on real datasets confirm the efficiency and scalability of proposed algorithms.

Keywords reverse furthest neighbor, road network, landmark, hierarchical partition

1 Introduction

Spatial database has been extensively studied in database community as it supports many applications from people's daily life to scientific research[1-10]. For instance, people use online map services to plan their trips. The query processing for sensor networks[11] needs the design of location-aware algorithms. In this work, we study a query type in road networks that finds wide applications. Given a road network $G = (V, E)$, where $V(E)$ denotes the set of vertices(edges) in G, a set of points of interest P and a query point q residing in G, we are interested in retrieving the set of points in P that take q as their furthest neighbors (in terms of the shortest path distance) compared with all points in P, i.e., collecting q's reverse furthest neighbors (RFN_R). This problem is referred to as the monochromatic reverse furthest neighbor (MRFN_R) query. It naturally has a bichromatic version as well (BRFN_R). Specifically, the query contains a set of query points Q residing in G and one point $q \in Q$. The goal in this case is to find a set of points $p \in P$ so that they all take q as their furthest neighbors compared with all points in Q.

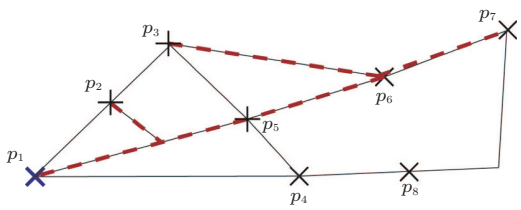
The examples of RFN_R are provided in Fig.1. In

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. U1636210, 61472039, 61373156, 91438121, and 61672351, the National Basic Research 973 Program of China under Grant No. 2015CB352403, the National Key Research and Development Program of China under Grant Nos. 2016YFB0700502, 2016YFC0803000, and 2016YFB0502603, the Scientific Innovation Act of Science and Technology Commission of Shanghai Municipality under Grant No. 15JC1402400, and Microsoft Research Asia.

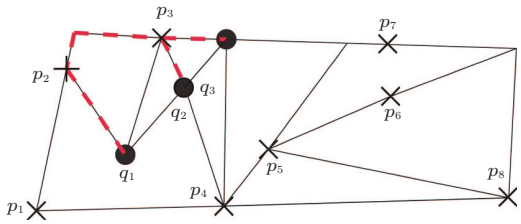
*Corresponding Author

Fig.1(a), the dashed line shows that p_1 is p_7's furthest neighbor, and p_7 is one of p_1's RFN_R. In Fig.1(b), since the distance from p_2 to q_3 is further than the distance from p_2 to q_1 and q_2, p_2 should be one of q_3's RFN_R w.r.t. $\{q_1, q_2, q_3\}$.



$P = \{p_1, \ldots, p_8\}$, $q = p_1$

(a)



$P = \{p_1, \ldots, p_8\}$, $Q = \{q_1, q_2, q_3\}$

(b)

Fig.1. RFN_R query examples. (a) MRFN_R query example. (b) BRFN_R query example.

The motivation to study the RFN_R queries is largely inspired by an important query type that has been extensively studied recently, namely, the reverse nearest neighbor (RNN) queries[12-14]. Intuitively, an RNN query finds a set of points taking the query point as their nearest neighbors and it exists in both the monochromatic and bichromatic versions. Both of these two versions of RNN query have been extended to the road networks. Many applications that are behind the studies of the RNN queries naturally have the corresponding "furthest" versions, including RNN queries on the road networks. Consider the next two examples for the MRFN_R and BRFN_R queries.

Example 1. Suppose a large group of friends want to find one of their houses to have a party, and some one (say Alice) would like to learn the set of friends who take her as their furthest neighbors compared with other friends. This has an implication that these friends are highly unlikely to visit Alice. Hence, Alice should put more efforts in persuading these friends.

Example 2. For a large collection of points of interest in a region, every point would like to learn the set of sites that take itself as their furthest neighbor compared with other points of interest. This has an implication that visitors to these sites (i.e., its reverse furthest neighbors) are highly unlikely to visit this point. Ideally, it should put more efforts in advertising itself to these sites.

In the above two examples, people are more concerned about road distance rather than Euclidean distance. Thus the reverse furthest neighbor query on the road network is more applicable to this issue than the reverse furthest neighbor query in the Euclidean space.

To the best of our knowledge, there are few discussions about RFN_R problems in large-scale road networks. The brute-force search algorithms for these problems are obviously too expensive to be of any practical use. Hence, large-scale road networks are calling for practical, efficient algorithms for these problems. More importantly, by taking the furthest neighbors, The RFN_R problems are different from the RNN problems in the geometric nature. Hence, we need to design new algorithms to process the RFN_R queries more efficiently by taking the new geometric perspectives into account.

Contributions. This work presents efficient algorithms for MRFN_R and BRFN_R problems. Specifically, we 1) propose two novel algorithms (the LM algorithm and the HP algorithm) for the MRFN_R query, 2) propose two novel algorithms (the PFC-BRFN_R algorithm and the FVCPar algorithm) for the BRFN_R query, 3) propose two algorithms to improve the performance of finding the furthest neighbor, which is the fundamental function of the query algorithms for both MRFN_R and BRFN_R problems, and 4) conduct comprehensive experiments on real datasets to evaluate the efficiency and scalability of all proposed algorithms.

The paper is organized as follows. Section 2 formulates the problem of the reverse furthest neighbors and Section 3 surveys related work. Section 4 provides novel methods based on landmarks and hierarchical partitioning to answer MRFN_R efficiently. Section 5 presents a progressive method combined with hierarchical partitioning to answer BRFN_R queries. Section 6 reports a comprehensive experimental study with real datasets and Section 7 concludes the paper.

2 Problem Formulation

Let P denote the points of interest (POIs) in a road network. The shortest path distance between any two points p and q is denoted by $||p - q||$, and the furthest neighbor of any point p w.r.t. P in road networks is simply defined as follows.

Definition 1. *The furthest neighbor of p to a dataset P is defined as $fn(p, P) = p^*$ s.t. $p^* \in P$, for*

$\forall p' \in P$ *and* $p' \neq p^*$, $||p^* - p|| \geqslant ||p' - p||$. *Ties are broken arbitrarily.*

The monochromatic RFN_R query is formally defined as follows.

Definition 2. *The* MRFN_R *of* q *w.r.t. the dataset* P *is a set of points from* P *that take* q *as their furthest neighbors compared with all points in* P, *i.e.,* $\text{MRFN}_R(q, P) = \{p | p \in P,\ fn(p, P \bigcup \{q\}) = q\}$.

The bichromatic RFN_R query takes additionally a set of query points Q as input, and is formally defined as follows.

Definition 3. *The* BRFN_R *of* $q \in Q$ *w.r.t. the dataset* P *and the query set* Q *is a set of points from* P *that take* q *as their furthest neighbors compared with all other points in* Q, *i.e.,* $\text{BRFN}_R(q, Q, P) = \{p | p \in P, fn(p, Q) = q\}$.

3 Background and Related Work

An interesting query type that has close relationship with RFN_R was defined in [15], in which the goal is to find the set of points from P that take the query point q as their nearest neighbors among all points in the dataset P. This is the monochromatic reverse nearest neighbor query (monochromatic RNN). Due to its wide applications, RNN queries have received considerable attention since its appearance[12,15-21].

The bichromatic RNN also finds many applications[13,15,17,20,22-23]. In this case, the query takes a set of query points Q and a query point $q \in Q$. The set of points returned from P all take q as their nearest neighbors w.r.t. other points in Q. The basic idea here is to use the Voronoi diagram and find the region that corresponds to the query point.

The RNN problem can be extended to graphs and road networks[23]. Generalization to any metric space appeared in [22]. Continuous RNN was explored by [13, 24]. The RNN for moving objects was studied in [25]. Reverse kNN search was examined by [12, 16]. Finally, the RNN for ad-hoc subspaces was solved by [14].

The RFN problem was firstly studied by Yao *et al.*[26] in Euclidean space. They studied both the MRFN and the BRFN versions, which take advantage of the R-tree, furthest Voronoi diagrams and the convex hulls of either the dataset P (in the MRFN case) or the query set Q (in the BRFN case). However, the proposed solutions do not apply for our situation since it is hard for the R-tree and the convex hull to define in the road networks. Hence, we need to design new indexes and query algorithms to efficiently answer the RFN_R query.

To the best of our knowledge, the BRFN_R query has not been studied in the literature. The MRFN_R query was firstly studied by Tran *et al.*[27]. For a particular MRFN_R query, each possible solution is checked by finding the partition it locates in, expanding the adjunct Voronoi partitions of the candidate partition until we find some point that is further than the candidate or we have browsed through all partitions. However, their method is only applicable when the density of points is low and would retreat to the brute-force method when the number of points is large. To solve the problem, we propose several novel approaches to index the road networks and filter the candidates, which are also useful for the BRFN_R query.

4 Monochromatic Reverse Furthest Neighbors in Road Networks

The basic algorithm for MRFN_R (denoted as BFS) was proposed by Tran *et al.*[27], which can be summarized as follows. For each $p \in P$, we check whether q is p's furthest neighbor, which is referred to as $isFN(p, q)$. Specifically, $isFN(p, q)$ expands from p using Dijkstra's algorithm until it meets the query node q. If q is the last node met in $P \bigcup \{q\}$, then q is p's furthest neighbor.

The BFS method takes $O(|V|^2 \log |V|)$ time, which does not scale well for large datasets. In the following, we propose two efficient algorithms for the MRFN_R query. One is based on the landmarks technique[28] (denoted as LM in Subsection 4.1). The other one is based on the graph partitioning technique (denoted as HP in Subsection 4.2). Besides, we discuss the efficient implementation of $isFN(p, q)$ based on the partitioning technique in Subsection 4.3.

4.1 LM Algorithm

The BFS solution will call $isFN(p, q)$ for each $p \in P$. $isFN(p, q)$ may visit lots of nodes in P if q is far away from p. Hence, we focus on reducing the number of nodes visited by $isFN(p, q)$. Specifically, we utilize the landmarks to prune some points, which are not too far away from p. By using landmarks technique, we need to carefully choose a small (constant) number of landmarks, and then compute and store the shortest path distances between all vertices and each of these landmarks. Lower-bound distances between any two vertices in road networks are computed in constant time using these distances in combination with the triangle inequality. We denote this algorithm as the LM

algorithm. This solution consists of two parts: the pre-processing and the query processing.

4.1.1 Pre-Processing

In this step, a set L of points in the road network are selected as the landmarks[28]. Then, the distances between landmarks and all other points in the road network are computed by using Dijkstra's algorithm. All these distances will be stored for the query processing.

The strategies of selecting the landmarks can be critical to the performance of the LM algorithm. In [28], Goldberg and Harrelson examined several methods for selecting landmarks to facilitate the lower-bound distance estimation between two points. However, our method needs to estimate the upper-bound distances. According to our observation, a set of landmarks selected with uniform distribution work the best among those methods proposed in [28] in our situation.

4.1.2 Query Processing

The LM algorithm is shown in Fig.2. For each node u in P, we check if its distance to q is farther than the distance $||q-f||$ by using the triangle inequality. By doing this, if we still cannot prune u, we call $isFN(p,q)$ to determine whether it is a final result.

```
LM (query q; nodes P; landmarks L)
1.  Initialize S = ∅
2.  Let f be q's farthest landmark
3.  For each node u in P
4.      For each node l in L
5.          If (||q − l|| + ||l − u|| < ||q − f||)
6.              Continue;
7.          Else if (isFN(u, q))
8.              S = S ⋃ {u};
9.  Return S
```

Fig.2. LM algorithm.

4.2 HP Algorithm

In this subsection, we propose to partition the road networks to further improve the efficiency of the query processing.

Suppose G is divided into a set of edge-disjoint subgraphs $SG_1, SG_2, \cdots, SG_m$. The graph partitioning has been extensively studied in many communities since 1970s for different purposes[29-32]. Most of the partitioning methods can be adopted for answering the MRFN$_R$ queries. In this subsection, we introduce the hierarchical partitioning (HP) method and the query algorithm based on it.

4.2.1 HP Tree

To ease the discussion, we introduce the following definitions.

Definition 4. *The boundary nodes set of partition SG_i is defined as*

$$bd_{SG_i} = \{p | \exists edge(p,p') \in E \wedge p \in V_{SG_i} \wedge p' \notin V_{SG_i}\},$$

where V_{SG_i} is the nodes set of SG_i.

Definition 5. *The upper(lower) bound distance from a node p to a partition SG_i, denoted as $ub^p_{SG_i}(lb^p_{SG_i})$, is defined as the maximum(minimum) distance from p to any node in SG_i. The diameter of a partition SG_i is defined as $\Phi_{SG_i} = \max\limits_{p \in V_{SG_i}} ub^p_{SG_i}$. Similarly, the upper(lower) bound distance between two nodes p, q, denoted as $ub^p_q(lb^p_q)$, is defined as the maximum(minimum) distance from p to q.*

Definition 6. *The furthest upper(lower) bound distance of a partition SG_i, denoted as $fub_{SG_i}(flb_{SG_i})$, is the maximum(minimum) distance from any node p inside SG_i to its furthest neighbor. Similarly, the furthest upper(lower) bound distance from node p to its furthest neighbor, denoted as $fub_p(flb_p)$ is the maximum(minimum) distance from node p to its furthest neighbor.*

Lemma 1. *If $ub^q_{SG_i} < flb_{SG_i}$ and $p \in V_{SG_i}$, then p cannot be q's RFN.*

Proof. If p is an RFN of q in SG_i, we have $ub^q_{SG_i} \geqslant ||q-p|| \geqslant flb_{SG_i}$, which contradicts the condition. □

Next, we discuss how to efficiently build the HP tree and compute the bounds. Our method is based on the hierarchical encoded path view (HEPV) structure[33] and the graph Voronoi diagram[34]. The HEPV structure constructs a hierarchical graph by fragmenting the flat graph into partitions and by pushing up border nodes to generate the hierarchy. The Voronoi diagram is a famous structure of computational geometry. There is a straightforward equivalent in graph theory which can be efficiently computed. In our algorithms, we use the HEPV structure to build a hierarchical graph and the graph Voronoi diagram to fragment each flat graph into subgraphs.

The HP tree can be built in a top-down fashion. The nodes in a road network are divided into m partitions, and each partition is divided into m subpartitions, recursively. To divide a partition SG_i into m subpartitions, we use the Erwig and Hagen's algorithm[34]. An example of the HP tree is shown in Fig.3. Fig.3(a) shows the road network and the two-layer partitioning. The road network consists of 10 points. SG_1, SG_2 and

SG_3 are the first layer partitions. The others are the second layer partitions. Fig.3(b) shows the tree structure of the partitions on this road network. Once we have the HP tree, we also need some auxiliary information enabling the query algorithm. Within one partition, the distances between all boundary nodes of its subpartitions are pre-computed. The furthest neighbors of the boundary nodes within and out of the partition should also be pre-computed.

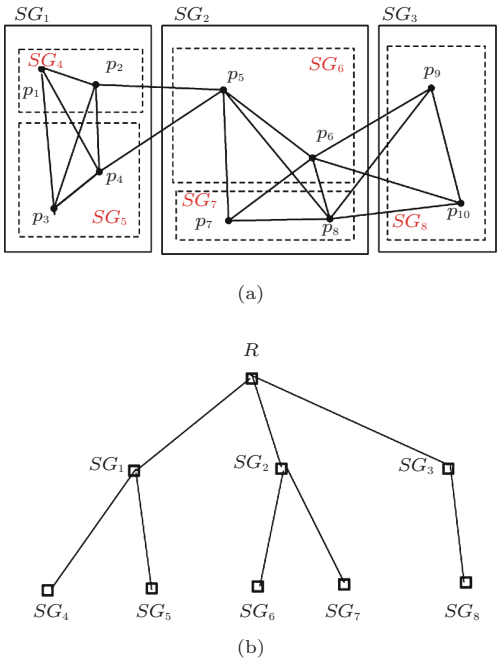


Fig.3. HP tree. (a) Partitioning example. (b) Corresponding tree structure.

By Definition 5, $ub^p_{SG_i}$ is the upper bound of the distances from node p to any nodes in a partition SG_i. When $p \in bd_{SG_i}$, we compute $ub^p_{SG_i}$ by finding p's furthest neighbor in SG_i. When $p \in V_{SG_i} \wedge p \notin bd_{SG_i}$, $ub^p_{SG_i} = \Phi_{SG_i}$. When $p \notin SG_i$, since any path from p to SG_i must go through some boundary nodes of SG_i, we can use the upper bounds between p and SG_i's boundary nodes to estimate $ub^p_{SG_i}$ as follows:

$$ub^p_{SG_i} = \max_{b \in bd_{SG_i}} (ub^b_p + ub^b_{SG_i}).$$

To compute flb_{SG_i}, we need to estimate the minimum of the distances between the nodes in SG_i and their furthest neighbors. We introduce the following lemma.

Lemma 2. *For* $\forall p \in V_{SG_i}, \forall b \in bd_{SG_i}, \forall f \in V$,

$$||b - f|| - ub^b_{SG_i} \leqslant ||p - fn(p)||.$$

Proof. By triangle inequality, we have $||b-f||-||p-b|| \leqslant ||p-f||$. Also, by Definition 5, $ub^b_{SG_i} \geqslant ||p-b||$ if $b \in V_{SG_i}$, hence we have:

$$\begin{aligned} g(b,f) &= ||b-f|| - ub^b_{SG_i} \leqslant ||b-f|| - ||p-b|| \\ &\leqslant ||p-f|| \leqslant ||p-fn(p)||. \end{aligned}$$ □

The inequality above shows flb_{SG_i} can be obtained by selecting a set of boundary nodes of SG_i and any nodes in G and calculating the maximum value of $g(b,f)$. Note that flb_{SG_i}s can be pre-computed during the construction of the HP tree.

4.2.2 Query Algorithm

The HP algorithm is described in Fig.4. The algorithm traverses the HP tree in the breadth-first style. For each visited SG_i, we check if it can be pruned by Lemma 1. If SG_i cannot be pruned, we keep adding its subpartitions into $\mathcal{L}$. For a leaf partition, we call the LM algorithm. Take the points in Fig.3(a) as an example. Suppose p_3 is the query point. In the first round, we only need to push SG_2 and SG_3 into the queue according to Lemma 1. This is because p_1, p_2, and p_4 in SG_1 cannot take p_3 as their furtherest neighbor considering the points in SG_3.

```
HP(query q; HP tree H; landmarks L)
1.  Initialize a queue L = ∅
2.  Push root node of H into L
3.  While (L ≠ ∅)
4.      e = L.pop()
5.      If (e is a partition)
6.          If (ub^q_e ⩾ flb_e)
7.              If (e has subpartitions)
8.                  For each subpartition e' ∈ e
9.                      L.push(e')
10.             Else
11.                 R = LM(q, V_e, L)
12.                 For each node e' ∈ R
13.                     L.push(e')
14.     Else If (isFN(e, q))
15.         S = S ⋃ {u}
16.     Return S
```

Fig.4. HP algorithm.

4.3 Improvement on $isFN(p,q)$ by Partitioning

Recall that the function $isFN(p,q)$ is implemented by the Dijkstra's algorithm. With the help of the HP tree, we can implement $isFN(p,q)$ in a more efficient way. We have the following property.

Lemma 3. *For* $\forall u \in V$, $\forall SG_i \subset G$, *if* $\exists p' \in V$, $ub^u_{SG_i} < ||u - p'||$, *then* $\forall p \in V_{SG_i}$, p *cannot be* u*'s furthest neighbor.*

Proof. It is indicated by Definition 5. □

With the help of Lemma 3, we design the novel algorithm for $isFN(p, q)$ (Fig.5). In a nutshell, we traverse the partitions of G in a descending order of $ub^q_{SG_i}$s. If we reach a partition in the leaf level of the HP tree, we calculate the exact distances from q to the nodes in the partition. Whenever an exact distance from a node to q supersedes the $ub^q_{SG_i}$s of all remaining partitions, we terminate the algorithm and return the node as the answer.

```
isFNPar(node p; query q; HP tree H)
1.  Initialize a priority queue L and push the root of H
    into L
2.  While (L ≠ ∅)
3.      e = L.pop()
4.      If (e is a partition)
6.          If (ub^q_e ⩾ flb_q)
7.              If (e has subpartitions)
8.                  For each subpartition e' ∈ e
9.                      e'.distance = ub^q_{SG_i}
10.                     L.push(e')
11.             Else
12.                 For each node e' ∈ e
13.                     Compute the distance d between q
                        and e'
14.                     e'.distance = d
15.                     L.push(e')
16.     Else If (e is a node)
17.         While (L.top().distance = e.distance)
18.             S = S ⋃ {L.pop()}
19.         If p ∈ S
20.             Return True
21.         Else
22.             Return False
```

Fig.5. isFNPar algorithm.

5 Bichromatic Reverse Furthest Neighbors in Road Networks

In this section, we consider the BRFN_R problem. We use a progressive algorithm to solve the problem and the graph-partitioning technique to accelerate it.

In the BFS algorithm, we could still use Dijkstra's algorithm to solve the BRFN_R problem. An expansion is performed from each node p in the road network G until query q is met. If all nodes in Q have been visited before the expansion reaches q, p is one of q's RFN w.r.t. Q.

5.1 PFC-BRFN_R Algorithm

For a set of points P, the furthest Voronoi diagram of P, denoted as $\mathcal{FD}(P)$, is similar to the well-known Voronoi diagram[35] of P except that the space is partitioned with the furthest Voronoi cells instead of the nearest Voronoi cells.

The furthest Voronoi diagram can also be built in road networks, which is quite similar to its counterpart in Euclidean space. The only difference is that in road networks, the furthest Voronoi cell is not necessarily a convex polygon. Instead, the cell consists of some nodes in road networks.

We denote the furthest Voronoi cell of a node q w.r.t. Q as $fvc(q, Q)$. We know $fvc(q, Q)$ is q's RFN w.r.t. Q. Hence, the BRFN_R problem can be solved by finding a method to construct $fvc(q, Q)$ effectively.

5.1.1 Algorithm

We use a progressive strategy to compute $fvc(q, Q)$. Although bisector lines in Euclidean space cannot be used to separate the furthest Voronoi cell in a road network, Erwig and Hagen's algorithm[34] naturally splits the road network by the nodes' distances to their nearest seed nodes.

The detail of the algorithm is described in Fig.6. We maintain a set S to save the points that are potential results. Initially, S is set to the point set P. At each iteration we take a node $q' \in Q$, using Erwig and Hagen's algorithm to split S into two parts depending on whether a node in S is closer to q or q'. Those nodes closer to q than q' are then removed. After each $q' \in Q$ is visited, the set S shrinks to $fvc(q, Q)$.

```
PFC-BRFN_R(query q; query set Q; point set P)
1.  S = P
2.  For (i = 1, 2, ..., m)
3.      Choose q' ∈ Q that has not been visited
4.      Divide S into two parts by the distances from
        the nodes in S to q and q' using Erwig and
        Hagen's algorithm
5.      Remove the part of S that is closer to q
6.      If (S = ∅ )
7.          Return ∅
8.  Return S
```

Fig.6. PFC-BRFN_R algorithm.

The algorithm in Fig.6 terminates whenever S is empty, which indicates if at each iteration q' is chosen smartly, the algorithm can be executed faster in the case where there is no solution to this BRFN_R problem. Heuristically, choosing the next q' that is close to the chosen nodes is not a good choice. This is because we can expect the splitting result of Erwig and Hagen's algorithm is similar to that of the last iteration for two close points. Based on this thought, we choose q' by taking a point far away from both q and the nodes that have been chosen.

5.2 FVCPar Algorithm

The bottleneck of the PFC-BRFN$_R$ algorithm is mainly in the splitting procedure. A method is proposed in this subsection to accelerate the splitting progress with the pruning power of the HP tree.

5.2.1 Bound Property

To ease the discussion, we denote a permutation of the query set Q as $Q_k = (q_1, q_2, ..., q_k)$. Hence, the PFC-BRFN$_R$ algorithm can be viewed as computing $fvc(q, Q_k)$ from $fvc(q, Q_{k-1})$ progressively.

Lemma 4. *Given a query q, a query set Q and a partition SG_i, we have*:

1) *if* $lb^q_{SG_i} > ub^{q_k}_{SG_i}$, $V_{SG_i} \cap fvc(q, Q_{k-1}) \subset fvc(q, Q_k)$;

2) *if* $ub^q_{SG_i} < lb^{q_k}_{SG_i}$, $V_{SG_i} \not\subset fvc(q, Q_k)$.

Proof. It is indicated by the definition of ub, lb and $fvc(q, Q)$. □

We could still adopt the partitioning strategy mentioned in Subsection 4.3 to calculate these bounds.

5.2.2 Algorithm

To compute $fvc(q, Q_k)$, we need to split $fvc(q, Q_{k-1})$ to two parts. To accomplish this, we do a preorder traversal of the HP tree.

In Fig.7, we maintain a queue $\mathcal{L}$ and push the root partition of the HP tree into $\mathcal{L}$. When an entry e in $\mathcal{L}$ is popped, we first check if it contains some nodes from $fvc(q, Q_{k-1})$ and discard those irrelevant subpartitions. For those subpartitions containing nodes from $fvc(q, Q_{k-1})$, we try to decide whether it is part of $fvc(q, Q_k)$ by Lemma 4. If these subpartitions pass the filtering of Lemma 4, we insert them into $\mathcal{L}$.

If e does not have subpartitions, we need to split the points in e based on the distance between q, q_k and these points. Generally, this procedure is done by constructing a shortcut graph G' based on the HP tree and running Erwig and Hagen's algorithm on this shortcut graph.

Specifically, we construct G' by adding e and the partitions containing q and q_k. Additionally, in order to maintain the connectivity of G', we need to add into G' the paths between the boundary nodes. Recall that we have pre-computed all the shortcut distances between any two (intra or inter) boundary nodes. In G', we only need to store the shortcut distances that are involved in e and the partitions containing q and q_k. It is easy to see that the splitting result on G' is the same with that on the original graph G, but with the less node access.

```
FVCPar (candidates set S, query q, node q_k, HP tree H)
1.  Initialize queue L = ∅
2.  Push the root element of H into L
3.  While ( L ≠ ∅)
4.      e = L.pop()
5.      If (e ∩ S = ∅)
6.          Continue
7.      Else If (lb^q_e > ub^{q_k}_e)
8.          Continue
9.      Else If (lb^{q_k}_e > ub^q_e)
10.         S = S − e
11.     Else If (e has subpartitions)
12.         For each subpartition e' ∈ e
13.             L.push(e')
14.     Else
15.         Build the shortcut graph G' including e,
            q, q_k and shortcuts
16.         Use Erwig and Hagen's algorithm to split G'
17.         Remove the part closer to q from S
18.     Return S
```

Fig.7. FVCPar algorithm.

6 Experiments

Our implementations were achieved in C++. All experiments were executed on a Linux machine with an Intel 3.07 GHz CPU and 4 GB memory. We adopted a disk-based storage model to represent the road network, which groups network nodes based on their connectivity and distance, as proposed in [36]. For all RFN$_R$ algorithms, we reported the total execution time. By default, 1 000 queries were generated for each experiment and we reported the average value.

Datasets. The real datasets were obtained from the digital chart of the world server, where points define the road networks from California (CA), San Francisco (SF) and USA (US). They also contain a large number of points of interest (e.g., restaurants, resorts). These datasets are available online[①].

The road networks for California (CA, 21 047 nodes, 21 692 edges) and San Francisco (SF, 174 955 nodes, 223 000 edges) were used to evaluate the performance of our RFN$_R$ algorithms with different sizes of road networks.

6.1 Evaluation of the MRFN$_R$ Algorithms

Experimental Setup. The datasets CA and SF were used to demonstrate the performance of our MRFN$_R$ algorithms in various situations. Queries were randomly selected from the datasets.

[①] www.cs.utah.edu/∼lifeifei/SpatialDataset.htm, Dec. 2016.

In both LM and HP algorithms, 64 landmarks were chosen through the network uniformly if not explicitly noted. For the HP tree, we took 2 levels and 20 sub-partitions for each partition (441 partitions totally) as the default setup.

Overall Performance. We demonstrated the effectiveness of the MRFN$_R$ algorithms in this paragraph. Fig.8(a) shows the execution time of the two proposed algorithms against the BFS approach. It is clear that both LM and HP are much more efficient than the BFS algorithm. The HP algorithm is the fastest algorithm, only taking less than 0.2% running time of the BFS algorithm.

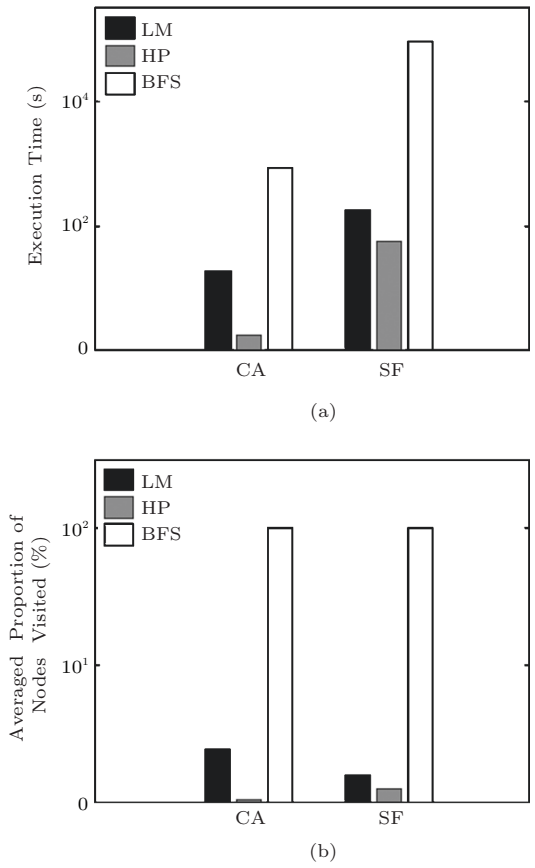


Fig.8. Performance comparison of MRFN$_R$ algorithms and BFS. (a) Execution time. (b) Average number of visited nodes.

Besides, Fig.8(b) presents the averaged proportion of nodes checked by each algorithm after pruning. Both LM and HP only visit less than 3% nodes, demonstrating the effectiveness of our pruning procedures. Another observation is that not only does the partitioning technique accelerate the pruning, but it also excludes more false positives compared with LM because of the additional pruning information provided by the HP tree.

Size of the Landmark Set. In this paragraph we would examine how the amount of the landmarks (denoted as $|L|$) affects the performance of the LM algorithm. We discuss the situations that the solution exists (denoted as $S = \emptyset$) or does not exist (denoted as $S \neq \emptyset$) separately because they perform differently when $|L|$ increases. Fig.9 indicates that when the solution exists, $|L|$ has little influence on neither the number of visited nodes nor the execution time of the LM algorithm. It is mainly because in this situation most false positives can be excluded with only a small set of landmarks and adding extra landmarks provides little improvement.

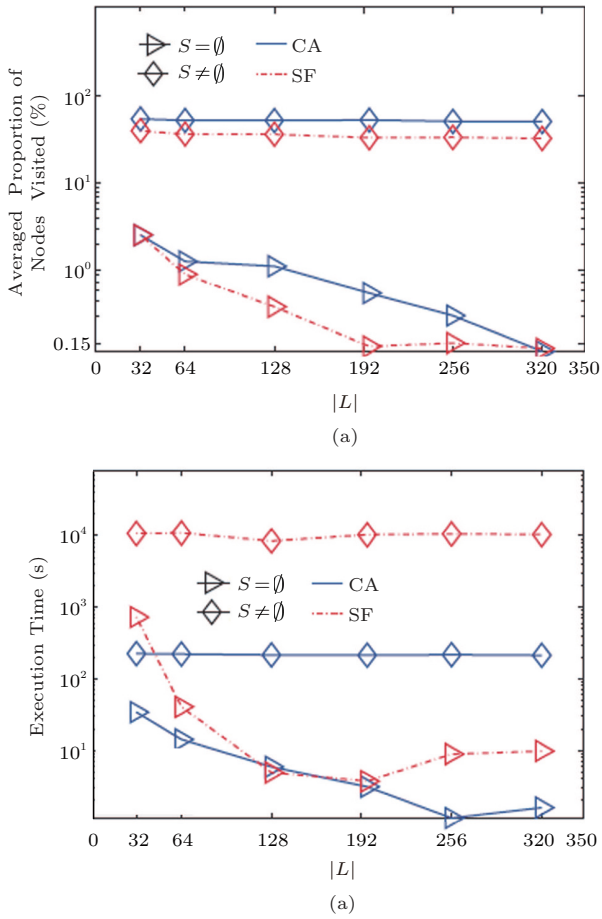


Fig.9. Effect of number of landmarks on LM algorithm's performance. (a) Averaged proportion of nodes visited. (b) Execution time.

In the situation that the solution does not exist, increasing $|L|$ paid off. More landmarks exclude more false positives, and less nodes are visited. However, the cost of pre-processing and computation increases linearly with $|L|$, and therefore we can expect that there is a best choice for $|L|$. This result can be observed in Fig.9(b): with the increase of $|L|$, the execution time

decreases first, but increases again when $|L|$ passes the nadir.

Choice of the HP Tree. Two parameters can affect the performance of the HP algorithm: the amount of total partitions (denoted as $|HP|$) and the depth of the partition tree. In Fig.10 we provide a comprehensive view about the performance of the HP algorithm in different configurations.

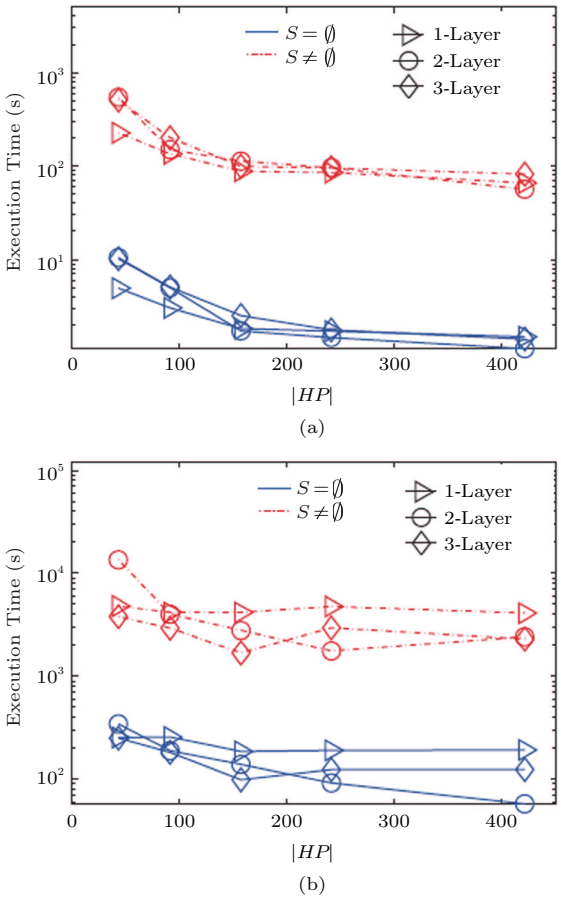


Fig.10. Effect of $|HP|$ and depth of partition tree on the performance of the HP algorithm. (a) CA. (b) SF.

Intuitively, more partitions provide more accurate bounds and thus bring better performance. This can be observed in Fig.10. For CA, the query when $|HP|$ is 420 only takes about one tenth of the time spent when $|HP|$ is 40. However, with a large $|HP|$, adding more partitions seems to be less effective due to extra pruning cost.

When $|HP|$ is small, HP trees with less layers outperform. This is because at the leaf level, it has smaller partitions and thus provides more accurate bounds. But with a larger $|HP|$, higher HP trees can cut more partitions at once and thus perform better. This can be observed in Fig.10. One-layer HP trees almost always outperform when $|HP|$ is small, but the two-layer and the three-layer HP trees scale better with the increase of $|HP|$.

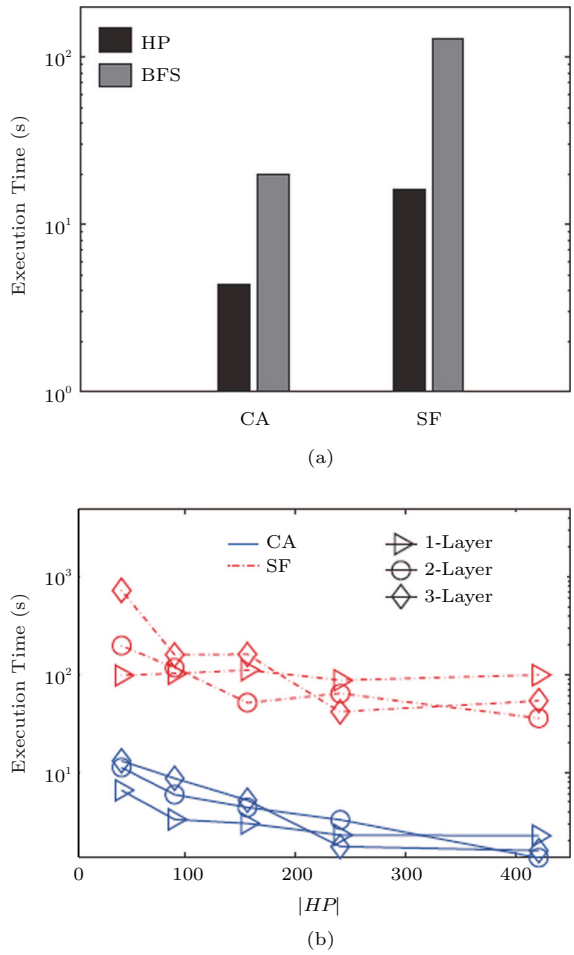


Fig.11. Effect of HP algorithm and depth of partition tree on the performance $isFN(p,q)$. (a) Overall performance. (b) Partition choice.

Accelerating $isFN(p,q)$ with Partitioning. In Fig.11, the performance of our novel method to compute $isFN(p,q)$ is examined against the brute-force solution. It can be seen that our method improves the efficiency of $isFN(p,q)$ by an order of magnitude. We also consider the performance of different configurations of the HP tree in Fig.11(b), which indicates similar results with MRFN_R query: larger $|HP|$ provides better performance; when $|HP|$ is small, lower depth is better; higher depth scales better with the increase of $|HP|$.

6.2 Evaluation of the BRFN_R Algorithms

Experimental Setup. In this subsection, the same datasets were used to examine the performance of our

BRFN$_R$ algorithms and the effect of parameters. The query set was chosen randomly in P. The query set size $|Q|$ was set to 1 280 if not explicitly denoted.

Overall Performance and Scalability. In Fig.12 we examine the overall performance of the algorithms with $|Q|$ scaling from 80 to 1 280. The first observation is that both the PFC-BRFN$_R$ algorithm and the FVCPar algorithm outperform the BFS method significantly. When $|Q|$ is 1 280, the FVCPar algorithm spends less than 1% of the time of the BFS algorithm to answer a query.

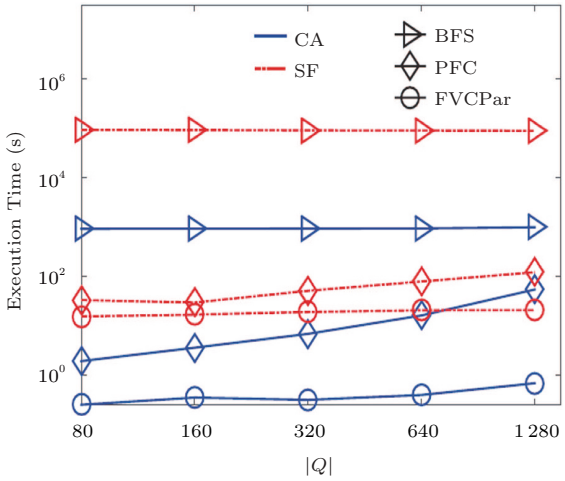


Fig.12. BRFN$_R$: overall performance and scalability.

The second observation is that the BFS algorithm has almost constant computation cost with different $|Q|$ since it visits every possible solution no matter what the query set is. On the other side, the execution time of both the PFC-BRFN$_R$ and the FVCPar algorithms increases linearly with $|Q|$, and the increase of the FVCPar is slower.

Choice of the HP Tree. In Fig.13 we examine the performance of the FVCPar algorithm under different HP tree configurations.

In the case that the solution does not exist, it seems that the pruning power of adding partitions does not benefit the performance much. When the solution exists, the situation is similar to the HP algorithm in MRFN$_R$ case: adding partitions can significantly improve the performance, but the effect of adding partitions decreases when $|HP|$ becomes larger.

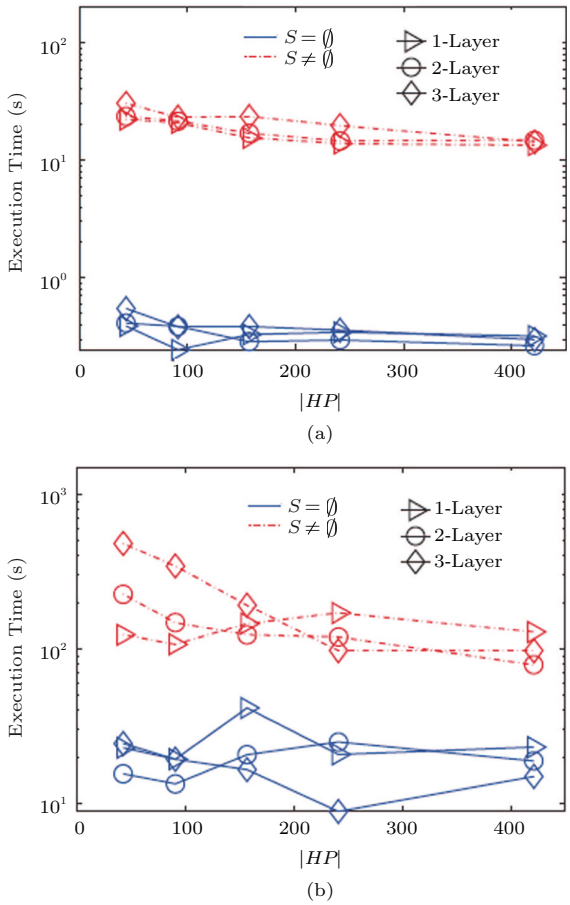


Fig.13. BRFN$_R$: choice of the HP tree. (a) CA. (b) SF.

7 Conclusions

This paper studied the reverse furthest neighbor queries that have many real-life applications. Our work solved the RFN$_R$ queries in both monochromatic and bichromatic versions. We proposed novel methods combining the pruning power of the landmarks technique, the HP tree and the furthest Voronoi cell. Our future work includes generalizing our algorithms to deal with moving points and continuous queries, and extending the method to answer reverse k-furthest neighbors.

There also exist several interesting studies on path planning[37-42], spatial and social information processing and understanding[43-52], and network information processing[53-60], which may be considered in our future studies.

References

[1] Lin X H, Kwok Y K, Wang H, Xie N. A game theoretic approach to balancing energy consumption in heterogeneous wireless sensor networks. *Wireless Communications and Mobile Computing*, 2015, 15(1): 170-191.

[2] Hao J Y, Leung H F, Ming Z. Multiagent reinforcement social learning toward coordination in cooperative multiagent systems. *ACM Transactions on Autonomous and Adaptive Systems*, 2015, 9(4): Article No.20.

[3] Tan L J, Lin F Y, Wang H. Adaptive comprehensive learning bacterial foraging optimization and its application on vehicle routing problem with time windows. *Neurocomputing*, 2015, 151: 1208-1215.

[4] Xu L, Hu Q H, Hung E, Chen B W, Tan X, Liao C R. Large margin clustering on uncertain data by considering probability distribution similarity. *Neurocomputing*, 2015, 158: 81-89.

[5] Chen H, Ni D, Qin J, Li S L, Yang X, Wang T F, Heng P A. Standard plane localization in fetal ultrasound via domain transferred deep neural networks. *IEEE Journal of Biomedical and Health Informatics*, 2015, 19(5): 1627-1636.

[6] Luo J P, Li X, Chen M R, Liu H W. A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows. *Information Sciences*, 2015, 316: 266-292.

[7] Li H C, Wu K S, Zhang Q, Ni L M. CUTS: Improving channel utilization in both time and spatial domain in WLANs. *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(6): 1413-1423.

[8] Cao W M, Liu N, Kong Q C, Feng H. Content-based image retrieval using high-dimensional information geometry. *Science China Information Sciences*, 2014, 57(7): 1-11.

[9] Lai Z H, Xu Y, Chen Q C, Yang J, Zhang D. Multilinear sparse principal component analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 2014, 25(10): 1942-1950.

[10] Chen W S, Wang W, Yang J W, Tang Y Y. Supervised regularization locality-preserving projection method for face recognition. *International Journal of Wavelets, Multiresolution and Information Processing*, 2012, 10(6): 1250053.

[11] Fekete S P, Kröller A. Geometry-based reasoning for a large sensor network. In *Proc. the 22nd ACM Symposium on Computational Geometry*, June 2006, pp.475-476.

[12] Tao Y F, Papadias D, Lian X, Xiao X K. Multidimensional reverse kNN search. *The VLDB Journal*, 2007, 16(3): 293-316.

[13] Kang J M, Mokbel M F, Shekhar S, Xia T, Zhang D H. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *Proc. the 23rd International Conference on Data Engineering*, April 2007, pp.806-815.

[14] Yiu M L, Mamoulis N. Reverse nearest neighbors search in ad hoc subspaces. *IEEE Transactions on Knowledge and Data Engineering*, 2007, 19(3): 412-426.

[15] Korn F, Muthukrishnan S. Influence sets based on reverse nearest neighbor queries. In *Proc. the ACM SIGMOD International Conference on Management of Data*, May 2000, pp.201-212.

[16] Tao Y F, Papadias D, Lian X. Reverse kNN search in arbitrary dimensionality. In *Proc. the 30th International Conference on Very Large Data Bases*, August 2004, pp.744-755.

[17] Singh A, Ferhatosmanoglu H, Tosun A. High dimensional reverse nearest neighbor queries. In *Proc. the 12th International Conference on Information and Knowledge Management*, November 2003, pp.91-98.

[18] Korn F, Muthukrishnan S, Srivastava D. Reverse nearest neighbor aggregates over data streams. In *Proc. the 28th International Conference on Very Large Data Bases*, August 2002, pp.814-825.

[19] Yang C Y, Lin K I. An index structure for efficient reverse nearest neighbor queries. In *Proc. the 17th International Conference on Data Engineering*, April 2001, pp.485-492.

[20] Stanoi I, Riedewald M, Agrawal D, Abbadi A E. Discovery of influence sets in frequently updated databases. In *Proc. the 27th International Conference on Very Large Data Bases*, September 2001, pp.99-108.

[21] Cheema M A, Lin X M, Zhang W J, Zhang Y. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *Proc. the 27th International Conference on Data Engineering*, April 2011, pp.577-588.

[22] Achtert E, Böhm C, Kröger P, Kunath P, Pryakhin A, Renz M. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proc. ACM SIGMOD International Conference on Management of Data*, June 2006, pp.515-526.

[23] Yiu M L, Papadias D, Mamoulis N, Tao Y F. Reverse nearest neighbors in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2006, 18(4): 540-553.

[24] Xia T, Zhang D H. Continuous reverse nearest neighbor monitoring. In *Proc. the 22nd International Conference on Data Engineering*, April 2006, p.77.

[25] Benetis R, Jensen C S, Karciauskas G, Saltenis S. Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal*, 2006, 15(3): 229-249.

[26] Yao B, Li F F, Kumar P. Reverse furthest neighbors in spatial databases. In *Proc. the 25th International Conference on Data Engineering*, March 2009, pp.664-675.

[27] Tran Q T, Taniar D, Safar M. Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems* I, Hameurlain A, Küng J, Wagner R (eds.), Springer-Verlag, 2009, pp.353-372.

[28] Goldberg A V, Harrelson C. Computing the shortest path: A search meets graph theory. In *Proc. the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2005, pp.156-165.

[29] Hendrickson B, Leland R. A multilevel algorithm for partitioning graphs. In *Proc. the IEEE/ACM SC95 Conference on Supercomputing*, December 1995, p.28.

[30] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAMJournal on Scientific Computing*, 1998, 20(1): 359-392.

[31] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 1970, 49(2): 291-307.

[32] Pellegrini F, Roman J. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. International Conference on High-Performance Computing and Networking*, April 1996, pp.493-498.

[33] Jing N, Huang Y W, Rundensteiner E A. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 1998, 10(3): 409-432.

[34] Erwig M. The graph Voronoi diagram with applications. *Networks*, 2000, 36(3): 156-163.

[35] Aurenhammer F. Voronoi diagrams — A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 1991, 23(3): 345-405.

[36] Shekhar S, Liu D R. CCAM: A connectivity-clustered access method for networks and network computations. *IEEE Transactions on Knowledge and Data Engineering*, 1997, 9(1): 102-119.

[37] Shang S, Chen L S, Wei Z W, Jensen C S, Wen J R, Kalnis P. Collective travel planning in spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 2016, 28(5): 1132-1146.

[38] Shang S, Liu J J, Zheng K, Lu H, Pedersen T B, Wen J R. Planning unobstructed paths in traffic-aware spatial networks. *GeoInformatica*, 2015, 19(4): 723-746.

[39] Shang S, Ding R G, Zheng K, Jensen C S, Kalnis P, Zhou X F. Personalized trajectory matching in spatial networks. *The VLDB Journal*, 2014, 23(3): 449-468.

[40] Shang S, Ding R G, Yuan B, Xie K X, Zheng K, Kalnis P. User oriented trajectory search for trip recommendation. In *Proc. the 15th International Conference on Extending Database Technology*, March 2012, pp.156-167.

[41] Zhu Z X, Xiao J, Li J Q, Wang F X, Zhang Q F. Global path planning of wheeled robots using multi-objective memetic algorithms. *Integrated Computer-Aided Engineering*, 2015, 22(4): 387-404.

[42] Guo X N, Zhang D, Wu K S, Ni L M. MODLoc: Localizing multiple objects in dynamic indoor environment. *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(11): 2969-2980.

[43] Li R H, Yu J X, Huang X, Cheng H, Shang Z C. Measuring the impact of MVC attack in large complex networks. *Information Sciences*, 2014, 278: 685-702.

[44] Shi Y F, Long P X, Xu K, Huang H, Xiong Y S. Data-driven contextual modeling for 3D scene understanding. *Computers & Graphics*, 2016, 55: 55-67.

[45] Li B C, Li R H, King I, Lyu M R, Yu J X. A topic-biased user reputation model in rating systems. *Knowledge and Information Systems*, 2015, 44(3): 581-607.

[46] Li B, Tan S Q, Wang M, Huang J W. Investigation on cost assignment in spatial image steganography. *IEEE Transactions on Information Forensics and Security*, 2014, 9(8): 1264-1277.

[47] Li B, Wang M, Li X L, Tan S Q, Huang J W. A strategy of clustering modification directions in spatial image steganography. *IEEE Transactions on Information Forensics and Security*, 2015, 10(9): 1905-1917.

[48] Yang X, Pei J H, Sun W. Elastic image registration using hierarchical spatially based mean shift. *Computers in Biology and Medicine*, 2013, 43(9): 1086-1097.

[49] Zhou F, Jiao J X, Lei B Y. A linear threshold-hurdle model for product adoption prediction incorporating social network effects. *Information Sciences*, 2015, 307: 95-109.

[50] Wang J G, Huang J Z, Guo J F, Lan Y Y. Recommending high-utility search engine queries via a query-recommending model. *Neurocomputing*, 2015, 167: 195-208.

[51] Lin J C, Gan W S, Fournier-Viger P, Hong T P, Tseng V S. Efficient algorithms for mining high-utility itemsets in uncertain databases. *Knowledge-Based Systems*, 2016, 96: 171-187.

[52] Du S Y, Guo Y R, Sanroma G, Ni D, Wu G R, Shen D G. Building dynamic population graph for accurate correspondence detection. *Medical Image Analysis*, 2015, 26(1): 256-267.

[53] Luo X, Ming Z, You Z H, Li S, Xia Y N, Leung H. Improving network topology-based protein interactome mapping via collaborative filtering. *Knowledge-Based Systems*, 2015, 90: 23-32.

[54] Li R H, Yu J X. Triangle minimization in large networks. *Knowledge and Information Systems*, 2015, 45(3): 617-643.

[55] Dai M J, Sung C W. Achieving high diversity and multiplexing gains in the asynchronous parallel relay network. *Transactions on Emerging Telecommunications Technologies*, 2013, 24(2): 232-243.

[56] Zhang D, Lu K Z, Mao R. A precise RFID indoor localization system with sensor network assistance. *China Communications*, 2015, 12(4): 13-22.

[57] Huang X, Cheng H, Li R H, Qin L, Yu J X. Top-K structural diversity search in large networks. *The VLDB Journal*, 2015, 24(3): 319-343.

[58] Wu R B, Li C, Lu D. Power minimization with derivative constraints for high dynamic GPS interference suppression. *Science China-Information Sciences*, 2012, 55(4): 857-866.

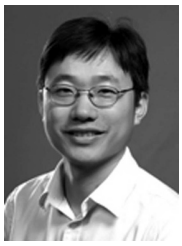
[59] Zhao Q L, Liew S C, Zhang S L, Yu Y. Distance-based location management utilizing initial position for mobile communication networks. *IEEE Transactions on Mobile Computing*, 2016, 15(1): 107-120.

[60] Wang J Y, Feng J W, Xu C, Zhao Y, Feng J Q. Pinning synchronization of nonlinearly coupled complex networks with time-varying delays using M-matrix strategies. *Neurocomputing*, 2016, 177: 89-97.

Xiao-Jun Xu is currently a Ph.D. candidate in the School of Software, Beijing Institute of Technology, Beijing, and the associate director of the Information Technology Security Evaluation Center of the Ministry of Public Security, Beijing. His major interests include network and information security, cloud computing and big data mining.

Jin-Song Bao received his Ph.D. degree in mechanical engineering from Shanghai Jiao Tong University (SJTU), Shanghai, in 2002. He is a professor at the College of Mechanical Engineering, Donghua University, Shanghai, and the director of Computer Integrated Manufacturing Institute Lab of SJTU and CAD/CAE/Collaborative Simulation/VR Lab of SJTU (C3VR Lab), Shanghai. His current research focuses on intelligent manufacturing and dominant innovation tools for shipbuilding, aerospace product in whole lifecycle, and smart predictive technologies using big data.

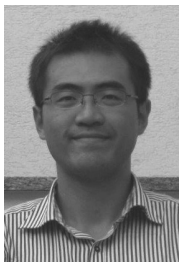
Bin Yao received his B.S. degree and M.S. degree in computer science from the South China University of Technology, Guangzhou, in 2003 and 2007 respectively, and his Ph.D. degree in computer science from the Florida State University, Florida, in 2011. He has been an associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, since 2014. His research interests are management and indexing of large databases, and scalable data analytics.



Jing-Yu Zhou received his B.S. degree in computer science from Zhejiang University, Hangzhou, in 1999. He received his M.S. and Ph.D. degrees in computer science from University of California, Santa Barbara, in 2003 and 2006 respectively. He is currently an associate professor at Shanghai Jiao Tong University, Shanghai. He is generally interested in distributed systems, information retrieval, and security. He has published more than 30 papers at various conferences and journals, including WWW, INFOCOM, ICS, TPDS, DSN, FSE, CIKM, and IPDPS. He has served as PC member for more than 20 international conferences.

Fei-Long Tang received his Ph.D. degree in computer science from Shanghai Jiao Tong University (SJTU), Shanghai, in 2005. Now, he is a full professor in the Department of Computer Science and Engineering at SJTU, Shanghai. In past years, he was the JSPS (Japan Society for the Promotion of Science) Research Fellow in Japan, and received the Distinguished Pu-Jiang Scholars Award from Shanghai Municipality. His research interests focus on mobile cognitive network, big data analysis and clouding computing. He has received two Best Paper Awards from international conferences. He served as the program co-chair for eight international conferences and is a member of CCF, ACM, IEEE.

Min-Yi Guo received his Ph.D. degree in computer science from the University of Tsukuba, Tsukuba, in 1998. He is currently a chair professor and the head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. His research interests include, but are not limited to parallel and distributed computing, compiler optimizations. He is on the editorial board of journals such as IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers. He is a senior member of IEEE.



Jian-Qiu Xu got his Ph.D. degree in computer science from FernUniversität in Hagen, Hagen, in 2012. He is now an associate professor in Nanjing University of Aeronautics and Astronautics, Nanjing, and a member of ACM SIGSPATIAL. He focuses on moving objects with multiple transportation modes. His research interests include moving objects databases and spatial databases. He has published papers in Geoinformatica and IEEE MDM.