

Ranked Continuous Visible Nearest Neighbor Search

¹Yan Chen*, ²Yunjun Gao*

*1, First and Corresponding Author Zhejiang Institute of Communications, chenyan@zjvit.edu.cn
*2, Corresponding Author College of Computer Science, Zhejiang University, gaoyj@zju.edu.cn
doi: 10.4156/jcit.vol5.issue8.16

Abstract

Physical obstacles (e.g., buildings, hills, and blindages, etc.) are ubiquitous in the real world, and their existence may affect the visibility between objects and thus the result of spatial queries such as range query, nearest neighbor search, and spatial join, etc. In this paper, we study a novel type of spatial queries, namely, ranked continuous visible nearest neighbor (RCVNN) search, which returns the k visible nearest neighbors that have the maximal optimality according to the predefined optimality metric. We first formulate the problem, and then present an efficient algorithm for RCVNN query processing and prove its correctness. Extensive experimental evaluation demonstrates the efficiency and effectiveness of our proposed algorithm using both real and synthetic datasets.

Keywords: Query Processing, Nearest Neighbor, Visible

1. Introduction

A large amount of physical obstacles (e.g., buildings, hills, and blindages, etc.) exist in the real world, and their existence may affect the *visibility* between objects and hence the result of spatial queries such as range query, nearest neighbor (NN) search, spatial join, and so forth. Furthermore, in some applications, users might be only interested in the objects that are visible or reachable to them.

Based on these observations, the impact of obstacles has recently been investigated in various spatial queries. They mainly include (i) *visible k nearest neighbor* (VkNN) retrieval [1, 2], which returns the k (≥ 1) closest objects that are visible to a given query point; (ii) *visible reverse k -nearest neighbor* (VRkNN) search [3, 4], which retrieves the points in a data set P that have a specified query point as one of their k visible nearest neighbors (VNNs), considering the blocks of obstacles in an obstacle set O ; (iii) *continuous visible nearest neighbor* (CVNN) query [5], which finds the visible nearest neighbor (VNN) of every point along a given query line segment; (iv) *obstructed nearest neighbor* (ONN) search [6, 7], which returns the k points from a dataset that have the smallest *obstructed distances*¹ to a predefined query point; and (v) *continuous obstructed nearest neighbor* (CONN) retrieval [8], which retrieves the ONN for every point along a specified query line segment according to the obstructed distance; (vi) *moving k nearest neighbor* (MkNN) query over obstructed regions [9], which retrieves the top k nearest neighbors while the query object moves, considering the impact of obstacles; and (vii) *spatial clustering in the presence of obstacles* [10, 11, 12, 13, 14, 15], which divides a set of two-dimensional data points into homogeneous groups (i.e., clusters) by taking the influence of obstacles into consideration. In addition, with the growing popularity of smart mobile devices and rapid advance of wireless technologies, more and more users issue queries even when they are moving. Consequently, the traditional *snapshot query* might not satisfy the real requirements from mobile users, and *continuous query* processing over a moving trajectory is required.

More recently, CVNN search has been well-studied in [5] due to its importance in a large number of applications such as decision support and location-based commerce. One example application is listed as follows.

Outdoor advertisement. Assume that an advertisement company plans to place outdoor advertisement posters at some popular locations in a downtown area. By performing a CVNN query which takes as input a set P of potential locations placing poster billboards, a set O of

¹ The obstructed distance between any two points in a data set is defined as the length of the shortest path that connects them without crossing any obstacle from a set of obstacles.

obstacles (e.g., buildings), and an anticipated segment/trajectory q (e.g., a commercial street), the decision-maker of the company can discover a set of $\langle loc, R \rangle$ tuples, such that location $loc \in P$ is the VNN of all customers shopping in the corresponding region R (i.e., an interval of q). Therefore, in this example, the CVNN retrieval provides a near optimal selection of the poster billboard locations in order to guarantee the visibility of the poster billboards.

Unfortunately, for a CVNN query, a user cannot control the number of *answer objects*² returned. Moreover, CVNN search treats every answer object equally. However, in some cases, the user may want to rank the answer objects returned by CVNN retrieval and/or want to specify the number of answer objects he/she wants to retrieve. Motivated by this, in this paper, we study a novel type of spatial queries, namely *ranked continuous visible nearest neighbor* (RCVNN) search, which returns the k VNNs that have the *maximal optimality* according to the predefined *optimality metric*. Specifically, we first formalize the problem, and then develop an efficient algorithm for RCVNN query processing and prove its correctness. Finally, extensive experiments with both real and synthetic datasets confirm the performance of our proposed algorithm in terms of efficiency and effectiveness.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 presents the problem statement. Section 4 elaborates an algorithm for answering RCVNN retrieval, and analyzes its time complexity and correctness. Extensive experimental evaluations and our findings are reported in Section 5. Finally, Section 6 concludes the paper with some directions for future work.

2. Related work

In this section, we overview the existing work related to RCVNN search. The existence of obstacles could affect the visibility, distance, and spatial object clustering.

First, in terms of visibility, two objects are visible to each other iff the straight line segment connecting them does not pass through any obstacle. Nutanong et al. [1, 2] introduce *visible nearest neighbor* (VNN) search to find the NN object that is visible to a given query point. A VNN query algorithm, based on the fact that a faraway object cannot affect the visibility of a nearby object, is proposed in [1, 2]. The basic idea is to perform NN search and check visibility condition in an incremental manner. However, the algorithm can only support VNN retrieval for a fixed point but not a line segment. Later, Gao et al. [3, 4] explore *visible reverse nearest neighbor* (VRNN) search where, given a data set P , an obstacle set O , and a query point p , the goal is to retrieve all the points in P that have p as their VNN. An algorithm for VRNN query processing, assuming that both P and O are indexed by R-trees [16], is presented in [3, 4]. The algorithm follows a filter-refinement framework, and requires zero pre-processing. Specifically, it identifies a candidate set during the filter step, and filters out false hits in the refinement step, with these two steps integrated into a single R-tree traversal. Furthermore, pruning techniques based on half-plane properties and visibility check are developed to further improve the search performance. Along this line, Gao et al. [5] also study *continuous visible nearest neighbor* (CVNN) search where, given a data set P , an obstacle set O , and a query line segment q , the goal is to return the VNN of every point on q . In [5], a CVNN search algorithm, assuming that both P and O are indexed by R-trees, is proposed. The basic idea is to traverse data points in P based on ascending order of their *mindist* to q . For each data point $p \in P$ visited, the algorithm evaluates p 's impact on the current query result. Although the above works on spatial visible queries consider the existence of obstacles, they employ the Euclidean distance but not the obstructed distance to measure the proximity between objects.

Second, in terms of distance, Zhang et al. [6] and Xia et al. [7] investigate the *obstructed nearest neighbor* (ONN) query where, given a data set P , an obstacle set O , and a query point p , the goal is to find the k (≥ 1) objects in P that have the smallest obstructed distances from p . Existing ONN search algorithm utilizes conventional NN retrieval to retrieve objects close to the query point as candidates, and terminates when the retrieved NN object has its Euclidean distance to the query point larger than the maximum obstructed distance of the candidates.

² For the rest of this paper, we refer to the data objects/points in the final query result as answer objects/points.

Apart from ONN search, Zhang et al. [6] also propose algorithms for processing other popular spatial queries, including range search, e -distance joins, and closest pairs, in the presence of obstacles. Recently, Gao et al. [8] explore *continuous obstructed nearest neighbor* (CONN) search where, given a data set P , an obstacle set O , and a query line segment q , the target is to return the ONN of every point on q . In [8], Gao et al. propose an efficient algorithm for CONN retrieval, assuming that the data set P and the obstacle set O are indexed by R-trees. The basic idea is to traverse data points in P in ascending order of their *mindist* (that is the lower bound of obstructed distances) to a given query line segment $q = [S, E]$. For each data point $p \in P$ visited, the algorithm first finds all the obstacles that may affect the obstructed distances from p to any point along q , then identifies the control point(s) of p over q , and finally evaluates the impact of p on the current result list RL . Li et al. [9] discuss the problem of *obstructed moving k nearest neighbor* (MkNN) search, which retrieves the top k nearest neighbors while the query object moves by taking into account obstacles. An algorithm for obstructed MkNN query processing is proposed in [9] using the concept of obstacle-free safe region.

Third, the *spatial clustering in the presence of obstacles* has attracted considerable attention in recent years. It divides a set of two-dimensional data points into smaller homogeneous groups (i.e., clusters) by considering the impact of obstacles. Handling these constraints can lead to effective and fruitful data mining by capturing application semantics [8]. A large number of clustering algorithms with obstacle constraints have been developed in the literature, including AUTOCLUST+ [10], COD_CLARANS [11], DBCLuC [12], DBRS+ [13], DBRS_O [14], and DBSCAN_MDO [15], etc.

3. Problem statement

In this section, we formally define the optimality metric and the RCVNN search. Take the *outdoor advertisement* described in Section 1 as an example. A CVNN query may return a result set which includes five poster billboard positions. Nevertheless, the company's budget can only afford two poster billboards. How to select two positions out of those five poster billboard positions? Intuitively, an ideal poster billboard location is expected to be visible to as many customers as possible, i.e., its visible region is preferred to be as long as possible. Consequently, given two answer objects o_1 and o_2 , and their corresponding visible regions on a specified query line segment are VR_{o_1} and VR_{o_2} , respectively. Object o_1 is *more optimal* than object o_2 if $\|VR_{o_1}\| > \|VR_{o_2}\|$ ³. In other words, the length of the visible region of a CVNN answer object contributes to its *optimality*. The *longer* the visible region is, the *more optimal* the answer object is. Consider, for instance, Figure 1, in which data set $P = \{a, b, c\}$, obstacle set $O = \{o_1, o_2\}$, and query line segment $q = [s, e]$. As depicted in Figure 1, the result of a CV2NN query is $\{\langle\{a, b\}, [s, s_1]\rangle, \langle\{b, c\}, [s_1, e]\rangle\}$, meaning that points a, b are 2 VNNs for any point along interval $[s, s_1]$, and points b, c are 2 VNNs for any point along interval $[s_1, e]$. Thus, the optimality of b is better than that of a because $\|VR_b\| (= \|[s, s_1]\| + \|[s_1, e]\|) > \|VR_a\| (= \|[s, s_1]\|)$.

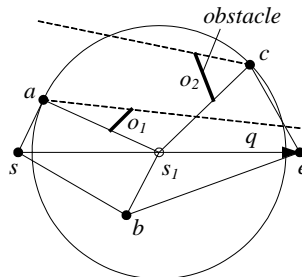


Figure 1. Illustration of optimality metric for RCVNN search

³ Note that the notation $\|r\|$ represents the length of an interval/region r in this paper.

However, when two candidate poster billboard locations o_1 and o_2 have the same visible region size (i.e., $\|VR_{o_1}\| = \|VR_{o_2}\|$), the tie has to be broken. Based on the assumption that customers prefer poster billboards that are close to them, we therefore employ the *average distance* from the candidate poster billboard location o to all the regions contained in its visible region VR_o over a given query line segment q , denoted by d_o , as the tie breaker. The *shorter* the average distance is, the *better* the object optimality is. In this paper, we consider the *mindist*⁴ metric, and define d_o as the average minimal distance between an object o and all the intervals/regions included in its corresponding visible region VR_o on q , i.e., $d_o = \frac{1}{t} \sum_{i=1}^t \text{mindist}(o, R_i)$ with $VR_o = \bigcup_{i=1}^t R_i$. As shown in Figure 1, although $\|VR_a\| (= \|[s, s_1]\|) = \|VR_c\| (= \|[s_1, e]\|)$, a is more optimal than c as $d_a < d_c$.

Based on the above analysis, the formal definition of the *optimality metric* is stated in Definition 1.

Definition 1 (Optimality metric). Given any two candidate visible nearest neighbors p and p' , p is *more optimal* than p' , denoted as $p < p'$, iff $\|VR_p\| > \|VR_{p'}\|$ or $\|VR_p\| = \|VR_{p'}\| \wedge d_p < d_{p'}$, where $\|VR_x\|$ is the length of x 's visible region and $d_x = \frac{1}{t} \sum_{i=1}^t \text{mindist}(x, R_i)$ with $VR_x = \bigcup_{i=1}^t R_i$.

Ranked continuous visible nearest neighbor (RCVNN) search is to retrieve the answer objects with the best optimality, as presented in Definition 2 based on Definition 1.

Definition 2 (Ranked continuous visible nearest neighbor query). Given a data set P , an obstacle set O , a query line segment q , and an integer k , we assume $RL = \bigcup_{i=1}^t \langle p_i, R_i \rangle$ is the result list returned by a CVNN query issued at q . A *ranked continuous visible nearest neighbor* (RCVNN) query returns a set of objects, denoted as $S = \bigcup_{i=1}^s \langle s_i, R_i \rangle$, such that (i) $S \subseteq RL$; (ii) $\forall s_i \in S, \nexists p \in RL - S$ that $p < s_i$; and (iii) $|S| \leq k$ (i.e., $s \leq k$).

RCVNN retrieval ranks the answer objects of a conventional CVNN query according to the optimality metric, and returns the top- k objects with the highest optimality. It is worth noting that the number of answer objects returned by a RCVNN query might be *smaller* than k when the initial result set of a CVNN query has its cardinality smaller than k .

4. RCVNN query processing

In this section, we first present the RCVNN query processing algorithm, namely *RCVNN Search Algorithm* (RCVNN), and then analyze the time complexity and correctness of RCVNN. The pseudo-code of RCVNN is shown in Algorithm 1. RCVNN takes as input an R-tree T_p on data set P , an R-tree T_o on obstacle set O , and a query line segment $q = [s, e]$, and outputs the final result set S of a RCVNN query.

RCVNN follows the best-first traversal paradigm. In order to enable the best-first traversal, it maintains two heaps H_p and H_o storing the data and obstacle entries visited so far respectively, sorted by ascending order of their minimal distances (i.e., *mindist*) to q . Initially, RCVNN initializes H_p and H_o via inserting all the child entries from T_p and T_o , respectively (lines 2-5). Then, the algorithm retrieves the visible nearest neighbor for every point along q by invoking the CVNN algorithm proposed in [5] (line 6). In particular, the basic idea of CVNN is to traverse data points in P based on the ascending order of their *mindist* to q . For each data point $p \in P$ visited, the algorithm needs to check whether p will update the current query result list RL which is initialized to $\langle \emptyset, [s, e] \rangle$. That is to evaluate whether p will violate the dominance of an existing answer object p_i on an interval/region R_i (either partially or completely), with $\langle p_i, R_i \rangle \in RL$. Once CVNN algorithm terminates, RCVNN sorts all answer points (i.e., VNN objects) in RL returned by CVNN using their optimality (line 7). Finally, the best k answer objects as the final result of the RCVNN query are returned (line 8).

⁴ The metric *mindist*(N, q) denotes the minimal Euclidean distance between a node (or a data point) N and a given query point q .

Algorithm 1 RCVNN Search Algorithm (RCVNN)

Input: a data R-tree T_p , an obstacle R-tree T_o , a query line segment $q = [s, e]$
Output: the result set S of a RCVNN query
 /* $T_p.root$ denotes the root node of T_p ; $T_o.root$ represents the root node of T_o ; RL specifies the result list of a CVNN query */
 1: $S = \{\langle \emptyset, \emptyset \rangle\}$, and $RL = \{\langle \emptyset, [s, e] \rangle\}$
 2: **for** each entry $e \in T_p.root$ **do**
 3: en-heap e into H_p
 4: **for** each entry $e \in T_o.root$ **do**
 5: en-heap e into H_o
 6: perform CVNN (T_p, T_o, q) with heaps H_p, H_o , and store its result in RL // CVNN alg. proposed in [5]
 7: sort all answer objects in RL using optimality metric defined in Definition 1, and keep its result in S
 8: return S

Next, we analyze the time complexity and correctness of RCVNN, respectively. Let $|T_p|$ and $|T_o|$ be the tree size of T_p and T_o respectively, $|L_o|$ be the maximal number of obstacles in a linked list L_o during the CVNN search, $|RL|$ be the maximal number of entries in a result list RL , and N be the number of data points accessed during the CVNN search. The time complexity of the RCVNN algorithm is presented in Theorem 1, while the correctness of the RCVNN algorithm is proved in Theorem 2.

Theorem 1. The time complexity of the RCVNN algorithm is $O((N \times \log |T_p| \times (\log |T_o| + |L_o| + |RL|)) + |RL| \log |RL|)$.

Proof. A RCVNN query mainly consists of two steps: (i) it performs CVNN algorithm which takes $O(N \times \log |T_p| \times (\log |T_o| + |L_o| + |RL|))$ (proved in [5]) to find the visible nearest neighbor for every point along a specified query line segment; and (ii) it sorts all the answer objects in RL , which incurs $O(|RL| \log |RL|)$. Consequently, the overall time complexity of the RCVNN algorithm is $O((N \times \log |T_p| \times (\log |T_o| + |L_o| + |RL|)) + |RL| \log |RL|)$.

□ **Theorem 2.** The RCVNN algorithm returns exactly the answer objects with the highest optimality, i.e., the algorithm has no false misses and no false hits.

Proof. The correctness of the RCVNN algorithm is obvious since the CVNN algorithm employed by RCVNN can retrieve exactly the VNN of every point along a given query line segment, as proved in [5], which guarantees no false misses and no false hits.

□

5. Experimental evaluation

This section experimentally evaluates the performance of our proposed algorithm in terms of both efficiency and effectiveness. In the following, we first describe the experimental settings, and then report experimental results and our findings.

5.1. Experimental setup

Our experiments are based on both real and synthetic datasets, with the search space fixed at a $[0, 10000] \times [0, 10000]$ square. Two real datasets are deployed, namely *CA* and *LA*. Specifically, *CA* contains two-dimensional (2D) points, representing 62,556 locations in California; *LA* includes 2D rectangles, representing 131,461 MBRs of streets in Los Angeles. All the datasets are normalized in order to fit the search range. Synthetic datasets are generated based on uniform distribution and zipf distribution respectively, with the cardinality varying from $0 \times |LA|$ to $10 \times |LA|$. The coordinate of each point in *Uniform* datasets is generated uniformly along each dimension, and that of each point in *Zipf* datasets is generated according to zipf distribution with skew coefficient $\alpha = 0.8$. We assume a point's coordinates on both dimensions are mutually independent. Since RCVNN search involves a data set P and an obstacle set O , we deploy three different combinations of the datasets, namely *CL*, *UL*, and *ZL*, representing $(P, O) = (CA, LA)$, (*Uniform*, *LA*), and (*Zipf*, *LA*), respectively. Note that the data points in P are allowed to lie on the boundaries of the obstacles, but not in their interior.

Table 1. Parameter ranges and default values

Parameter	Range	Default
query length ql (% of space side)	5, 10, 15, 20, 25	15
k	1, 3, 5, 7, 9	5
$ P / O $	0.1, 0.2, 0.5, 1, 2, 5, 10	1

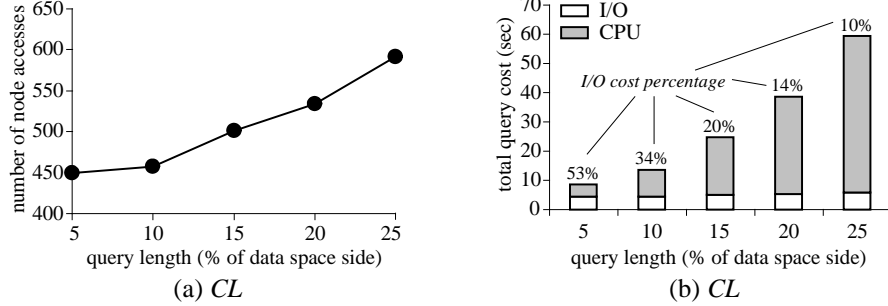


Figure 2. Performance vs. ql ($k = 5$)

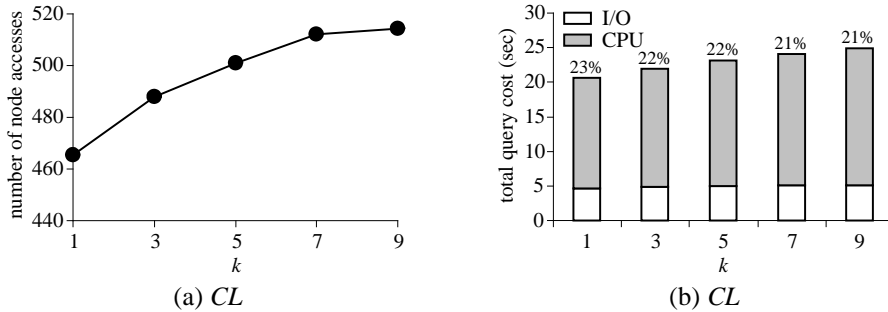


Figure 3. Performance vs. k ($ql = 15\%$)

All data and obstacle sets are indexed by R*-trees [16], with a page size of 4,096 bytes. We employ an LRU memory buffer whose default size is set to 10% of the tree size. Table 1 lists all the parameters and their default settings that are considered in the experiments. In each experiment, only one parameter is changed in order to evaluate its impact on the performance, while all the other parameters are fixed at their default values. We run 200 queries for each experiment, and the average performance is reported.

We employ the number of node/page accesses (i.e., I/O cost), CPU time, and total query cost (i.e., the summation of the I/O time and CPU time, where the I/O time is computed by charging 10ms for each page access) as the major performance metrics in the experiments. Given a query length ql , a query line segment is randomly generated for each query. In particular, each query line segment is generated by (i) selecting uniformly a point in the data space as the starting point of the query line segment, and (ii) selecting randomly an orientation (angle with the x -axis) from the range $[0, 2\pi)$, with its length controlled by the specified query length ql .

5. 2. Performance study

First, we investigate the effect of query length ql on the efficiency of the RCVNN algorithm based on the real dataset combination CL with k set to 5. The results are depicted in Figure 2. It is observed that the cost of RCVNN queries increases with ql . The reason behind is that, as the query length becomes longer, both the number of data points processed and the number of the splitting regions in the specified query line segment ascend, resulting in more distance computation, visibility examination, and result list updating.

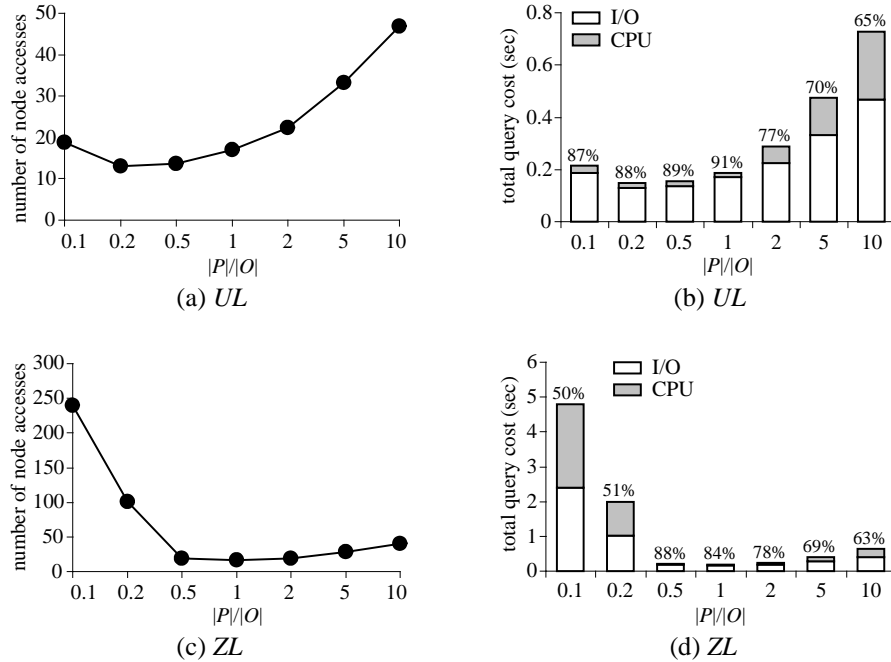


Figure 4. Performance vs. $|P|/|O|$ ($ql = 15\%$, $k = 5$)

To explore the influence of k , we employ *CL* again, fix ql to 15%, and vary k between 1 and 9. Figure 3 illustrates the performance of the RCVNN algorithm under different k values. As expected, the cost of RCVNN retrieval increases with the growth of k , because a higher k value incurs a larger search space, more distance computation, and more result list maintenance cost.

Finally, the third experiment studies the impact of $|P|/|O|$. Towards this, we fix ql and k to 15% and 5 respectively, change $|P|/|O|$ in the range $[0.1, 10]$, and deploy synthetic dataset combinations *UL* and *ZL*. Figure 4 plots the efficiency of the RCVNN algorithm with respect to $|P|/|O|$. Observe that, as $|P|/|O|$ increases from 0.1 to 10, the cost of RCVNN queries first drops and then increases. This is because initially, the density of dataset P ascends with the growth of $|P|/|O|$, which implies a smaller search range for the answer points and obstacles. Thus, the performance improves. However, as $|P|/|O|$ further grows, the interval dominated by each data point becomes shorter, and the result list contains more answer points. The gain from the reduced search range cannot pay off the cost of frequent result list update operation, and hence the performance deteriorates. Notice that the performance of RCVNN search achieves the best performance when P and O share similar cardinalities (e.g., $|P|/|O| = 0.5$ or 1 in Figure 4).

6. Conclusions

This paper introduces and solves a novel type of spatial queries, namely *ranked continuous visible nearest neighbor* (RCVNN) search. RCVNN retrieval is not only interesting from a research point of view, but also useful in many decision support applications involving spatial data and obstacles. First, we formalize the optimality metric and the RCVNN query, respectively. Then, an algorithm, i.e., RCVNN, is developed for efficiently processing RCVNN search. Finally, extensive experimental evaluation verifies the performance of the proposed RCVNN algorithm.

In the future, an interesting direction is to investigate visibility queries for moving objects and moving obstacles. In addition, a topic worth studying concerns visibility queries in metric spaces, where the proximity of data/obstacle objects is evaluated by a given distance function.

7. Acknowledges

Yunjun Gao was supported in part by the NSFC titled “*Research on Reverse Nearest Neighbor Query Processing Technologies in the Presence of Obstacles*”, the ZJNSF under Grant no. Y1100278, and the Fundamental Research Funds for the Central Universities titled “*Research on Several Problems of Spatial Database Query Processing Technologies*”. We would also like to thank the anonymous reviewers for their valuable comments.

8. References

- [1] S. Nutanong, E. Tanin, R. Zhang, “Visible Nearest Neighbor Queries”, In Proceedings of the 12th International Conference on Database Systems for Advanced Applications, pp. 876-883, 2007.
- [2] S. Nutanong, E. Tanin, R. Zhang, “Incremental Evaluation of Visible Nearest Neighbor Queries”, IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 5, pp. 665-681, 2010.
- [3] Y. Gao, B. Zheng, G. Chen, W.-C. Lee, K. C. K. Lee, Q. Li, “Visible Reverse k -Nearest Neighbor Queries”, In Proceedings of the 25th International Conference on Data Engineering, pp. 1203-1206, 2009.
- [4] Y. Gao, B. Zheng, G. Chen, W.-C. Lee, K. C. K. Lee, Q. Li, “Visible Reverse k -Nearest Neighbor Query Processing in Spatial Databases”, IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, pp. 1314-1327, 2009.
- [5] Y. Gao, B. Zheng, W.-C. Lee, G. Chen, “Continuous Visible Nearest Neighbor Queries”, In Proceedings of the 12th International Conference on Extending Database Technology, pp. 144-155, 2009.
- [6] J. Zhang, D. Papadias, K. Mouratidis, M. Zhu, “Spatial Queries in the Presence of Obstacles”, In Proceedings of the 9th International Conference on Extending Database Technology, pp. 366-384, 2004.
- [7] C. Xia, D. Hsu, A. K. H. Tung, “A Fast Filter for Obstructed Nearest Neighbor Queries”, In Proceedings of the 21st British National Conference on Databases, pp. 203-215, 2004.
- [8] Y. Gao, B. Zheng, “Continuous Obstructed Nearest Neighbor Queries in Spatial Databases”, In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 577-590, 2009.
- [9] C. Li, Y. Gu, F. Li, M. Chen, “Moving k -Nearest Neighbor Query over Obstructed Regions”, In Proceedings of the 12th Asia-Pacific Web Conference on Advances in Web Technologies and Applications, pp. 29-35, 2010.
- [10] V. Estivill-Castro, I. Lee, “Autoclust+: Automatic Clustering of Point-Data Sets in the Presence of Obstacles”, In Proceedings of the First International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining, pp. 133-146, 2000.
- [11] A. K. H. Tung, J. Hou, J. Han, “Spatial Clustering in the Presence of Obstacles”, In Proceedings of the 17th International Conference on Data Engineering, pp. 359-367, 2001.
- [12] O. R. Zaiane, C. H. Lee, “Clustering Spatial Data in the Presence of Obstacles: A Density-Based Approach”, In Proceedings of the 6th International Database Engineering & Applications Symposium, pp. 214-223, 2002.
- [13] X. Wang, C. Rostoker, H. J. Hamilton, “Density-Based Spatial Clustering in the Presence of Obstacles and Facilitators”, In Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 446-458, 2004.
- [14] X. Wang, H. J. Hamilton, “Clustering Spatial Data in the Presence of Obstacles”, International Journal on Artificial Intelligence Tools, vol. 14, no. 1-2, pp. 177-198, 2005.
- [15] S. H. Park, J. H. Lee, D. H. Kim, “Spatial Clustering Based on Moving Distance in the Presence of Obstacles”, In Proceedings of the 12th International Conference on Database Systems for Advanced Applications, pp. 1024-1027, 2007.
- [16] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger, “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles”, In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 322-331, 1990.