Optimal-Nearest-Neighbor Queries

Yunjun Gao[#1], Jing Zhang[#1], Gencai Chen[#1], Qing Li[*2], Shen Liu[#1], Chun Chen[#1]

[#] *College of Computer Science, Zhejiang University*
Hangzhou 310027, P. R. China
[1] {gaoyj, zhangj, chengc, lius, chenc}@zju.edu.cn

[*] *Department of Computer Science, City University of Hong Kong*
Tat Chee Avenue, Kowloon, Hong Kong, P. R. China
[2] itqli@cityu.edu.hk

***Abstract*— Given two sets D_A and D_B of multidimensional objects, a spatial region *R*, and a critical distance d_c, an optimal-nearest-neighbor (ONN) query retrieves outside *R*, the object in D_B with maximum optimality. Let *CAR* (S_p, *p*) be the cardinality of the subset S_p of objects in D_A which locate within *R* and are enclosed by the vicinity circle centered at *p* with radius d_c. Then, an object *o* is said to be *better* than another one *o'* if (i) *CAR* (S_o, *o*) > *CAR* ($S_{o'}$, *o'*), or (ii) when *CAR* (S_o, *o*) = *CAR* ($S_{o'}$, *o'*) the sum of the weighted distance from each object in S_o to *o* is smaller than the sum of the weighted distance between every object in $S_{o'}$ and *o'*. This type of queries is quite useful in many decision making applications. In this paper, we formalize the ONN query, develop the optimality metric, and propose several algorithms for finding optimal nearest neighbors efficiently. Our techniques assume that both D_A and D_B are indexed by R-trees. Extensive experiments demonstrate the efficiency and scalability of our proposed algorithms using both real and synthetic datasets.**

I. Introduction

Nearest neighbor (NN) search is one of the most common query types in spatial databases. A popular generalization is the *k*-NN retrieval. Given a multidimensional data set *D* and a query point *q*, a *k*-NN query returns the set $P \subseteq D$ such that $|P| = k$ and for any $p \in P$ and $p' \in D - P$, *dist* (*p*, *q*) ≤ *dist* (*p'*, *q*) where *dist* is a distance[1] metric. Over the last decade, NN query processing has already received considerable attention in the database community due to its importance in a wide range of applications such as decision making, location based services, resource management, etc. Traditional NN queries require input as points from which the nearest neighbors (NNs) are computed. Thus, they are also called *point* NN (PNN) queries. The existing algorithms [3, 5] for PNN queries assume that *D* is indexed by a spatial access method (e.g., R*-tree [1], etc.) and utilize some pruning bounds to restrict the search space. In addition to PNN queries, other versions of the NN retrieval, e.g., *reverse* NN search [4], *constrained* NN search [2], etc. have been explored as well.

In this paper, we study a new form of NN queries, called optimal-nearest-neighbor (ONN) search. Given two multidimensional object sets D_A and D_B, a spatial region *R*, and a critical distance d_c, an ONN query retrieves outside *R*, the object in D_B with maximum optimality. Let *CAR* (S_p, *p*) be the cardinality of the subset S_p of objects in D_A which locate within *R* and are enclosed by the vicinity circle centered at *p* with radius d_c. Then, an object *o* is said to be *better* than another object *o'* if (i) *CAR* (S_o, *o*) > *CAR* ($S_{o'}$, *o'*), or (ii) when *CAR* (S_o, *o*) = *CAR* ($S_{o'}$, *o'*) the sum of the weighted distance from each object in S_o to *o* is smaller than that of the weighted distance between every object in $S_{o'}$ and *o'*. Moreover, a generalized case is a *k*-ONN query, which returns the *k* optimal nearest neighbors (ONNs), given a positive integer *k*.

ONN queries are quite useful in many applications which use spatial data for decision making. As a motivating example, consider a case where one data set (D_A) stands for some important facilities (e.g., hospitals, schools, markets, parks, etc.), while the second data set (D_B) represents the available flats for sale. An ONN query can discover the flat with *maximum optimality* outside a user preference region *R* (e.g., shopping centre, scenic area, etc.) that maximally benefits the customers. Such information could be utilized by a real estate agency for service recommendation purpose. The value of d_c used in this example is dependent on the tolerable distance of the customer who wants to look for a suitable flat.

Despite the usefulness of the ONN query, to our knowledge, it has not been investigated in the past. Our study to be presented in this paper addresses this new type of spatial queries. Specifically, we formalize the ONN query, develop the optimality metric that serves as the standard for optimality evaluation, and propose several algorithms for processing ONN queries efficiently. Our approaches use the R-tree as the underlying index structure due to its efficiency and popularity in the research and industry community. Finally, an extensive experimental study with both real and synthetic datasets demonstrates the performance properties of our proposed algorithms in terms of efficiency and scalability.

II. Problem Statement

Suppose that *S* (*S'*) is the set of points in D_A within *R* and these points are covered by the circle centered at point b_j (b_j') in D_B with radius d_c. Intuitively, the larger the cardinality $|S|$ of *S* is, the better the optimality evaluation of b_j is. Furthermore, when the $|S|$ is identical to the cardinality $|S'|$ of *S'*, b_j is better than b_j' if the sum of the weighted distance from each point in *S* to b_j is smaller than that of the weighted distance between every point in *S'* to b_j'; otherwise, b_j' is better than b_j. In view of these observations, the optimality metric of b_j, denoted as OPT_{bj}, is defined as:

This work is supported in part by grant 60703040 from NSFC of china

[1]In this paper, we assume Euclidean distance, although any distance metric can be used in general.

$$OPT_{bj} = |R| - |S| + \sum_{s_t \in S, 1 \le t \le |S|} \frac{Dist(s_t, b_j)}{|S| \times d_c + 1} \times \frac{1}{t} \quad (1)$$

where $|R|$ is the number of points in D_A that are enclosed by R and s_t ($1 \le t \le |S|$) in S are sorted in ascending order of their distances from b_j. The motivation for this optimality definition is to employ a single value to measure a data object's optimality. Thus, we only need to find S and calculate $Dist$ (s_t, b_j) for every $s_t \in S$ in order to compute OPT_{bj}.

Definition 1 ($b_j \succ b_j'$). *b_j is superior to b_j', denoted as $b_j \succ b_j'$, if and only if $OPT_{bj} < OPT_{bj'}$.*

Definition 2 ($b_j \equiv b_j'$). *b_j is equivalent to b_j', denoted as $b_j \equiv b_j'$, if and only if $OPT_{bj} = OPT_{bj'}$.*

Definition 3 ($b_j \succ\equiv b_j'$). *b_j is not inferior to b_j', denoted as $b_j \succ\equiv b_j'$, if and only if $OPT_{bj} \le OPT_{bj'}$.*

Based on the above definitions, we formally define the notions of ONN and k-ONN queries, respectively.

Definition 4 (Optimal-Nearest-Neighbor query). *Given two datasets D_A and D_B, a spatial region R, and a critical distance d_c, an Optimal-Nearest-Neighbor (ONN) query finds an object $b_j \in D_B$, such that (i) b_j is outside R, and (ii) $b_j \succ\equiv b_j'$, where b_j' is any of the remaining object in D_B outside R.*

Definition 5 (k-Optimal-Nearest-Neighbor query). *Given two datasets D_A and D_B, a spatial region R, and a critical distance d_c, a k-Optimal-Nearest-Neighbor (k-ONN) query finds a set S_{rslt} of objects, such that (i) S_{rslt} contains k objects from D_B, (ii) $\forall\ b_j \in S_{rslt}$, b_j is outside R, and (iii) $\forall\ b_j' \in D_B' - S_{rslt}$, $b_k \succ\equiv b_j'$, where D_B' is the set of the objects in D_B that are not contained in R and b_k is the k-th ONN.*

In this paper, we study the problem of efficiently handling k-ONN ($k \ge 1$) queries.

III. ALGORITHMS FOR K-ONN QUERIES

In this section, we propose five algorithms, namely, *Forward Processing algorithm* (FP), *Reverse Processing algorithm* (RP), *Three-Step algorithm* (TS), *Reuse-based FP algorithm* (RFP), and *Reuse-based RP algorithm* (RRP), for dealing with the k-ONN query efficiently. Each algorithm takes as input an R-tree T_A on D_A, an R-tree T_B on D_B, the number of ONNs k, a spatial region R, a critical distance d_c, and outputs the heap H_{rslt} of k (≥ 1) ONNs.

A. Forward Processing Algorithm

Our first approach FP adopts a two-step framework that first retrieves a set of candidate ONNs and then evaluates their optimality to obtain the final query result.

Initially, FP retrieves from T_B all the points outside R whose distances from R are smaller than or equal to d_c, and stores them in a min-heap H_B. Then, it recursively evaluates every candidate point in H_B by repeatedly visiting T_A. Finally, H_{rslt} is returned as the query result.

B. Reverse Processing Algorithm

The above presented FP is expensive in terms of the I/O cost and CPU cost, because it requires traversing the tree T_A multiple times for evaluating all candidate points. Moreover, FP may access some unnecessary points (which do not contribute to the final query result) in the first step of the algorithm. To address these shortfalls, we present our second algorithm RP.

In the first place, RP finds all the points located within R by traversing T_A, and stores them in a min-heap H_A. Note that the order of the points in H_A is very important since if two consecutive points are close to each other, a large percentage of pages from T_B needed during the second query will be in the LRU memory buffer owing to the first one. Motivated by this, RP sorts the points in H_A using their Hilbert values and visits them in this order to maximize spatial locality. Once H_A is obtained, for each point e in H_A RP first retrieves from T_B a set st_B of the points outside R whose distances from e are within d_c, and then adds all the points in st_B to an information list, denoted as *InfoList*, and updates *InfoList* if necessary. Finally, RP evaluates the optimality of each entry in *InfoList*, inserts it into H_{rslt} as well as updates H_{rslt} if necessary, and outputs H_{rslt} as the answer.

C. Three-Step Algorithm

While the above RP can reduce the search space in comparison with FP, it still has two problems. First, RP traverses the tree T_B multiple times, leading to large numbers of redundant node accesses. Secondly, the maintenance of the *InfoList* structure costs a lot of memory space, especially for huge *InfoList* size. To reduce the I/O cost and CPU time, we develop our third algorithm TS which achieves a single traversal of the trees T_A and T_B.

TS follows a three-step framework. Specifically, first TS retrieves from T_B all the points outside R whose distances from R are within d_c, and stores them in a stack st_B sorted in increasing order of a certain coordinate (e.g., x-coordinate, assuming that the x-axis having the largest axis projection). Secondly, TS employs a spatial range query to find from T_A all the points that are located inside R, and preserves them in st_A, sorted in ascending order of their x-coordinates as well. Thirdly, TS recursively evaluates the optimality of every candidate point in st_B by distance computations and comparisons, and obtains the final query result. Here a brute-force approach would compute the distance $dist$ (a_i, b_j), $\forall\ a_i \in st_A$, $\forall\ b_j \in st_B$. Obviously, the method is costly. Fortunately, we can reduce this quadratic cost as follows: For each point $b_j \in st_B$, TS first finds from st_A the set P of the points whose x-coordinates fully fall into the range [$b_j.x - d_c$, $b_j.x + d_c$] and then for every point $e \in P$ it inserts e into a heap H_A if $dist$ (e, b_j) $\le d_c$. Next, b_j's optimality is evaluated as in FP.

D. Reuse-based FP Algorithm

As mentioned earlier, the main shortcoming of FP is that it incurs a great deal of unnecessary I/O overhead, since the algorithm needs to retrieve the tree T_A multiple times during the query processing. To overcome this limitation, we present our fourth algorithm RFP which can reduce the I/O cost significantly by utilizing reuse technology.

Like FP, RFP first finds from T_B all the points outside R whose distances from R are bounded by d_c, and stores them in a heap H_B. But unlike FP, it then recursively evaluates every point in H_B its optimality by repeatedly reusing all the entries (including non-leaf nodes and data points) visited so far.

E. Reuse-based RP Algorithm

Our fifth approach RRP is also inspired by the idea of reuse technology. Like RP, RRP first finds from T_A all the points that are contained in R, and stores them in H_A. However, unlike RP, for each point e in H_A, RRP first recursively retrieves from T_B all the data points outside R whose distances from e are smaller than or equal to d_c through repeatedly reusing all the entries (containing non-leaf nodes and data points) visited so far, and then adds them to an *InfoList* structure which is updated whenever necessary. Finally, RRP evaluates the optimality of each entry in *InfoList*, inserts it into H_{rslt} as well as updates H_{rslt} if necessary, and returns H_{rslt} as the answer.

IV. Experimental Evaluation

This section experimentally evaluates the efficiency and scalability of our proposed algorithms by using both real and synthetic datasets. All the algorithms were implemented in C++ and the experiments were conducted on a Pentium IV 3.0GHz PC with 1GB of RAM.

We have used four real data sets in the experiments. Specifically, *CL* and *RS* datasets represent different layers of North America's map, available at *http://www.maproom.psu.edu/dcw*. Datasets containing line segments were transformed to point datasets, by taking the middle point of each segment. *LB* and *CA* datasets contain 2D points representing geographical locations in Long Beach County and California respectively, available at *http://www.census.gov/geo/www/tiger*. For all the real datasets, each dimension of the data space is normalized to the range of [0, 10000]. We have also created several synthetic datasets with dimensionality in the range of [2, 5] and cardinality in the range of [50k, 450k], following uniform and Zipf distributions. The coordinates of each point in a *uniform* dataset are generated randomly in [0, 10000], whereas, for a *Zipf* dataset, the coordinates follow a Zipf distribution with a skew coefficient $\alpha = 0.8$.

TABLE I
Parameter Ranges and Default Values

Parameter	Range	Default
k	1, 2, …, 32, …, 512, 1024	32
d_c	250, 500, 750, 1000, 1250	750
R (% of full space)	0.25, 1, 2.25, 4, 6.25	2.25
dimensionality *dim*	2, 3, 4, 5	2
cardinality n (× 10K)	5, 10, …, 25, …, 40, 45	3.2, 12.8, 25.6
buffer size *bs* (pages)	0, 4, 16, 64, 256, 1024	0

Every dataset is indexed by R*-tree [1] with 4K bytes page size. We investigate the performance of the algorithms under various parameters which are summarized in Table 1. In each experiment, only one parameter varies while the others are fixed to their default values. Note that the spatial region R in our experiments resides in the middle of the entire data space and all of its edges have the same length.

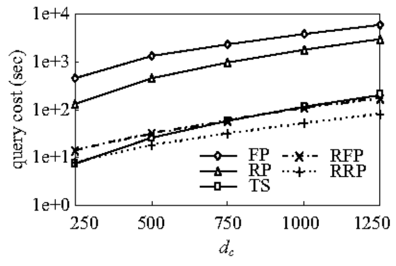

Fig. 1 Performance varying d_c on $D_A = CL$ and $D_B = RS$

The performance metric is the query cost (i.e., the sum of the I/O time and CPU time, where the I/O time is computed by charging 10ms for each page access). Unless specifically stated, the size of LRU buffer is 0 in the experiments, i.e., the I/O cost is determined by the number of nodes accessed.

In the first three sets of experiments, we study three factors, including k, d_c, and R, which can affect the performance of the algorithms. Towards this, we deploy three different combinations of the real datasets, i.e., (i) $D_A = RS$ and $D_B = CL$, (ii) $D_A = CL$ and $D_B = RS$, and (iii) $D_A = LB$ and $D_B = CA$, for representing different size cases in practice. In the subsequent experiments, we compare the cost of the algorithms on synthetic datasets wrt different parameters. We plot only the query cost of the algorithms as a function of d_c in Figure 1 due to the space limitation. From all the experimental results, we can conclude that RRP is the best algorithm in various settings. Although TS and RFP are less efficient than RRP, they are always more efficient than FP and RP significantly. When the cardinalities of the two datasets are small, TS is a good choice. FP and RP are clearly inappropriate for ONN queries, because they are consistently outperformed by the other three algorithms.

V. Conclusion

We have presented in this paper the first piece of work that tackles ONN queries, a new class of NN search. ONN is not only interesting from a research point of view but also useful in many practical applications involving spatial data. We have proposed several algorithms for processing the ONN query efficiently, and experimentally evaluated their performance under a variety of settings.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, pp. 322–331, 1990.

[2] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. Abbadi. Constrained Nearest Neighbor Queries. In *SSTD*, pp. 257–278, 2001.

[3] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *TODS*, 24(2):265–318, 1999.

[4] F. Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. In *SIGMOD*, pp. 201–212, 2000.

[5] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *SIGMOD*, pp. 71-79, 1995.