An Efficient Method for Min-dist Location Selection

Jianzhong Qi[1] **Zhenghua Xu**[2] **Yuan Xue**[3] **Zeyi Wen**[4]

Department of Computer Science and Software Engineering
University of Melbourne,
111 Barry St, Carlton, Victoria 3053, Australia

Email: [1,2,4]{jiqi,zhxu,zeyiw}@csse.unimelb.edu.au
[3]yuaxue@unimelb.edu.au

Abstract

Given a set of clients and a set of existing facilities, the min-dist location selection query returns a location from a given set of potential locations for establishing a new facility so that the average distance between a client and her nearest facility is minimized. This type of queries has a wide range of applications in marketing, decision support systems, urban development simulations and massively multiplayer online games. The computational cost of a naive algorithm, which sequentially scans all the potential locations, is too high to process this type of queries in real time. Motivated by this, we propose a branch and bound algorithm that exploits geometric properties of the data objects and early prunes unpromising potential locations from consideration to achieve a higher query processing efficiency. We conduct a detailed cost analysis and extensive experiments to validate the efficiency of the branch and bound algorithm. The results show that the proposed algorithm outperforms the naive algorithm constantly.

Keywords: Spatial databases, optimal location selection, Min-dist metric

1 Introduction

Many businesses or organizations manage large numbers of facilities. For example, Walmart has warehouses, and Australia Post has branch offices. It is a common need for a business to add facilities. For example, Walmart might want to add a warehouse to reduce the distances between its stores and warehouses; Australia Post might want to add a new post office in an emerging suburb to reduce the traveling distances for their postmen. Usually, a set of potential locations is available, typically from a real estate agent (e.g., real estate websites provide hundreds of thousands of places for renting or buying).

This paper investigates the *min-dist location selection query, which selects a location (for Walmart or Australia Post) that minimizes the average distance between a client (a chain store or an addressee) and her nearest facility (warehouse or post office) to reduce logistics cost or to improve the quality of service.* We assume that the business has knowledge about its client distribution from surveys or past sales records.

The min-dist location selection query also has other applications as follows. In urban development simulation, very often urban planners need to simulate the above location selection procedure, so that the influence of establishing a new public facility or business center on the residents can be observed. In the multi-billion dollar computer game industry, massively multiplayer online games (MMOGs) like *World of Warcraft* have group quests for players to complete in teams, which mostly involve killing mobs (monsters or other non-player characters). As the quests often take players days or even weeks to complete, it is common for players to leave and rejoin the game during a quest. When a player rejoins the game, the subquest she was on may have been completed by her teammates and the team has moved on to another region to complete other subquests. It would be a waste of time for this player to rejoin the game from where she left. A very helpful utility for the game is selecting a starting point from a set of preset rejoin locations to minimize the average distance between a mob and its nearest player, so that players can focus on fighting mobs and complete quests rather than walking.

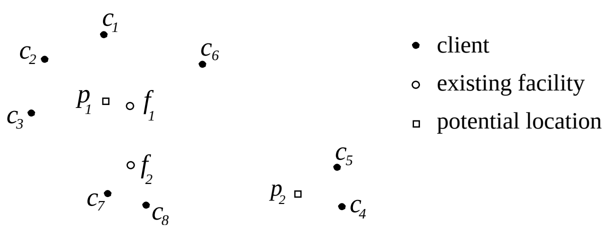


Figure 1: An example for the query

The example in Figure 1 illustrates the query: $\{c_1, c_2, ..., c_8\}$ is a set of clients (residents or mobs), $\{f_1, f_2\}$ is a set of existing facilities (public facilities or teammates) and $\{p_1, p_2\}$ is a set of potential locations (candidate locations for new facility establishment or rejoin). Now we need to select one from the potential locations to establish a new facility. Before adding a new facility, f_1 is the nearest facility of c_1, c_2, c_3 and c_6; f_2 is the nearest facility of c_4, c_5, c_7 and c_8. If a new facility is established at p_1, it will become the nearest facility for c_1, c_2 and c_3. If a new facility is established at p_2, it will become the nearest facility of c_4 and c_5. As we can observe, p_2 results in a smaller average distance between a client and the nearest facility, so it is selected as the answer.

Although an existing commercial software (ArcGIS 2011) can solve several kinds of simpler location optimization problems, none can solve the min-dist location selection problem, as we will detail in Section 2.

1.1 Contributions and Organization of the Paper

In this paper, we examine solutions to the min-dist location selection query and make the following contributions.

- We formulate the min-dist location selection query.
- We analyze the properties of the query and propose pruning techniques to reduce the search space for processing the query.
- Based on the proposed pruning techniques, we propose a branch and bound method to efficiently process the query.

- We perform an analytical cost study and an extensive experimental study. The results confirm the effectiveness of the proposed pruning techniques and the efficiency of the proposed algorithm.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formulates the query and presents a naive algorithm. Section 4 presents the branch and bound method. Section 5 analyzes the cost of the proposed method and Section 6 presents the experimental results. Finally, Section 7 concludes the paper.

2 Related Work

We review four categories of work below: work on the *nearest neighbor (NN)* query, work on the *reverse nearest neighbor (RNN)* query, work on *max-inf* location optimization problems and work on *min-dist* location optimization problems.

NN query: Given a set of objects S and a query object q, the NN query returns q's nearest objects in S. Two popular NN query processing algorithms are the depth-first algorithm (Roussopoulos et al. 1995) and the best-first algorithm (Hjaltason & Samet 1995).

RNN query: Korn & Muthukrishnan (2000) first propose the reverse nearest neighbor (RNN) query and define the RNNs of an object o to be the objects whose respective nearest neighbor is o. In the same paper, Korn and Muthukrishnan propose an *RNN-tree* based solution to the query, where the RNN-tree is an R-tree (Guttman 1984) variant that indexes NN circles of the data objects rather than the data objects themselves. Here, the NN circle of an object o is defined to be a circle that centers at o with its radius being the distance between o and o's nearest neighbor. Based on the NN circles, to find the RNN of an object o only requires checking which objects' NN circles enclose o. However, the RNN-tree based solution has two major drawbacks. One is that it requires the extra maintenance of an RNN-tree. The other is that it requires precomputing the NN circles. Therefore, this solution cannot handle objects with frequent updates. To solve the first problem, Yang & Lin (2001) propose to integrate the NN circle information into an R-tree that indexes the clients themselves, so that the resultant R-tree variant, the *RdNN-tree*, can be used to process RNN queries as well as other common types of queries on the clients, and thus avoid the maintenance of an extra RNN-tree. To solve the second problem, Stanoi et al. (2001) propose an approximation-refinement framework to compute the RNNs on the fly, so that no precomputation is needed.

There are also studies (Tao et al. 2006, Wu et al. 2008, Achtert et al. 2009, Cheema et al. 2010) that work on RNN query variants. A most relevant variant, the *reverse k nearest neighbor (RkNN)* query, extends the answer set to include objects who perceive the query object as among their k nearest neighbors. Wu et al. (2008) study the RkNN query on continuously moving objects, which correlates two sets of moving objects according to their closeness.

While these methods work well for processing a single R(k)NN query, they are not designed to compute R(k)NNs for large amount of objects at the same time, which is one of the key difficulties in the related location optimization problems. Thus, the R(k)NN problem can be viewed as a sub problem of the related location optimization problems, but its problem solving techniques do not solve the related location optimization problems.

Max-inf problems: Max-inf location optimization problems aim at maximizing the influence of the locations, where the influence of a location is defined by the number of clients it attracts. Here, the concept of "*attract*" can have different meanings in different max-inf problems. Cabello et al. (2005) propose a facility location problem based on the MAXCOV optimization criterion, which is to find regions in the data space to maximize the numbers of RNNs for the points in these regions. Figure 2

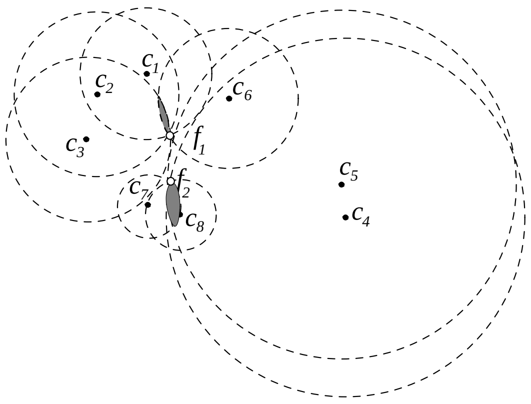


Figure 2: A max-inf problem (Cabello et al. 2005)

gives an example, where the gray regions are the optimal regions. Points in these regions have 4 RNNs, while any point outside of these regions has at most 3 RNNs. Cabello et al. (2005) introduce the concept of *nearest location circle (NLC)* to solve the problem, where the NLC of a client c is a circle centered at c with its radius being the distance between c and c's nearest existing facility. To find the solution for the MAXCOV criterion based problem is to find the regions that are enclosed by the largest number of NLCs, which requires complex computations. The study only gave a theoretical analysis of the problem. Wong et al. (2009) study this problem further and propose a method to compute the regions overlapped by the largest number of NLCs. Xia et al. (2005) propose the top-t most influential sites problem and a branch and bound algorithm to solve the problem. This problem finds the top-t most influential existing sites within a given region Q. It does not consider any potential locations. Du et al. (2005) propose to find a point from a continuous candidate region that can maximize the influence value. They use L_1 as the distance metric and have a strong assumption that all the roads are either horizontal or vertical. Cheema et al. (2011) propose to find an influence zone for a query location p, where the clients inside this zone form exactly the RkNN query result of p. A more recent study (Huang, Wen, Pathan, Taylor & Zhang 2011) ranks the candidate locations according to their influence values and another study (Huang, Wen, Qi, Zhang, Chen & He 2011) contributes in efficient algorithms to compute the top-k most influential candidate locations. Unlike the above problems, which relate the influence values to the cardinalities of RNN sets, Gao et al. (2009) propose to find the optimal location p outside a given region Q based on locations' *optimality*, where the optimality of a location p is modeled by the amount of clients in Q whose distances to p is within a given threshold t. Intuitively, the more clients p attracts, the greater its optimality. These studies differ from ours in optimization functions and other settings. Thus, their solutions do not apply.

Min-dist problems: Zhang et al. (2006) propose the min-dist optimal-location problem. Given a client set C and an existing facility set F, it finds points within a given region Q so that if a new facility is established at any one of these points, the average distance of the clients to their respective nearest facilities is minimized. Figure 3 gives an example, where pt may be one of the points in the answer set and it is different from the solution p_2 to our problem. To solve this problem, assuming L_1 distance metric and all the roads are either horizontal or vertical, Zhang et al. (2006) propose a method that divides the solution

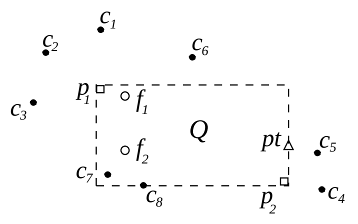


Figure 3: The min-dist optimal-location problem

space progressively to minimize the candidate region until the answer set is found. Approximate solutions are provided during the process of refinement.

Zhang et al.'s problem definition has the same min-dist optimization function as ours, but our problem definition has an additional set of potential locations given as candidates for selection. **In many real applications, we can only choose from some candidate locations**, e.g., Walmart may only establish a new warehouse at a place for rent or sale rather than anywhere in a region. Zhang et al.'s query may return points or a region in the middle of a river or a football field, which is not a useful result for our applications. Moreover, *Zhang et al.'s algorithm relies on L_1 distance assumption* (i.e., distance must follow vertical or horizontal roads), making it inapplicable for the urban development simulation or MMOG applications, where people can walk freely. Our problem definition assumes L_2 distance, which suits our applications better.

Note that in computational geometry, given a set C of object locations (e.g., clients), the k-medoid query (Mouratidis et al. 2005) finds a set of medoids $C' \subseteq C$ with cardinality k that minimizes the average distance from each object $c \in C$ to its closest medoid in C. The k-median query is a variation, where we find k locations called medians, not necessarily in C, which minimize the average distance (from each object $c \in C$ to its closest median). These two types of queries are actually using the min-dist metric. However, our problem is different from both of them. A fundamental difference is that these problems do not assume a set of existing facilities or a set of potential locations, but we do. If there is at least one existing facility or some potential locations to be chosen from in a specific location selection problem, k-medoid queries or k-median queries do not apply.

Related commercial software: As mentioned in the Introduction, an existing commercial software (ArcGIS 2011) can solve several kinds of simpler location optimization problems. The most related problem this software can solve is called the *minimize impedance query*, which finds locations for a set of new facilities to minimize the sum of distances between clients and their respective nearest facilities. However, this problem does not consider existing facilities. If we use this software to find a set of locations S_l for new facilities, there is no guarantee that S_l will contain all the points in the set of existing facilities F. Therefore, this software does not solve our problem.

3 Preliminaries

This section formulates the min-dist location selection query and presents a naive algorithm to process the query. Frequently used symbols are summarized in Table 1.

3.1 Problem Formulation

All data objects (clients, facilities and potential locations) are represented by points in an Euclidean space. We may refer to the data objects as *data points* or simply as *points*.

Table 1: Frequently Used Symbols

Symbols	Explanation
e	An entry in a R-tree node
o	Any point in the data space
$dist(o_1, o_2)$	The distance between two points o_1 and o_2
C, F, P	The set of clients, existing facilities and potential locations, respectively
n_c, n_f, n_p	Cardinality of C, F, and P, respectively
c, f, p	A client in C, an existing facility in F and a potential location in P, respectively

Let $dist(o_1, o_2)$ denote the distance between two points o_1 and o_2, and n_c be the number of clients. The min-dist location selection query is defined as follows.

Definition 1. ***Min-dist location selection query****.*
Given a set of points C as clients, a set of points F as existing facilities and a set of points P as potential locations, the min-dist location selection query finds a potential location $p_m \in P$ for a new facility to be established at, so that $\forall p \in P$:[1]

$$\frac{\sum_{c \in C}\{\min\{dist(c,o)|o \in F \cup p_m\}\}}{n_c}$$
$$\leq \frac{\sum_{c \in C}\{\min\{dist(c,o)|o \in F \cup p\}\}}{n_c}.$$

Since the denominator is the same on both sides of the inequality, the problem is equivalent to minimizing the sum (instead of the average) of the distances between the clients and their respective nearest facilities.

3.2 A Naive Algorithm

A straightforward algorithm to the min-dist location selection query is to sequentially check all potential locations. For every new potential location p, we compute the sum of the distances of all clients to their respective nearest facilities. The potential location with the smallest sum is the answer. We call this algorithm the *sequential scan (SS)* algorithm.

In SS, repeatedly finding the nearest facility to each client for every potential location is too expensive. Therefore, we precompute the distances of all the clients to their respective nearest facilities and store the distances. This precomputation involves a nested loop iterating through every client and for every client iterating through every facility. *K*NN-join algorithms (e.g., Yu et al. (2010)) can do this more efficiently and maintain the results dynamically when clients and facilities are updated. The SS algorithm with precomputation is shown in Algorithm 1, where $c.dnn(c, F)$ denotes c's precomputed distance to her closest existing facility and is stored with c's record.

We see that even with precomputation SS is still very costly as it has to access the whole client dataset $\frac{n_p}{C_p}$ times, where n_p is the cardinality of P and C_p is the capacity of a block for P (assuming we read P in disk blocks). Therefore, the need for an efficient algorithm is obvious.

4 A Branch and Bound Method

In this section, we propose a brand and bound method that exploits data objects' geometric properties to prune unpromising potential locations from consideration, so that the min-dist location selection query can be processed more efficiently. This method requires the query to be

[1]Note that there may be ties in the average distances. To simplify our discussion, we always return the first potential location found that have the smallest average distance.

ALGORITHM 1: SS(C, P)

```
1  $optLoc \leftarrow$ NULL; // $optLoc$ is the optimal location;
2  for $p \in P$ do
3      $p.distSum \leftarrow 0$;
4      for $c \in C$ do
5          if $dist(p, c) < c.dnn(c, F)$ then
               // $c.dnn(c, F)$ is precomputed
6              $p.distSum \leftarrow p.distSum + dist(p, c)$;
7          else
8              $p.distSum \leftarrow$
               $p.distSum + c.dnn(c, F)$;
9      if $optLoc =$ NULL or
       $p.distSum < optLoc.distSum$ then
10         $optLoc \leftarrow p$;
11 return $optLoc$;
```

redefined in a form that enables the computation for the bounds. Next, we start with redefining the query.

4.1 Query Redefinition

We call the distance between a client c and her nearest facility the *nearest facility distance (NFD)* of c. Let $dnn(o, S)$ denote the distance between a point o and its nearest point in a set S. Then $dnn(c, F)$ and $dnn(c, F \cup p)$ denote the NFD of c before and after a new facility is established on a potential location p, respectively. The min-dist location selection query is actually minimizing the sum of all the clients' NFD.

If o is a point not in the set F and $dist(c, o) < dnn(c, F)$, then establishing a new facility at o will reduce the NFD of c. In this case, we say that c can get an *NFD reduction* from o. We define the *influence set* of o, denoted by $IS(o)$, as the set of clients that can get NFD reduction from o. Formally, $IS(o) = \{c | c \in C, dist(c, o) < dnn(c, F)\}$. The influence set of a potential location p includes all the clients that will reduce their NFD if a new facility is established at p. For example, in Figure 1, $IS(p_1) = \{c_1, c_2, c_3\}$, and $IS(p_2) = \{c_4, c_5\}$.

If $IS(p) \neq \emptyset$ for a potential location p, then establishing a new facility at p will reduce the sum of the clients' NFD. We call the sum of the clients' NFD reduced by p the *distance reduction* of p, denoted by $dr(p)$. Formally, $dr(p) = \sum_{c \in IS(p)} (dnn(c, F) - dnn(c, F \cup p))$. Minimizing the sum of the clients' NFD when adding a facility on p is equivalent to maximizing $dr(p)$. Therefore, the min-dist location selection query can be redefined as follows.

Definition 2. *Given a set of points C as clients, a set of points F as existing facilities and a set of points P as potential locations, the min-dist location selection query finds a potential location $p_m \in P$, so that $\forall p \in P$: $dr(p) \leq dr(p_m)$.*

4.2 The Branch and Bound Algorithm

The *Branch and Bound (BB)* method estimates the potential locations' dr values to achieve early pruning. It assumes that the datasets are indexed in spatial indexes. Specifically, it assumes an R-tree R_P to index the potential location set P, and an R-tree variant R_C^b to index the client set C and store some other information for the computing the bounds (details are in Section 4.3), although the method can be easily adapted to any hierarchical spatial index. The branch and bound scheme is performed during a traversal on both trees.

Before explaining the BB method, we need to extend the concept of the influence set of a point to the *influence set of a node* in R_P. Let N_P be a node in R_P. The influence set of the node N_P is defined as $IS(N_P) = \{c | c \in C$ and $\exists o \in N_P.mbr : dist(c, o) < dnn(c, F)\}$. A client is in $IS(N_P)$ if there is a point (not necessarily a potential location) in the minimum bounding rectangle (MBR) of N_P that can reduce the client's NFD. Intuitively, $IS(N_P)$ defines the set of clients which might achieve distance reduction without knowing which potential locations are actually in N_P; it is the union of $IS(o)$ for any possible point o in the MBR of N_P. Figure 4 gives an example. Node N_P indexes two potential locations p_1 and p_2. We can observe that $IS(p_1) = \{c_1, c_2, c_3\}$ and $IS(p_2) = \{c_4, c_5\}$. Also, there are three points o_1, o_2 and o_3 in the MBR of N_P. We have $dist(o_1, c_6) < dnn(c_6, F)$, $dist(o_2, c_7) < dnn(c_7, F)$ and $dist(o_3, c_8) < dnn(c_8, F)$. Therefore, $IS(N_P) = \{c_1, c_2, ..., c_8\}$.

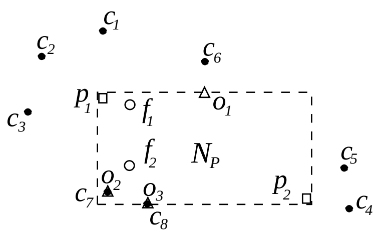


Figure 4: $IS(N_P)$

The idea of the BB method is as follows. We traverse R_P in a depth-first order and simultaneously traverse the R-tree variant on C, R_C^b (recall that this R-tree maintains some additional information for computing bounds). We use the tree structure to narrow down the clients we have to examine for identifying the influence sets of a potential location. As we visit a node N_P of R_P (suppose N_P is in level i of R_P), we identify a set of nodes from level i of R_C^b whose subtrees must cover all the clients in $IS(N_P)$; we call this set of nodes from R_C^b the *influence nodes* (IN) of N_P and denote it by $IN(N_P)$. Based on the MBR of N_P and the aggregate information stored in the nodes of $IN(N_P)$, we can compute a lower bound and an upper bound for the distance reduction of all the potential locations contained in the subtree rooted at N_P. As we traverse down R_P, the bounds will become tighter. We record the largest lower bound so far during traversal, which serves as a pruning distance, denoted as pd. If at any time we encounter a node in R_P with an upper bound of distance reduction smaller than pd, then that node can be discarded from the search, since our goal is to find the potential location with the largest distance reduction. When we reach the leaf level of R_P, we get the exact information of the potential locations and can compute their exact distance reductions. If pd is smaller than an exact distance reduction, then pd gets updated to it. The search stops as we finish traversing R_P and the potential location with the largest dr value is the answer.

The derivation of the upper bound $maxdr$ and the lower bound $mindr$ are presented in Sections 4.3 and 4.4, respectively. The condition to identify $IS(N_P)$ of N_P, and the structure of R_C^b are related to the upper bound, so we present them in Section 4.3.

The recursive part of the BB algorithm is summarized in Algorithm 2. We explain the algorithm together with the example in Figure 5, where the nodes within a dotted rectangle represent the IN of a node in R_P. Initially, N_P is set to the root node of R_P, which is N_0, IN is set to the root node of R_C^b, N_0^b, pd is set to 0 and $optLoc$ is set to NULL. For each node N_P being accessed, we construct $IN(e_p)$ for each of its entry e_p using the child nodes of the nodes in $IN(N_P)$ (lines 1 to 3). In Figure 5, we have $IN(e_1) = \{N_1^b\}$, $IN(e_2) = \{N_2^b, N_3^b\}$ and $IN(e_3) = \emptyset$. These actually means $IN(N_1) = \{N_1^b\}$, $IN(N_2) = \{N_2^b, N_3^b\}$ and $IN(N_3) = \emptyset$. Then the child nodes of N_1, N_2 and N_3 will use the child nodes of these INs to construct their own INs. For example, $IN(N_{11})$ is constructed with the child nodes of N_1^b, and the resultant $IN(N_{11})$ is $\{N_{11}^b, N_{12}^b\}$. Similarly, $IN(N_{21})$ and $IN(N_{22})$ are constructed with the child nodes of N_2^b and N_3^b, which results in $IN(N_{21}) = \{N_{21}^b\}$ and $IN(N_{22}) = \{N_{22}^b, N_{23}^b, N_{31}^b\}$. This ensures that the level of $IN(N_P)$

ALGORITHM 2: BB(N_P, IN, pd, $optLoc$)

```
1  if $N_P$ is a non-leaf node then
2      for $e_p \in N_P$ do
3          Construct $IN(e_p)$, compute $maxdr$ and
           $mindr$;
4          if $maxdr > pd$ then
5              if $mindr > pd$ then
6                  $pd \leftarrow mindr$;
7              BB($e_p.childnode$, $IN(e_p)$, $pd$, $optLoc$);
8  else if $N_P$ is a leaf node but nodes in $IN$ are
   non-leaf nodes then
9      for $N_C^b \in IN$ do
10         $IN' \leftarrow \emptyset$;
11         for $e_c^b \in N_C^b$, $e_c^b$ satisfies IN conditions of
           $N_P$ do
12             $IN' \leftarrow IN' \cup e_c'.childnode$;
13         BB( $N_P$, $IN'$, $pd$, $optLoc$ );
14 else
       // $N_P$ and nodes in $IN$ are all
          leaf nodes
15     for $N_C^b \in IN$ do
16         for $e_p \in N_P$ do
17             for $e_c^b \in N_C$,
               $dist(e_p, e_c^b) < e_c^b.dnn(c, F)$ do
18                 $e_p.dr \leftarrow$
                   $e_p.dr + e_c^b.dnn(c, F) - dist(e_p, e_c^b)$;
19             if $optLoc =$ NULL or $e_p.dr > optLoc.dr$
               then
20                 $optLoc \leftarrow e_p$;
21     if $optLoc.dr > pd$ then
22         $pd \leftarrow optLoc.dr$;
```

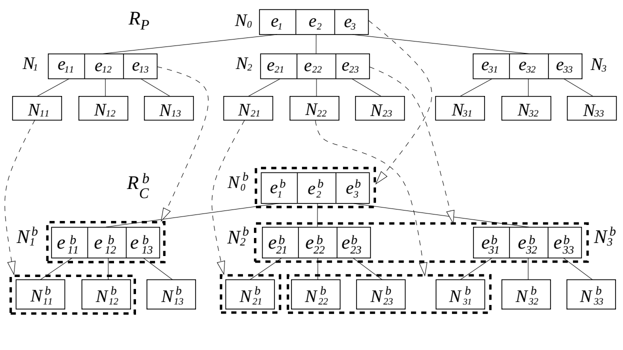


Figure 5: Example of algorithm BB

and that of N_P are the same. The bounds $maxdr$ and $mindr$ of an entry e_p are computed using the aggregated attributes stored in the parent entries of $IN(e_p)$. The child node of e_p will be pruned if $maxdr \leq pd$ (line 4). if $mindr > pd$, pd will be updated to be $mindr$ (lines 3 to 6). Then the child nodes of the unpruned entries of N_P are traversed (line 7). Note that the heights of R_P and R_C^b may be different. Thus, there are different procedures for the condition where the traversal reaches the leaf level of only one tree. (i) If the traversal reaches an non-leaf node N_P of R_P, and $IN(N_P)$ are leaf nodes, we construct the INs of the child nodes of N_P using nodes of $IN(N_P)$ since there is no child node for the nodes in $IN(N_P)$ (lines 1 to 7). (ii) If the traversal reaches a leaf node N_P of R_P, and $IN(N_P)$ are non-leaf nodes, we construct a subset of $IN(N_P)$ with the entries of each node $N_C^b \in IN(N_P)$, denote it as $IN'(N_P)$, and perform algorithm BB on N_P and $IN'(N_P)$ (lines 8 to 13). Doing this recursively guarantees that all clients contained in the subtrees of the nodes in $IN(N_P)$ will be accessed, which means $IS(N_P)$ is fully accessed. The advantage of this method compared to the construction of INs for the data entries directly is its reduction of node accesses in R_C^b. Since entries of a same node tend to have similar $IN's$, if we access the nodes in $IN(N_P)$ directly for each data entry of N_P, many nodes (and their descendant nodes) will be accessed repeatedly and the number of node accesses will be large. When the traversal reaches the leaf nodes of both trees, for each data entry e_p of a node N_P of R_P, all entries of each node $N_C^b \in IN(N_P)$ are accessed to update $e_p.dr$ (lines 14 to 18); $optLoc$ and pd are updated accordingly (lines 19 to 22). When the traversal ends, IN has been accessed for all potential locations and $optLoc$ is found.

4.3 An Upper Bound

We derive an upper bound $maxdr$ for the distance reduction of all potential locations contained in the subtree rooted at a node N_P. To simplify our discussion, we use $sub(N)$ to denote the set of data entries contained in the subtree rooted at the node N. Then $|sub(N)|$ denotes the cardinality of $sub(N)$. For example, $sub(N_P)$ denotes the set $\{p|p$ is a potential location indexed in the subtree root at node $N_P\}$.

The following discussion holds for any R-tree based index on the set of clients, we use N_C (instead of N_C^b used in the above subsection) to denote a node in such an index. Recall the definition of dr of a potential location p, $dr(p) = \sum_{c \in IS(p)}(dnn(c, F) - dnn(c, F \cup p))$. We derive $maxdr$ in a similar way. Since we are using a set of nodes to compute $maxdr$ for a node N_P of R_P, $maxdr(N_P) = \sum_{N_C \in S}(|sub(N_C)| \cdot (g_1(N_C) - g_2(N_C, N_P)))$, where S is a set of client R-tree nodes, g_1 is a metric related to a node N_C, and g_2 is a metric related to N_C and N_P. We use $|sub(N_C)|$ because, in an ideal condition, every client $c \in sub(N_C)$ is in $IS(p)$ of some potential location $p \in N_P$. To derive a reasonable upper bound, we try to find small S and $g_1(N_C)$ and a large $g_2(N_C, N_P)$. We use the metrics, $maxFDist$ and $minDist$, as g_1 and g_2, respectively. $IN(N_P)$ is used as S, which will be established based on $maxFDist$ and $minDist$. The definitions of $maxFDist$ and $minDist$ are as follows.

Definition 3. *The largest NFD value ($maxFDist$) for all clients that are in the subtree rooted at a node N_C.*

This metric is proposed by Yang & Lin (2001) as max_dnn and it is defined in a bottom-up fashion. For the leaf nodes, $maxFDist(N_C)$ is defined as the largest NFD value for all the clients indexed in N_C, i.e., $maxFDist(N_C) = \max\{dnn(c, F)|c \in N_C\}$, while for the non-leaf nodes, $maxFDist(N_C)$ is defined as the largest $maxFDist$ value for all the child nodes of N_C, i.e, $maxFDist(N_C) =$

$\max\{maxFDist(e_c.childnode)|e_c \in N_C\}$.

Figure 6 gives an example, where $dnn(c_3, F)$, $dnn(c_6, F)$ and $dnn(c_4, F)$ are the largest NFD values of the clients indexed in the leaf nodes N_1, N_2 and N_3, respectively. The three NFD values are picked as the respective $maxFDist$ values of the three nodes. Meanwhile, the largest value among $maxFDist(N_1)$, $maxFDist(N_2)$ and $maxFDist(N_3)$ is picked as the $maxFDist$ of the parent node (N_4) of these three nodes. Therefore, we get $maxFDist(N_4) = maxFDist(N_3) = dnn(c_4, F)$, which is effectively the largest NFD value for all the clients in the subtree rooted at N_4.

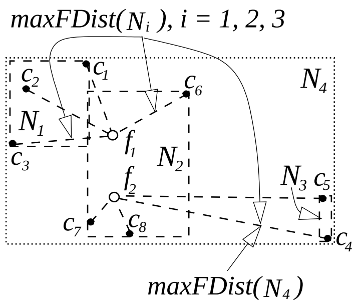


Figure 6: Example of $maxFDist$

Definition 4. *The smallest distance ($minDist$) between two objects.*

This metric is proposed by Roussopoulos et al. (1995). We use $minDist(c, N_P)$ to denote the smallest distance between a point c and the MBR of a node N_P. If c is within the MBR of N_P, then $minDist(c, N_P) = 0$. Otherwise, $minDist(c, N_P)$ is the distance between c and its nearest point on the MBR of N_P. Similarly, we use $minDist(N_C, N_P)$ to denote the smallest distance between the MBR of N_C and the MBR of N_P. If these two MBRs overlap, then $minDist(N_C, N_P) = 0$. Otherwise, $minDist(N_C, N_P)$=min $\{dist(o_1, o_2)|o_1, o_2$ are points on the MBRs of N_C and N_P, respectively$\}$. (Figure 7)

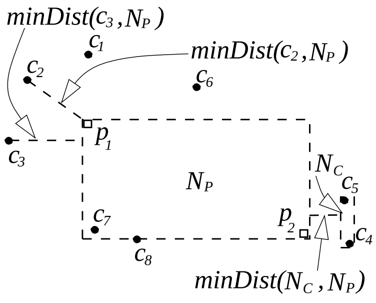


Figure 7: Example of $minDist$

The following theorem guarantees that the subtrees of the nodes in $\{N_C|minDist(N_C, N_P) < maxFDist(N_C)\}$ cover all the clients in $IS(N_P)$. This set defines $IN(N_P)$.

Theorem 1. *Given two nodes N_C and N_P, $sub(N_C) \cap IS(N_P) = \emptyset$ if $minDist(N_C, N_P) \geq maxFDist(N_C)$.*

Proof. According to the definitions of $minDist$ and $maxFDist$, we have:

$$\begin{aligned}&(1)\forall c \in sub(N_C)\forall p \in sub(N_P):\\ &\qquad dist(c, p) \geq minDist(N_C, N_P);\\ &(2)\forall c \in sub(N_C):\\ &\qquad maxFDist(N_C) \geq dnn(c, F).\end{aligned}$$

Hence, if $minDist(N_C, N_P) \geq maxFDist(N_C)$, we can obtain:

$$\begin{aligned}&\forall c \in sub(N_C)\forall p \in sub(N_P):\\ &\qquad dist(c, p) \geq dnn(c, F).\end{aligned}$$

Therefore, $sub(N_C) \cap IS(N_P) = \emptyset$. □

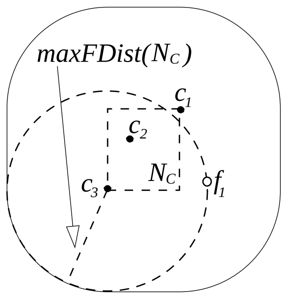


Figure 8: Example of Theorem 1

Figure 8 illustrates Theorem 1. Here, the rounded rectangle represents a region where $minDist(N_C, N_P) < maxFDist(N_C)$. If a point p lies outside the rectangle, it satisfies the condition of Theorem 1, hence no client $c \in sub(N_C)$ is contained by $IS(p)$.

The following theorem defines and proves the upper bound $maxdr$.

Theorem 2. *The following expression defines an upper bound for the dr values of all data points indexed in the subtree rooted at a node N_P of R_P.*

$$\sum_{N_C \in IN(N_P)}\{|sub(N_C)|\cdot [maxFDist(N_C) - minDist(N_C, N_P)]\}$$

Proof. An implicit statement about the definition of $dr(p)$ (cf. Section 4.1) for a potential location p is that $dnn(c, F) > dist(c, p)$. However, this may not be true if $IN(N_P)$ is used instead of $IS(p)$ when $c \in sub(N_C)$ and $N_C \in IN(N_P)$ but $c \notin IS(p)$. Let us define a set $S_p(N_C)$ which contains the clients in $sub(N_C)$ who are also in $IS(p)$. Formally,

$$\begin{aligned}S_p(N_C) = \{c|&dnn(c, F) - dist(c, p) > 0,\\ &c \in sub(N_C), N_C \in IN(N_P), p \in sub(N_P)\}.\end{aligned}$$

We can see that $S_p(N_C) \subseteq sub(N_C)$. Take advantage of the following relationships.

$$\begin{aligned}&(1)\forall p \in sub(N_P):\\ &\{c|c \in sub(N_C), N_C \in IN(N_P)\} \supseteq IS(N_P) \supseteq IS(p);\\ &(2)\forall c \in sub(N_C):\\ &\qquad maxFDist(N_C) \geq dnn(c, F);\\ &(3)\forall c \in sub(N_C)\forall p \in sub(N_P):\\ &\qquad minDist(N_C, N_P) \leq dist(c, p).\end{aligned}$$

For any potential location $p \in sub(N_P)$, we obtain:

$$\begin{aligned}&dr(p) = \sum_{c \in IS(p)}[dnn(c, F) - dist(c, p)]\\ &= \sum_{N_C \in IN(N_P)}\sum_{c \in S_p(N_C)}[dnn(c, F) - dist(c, p)]\\ &\leq \sum_{N_C \in IN(N_P)}\sum_{c \in S_p(N_C)}[maxFDist(N_C) - minDist(N_C, N_P)]\\ &\leq \sum_{N_C \in IN(N_P)}\sum_{c \in sub(N_C)}[maxFDist(N_C) - minDist(N_C, N_P)]\\ &= \sum_{N_C \in IN(N_P)}\{|sub(N_C)|\cdot [maxFDist(N_C) - minDist(N_C, N_P)]\}\end{aligned}$$

Thus, the upper bound holds. □

To compute $maxdr$ in the process of traversal, each entry e_c^b of R_C^b stores $|sub(N_C^b)|$ and $maxFDist(N_C^b)$, denoted as $cNum$ and $maxFDist$, for its child node N_C^b. For the data entries, $cNum = 1$ and $maxFDist = dnn(e_c^b, F)$. Recursively from the leaf nodes to the root node, the values of the two new attributes can be computed for each entry of the non-leaf nodes based on their definitions. When there is an update in R_C^b, the structure can be maintained efficiently in a similar recursive manner. Tree R_C^b can also be used in processing conventional queries on an R-tree efficiently, since adding two attributes will not significantly increase the height of the tree. This will be validated in our cost analysis and the experiments.

4.4 A Lower Bound

The way we derive the lower bound $mindr$ is similar to that of deriving $maxdr$. We define $mindr(N_P)$ in the form of $\max\{g_1(N_C) - g_2(N_C, N_P)|N_C \in S\}$, where S is a set of client R-tree nodes, g_1 is a metric related to a node N_C, and g_2 is a metric related to N_C and N_P. We use the maximum value instead of the sum value because it is possible for the $IS(p)$ of a potential location p to be indexed in only one subtree rooted at a node of R_C^b. We use $maxFDist$ as g_1, and $minExistDNN$ as g_2. $IN(N_P)$ is used as S. Metric $minExistDNN$ is defined based on $minMaxDist$. These two metrics are proposed by Roussopoulos et al. (1995) and Xia et al. (2005), respectively. We present their definitions before presenting the definition of $mindr$.

Definition 5. *The minimum upper bound of the distance between a point and its nearest data point o in another MBR ($minMaxDist$).*

$minMaxDist(c, N_P)$ denotes the minimum upper bound of the distance from a client c to her nearest potential location in $sub(N_P)$. It is the distance between c and the second nearest corner of $N_P.mbr$ since there must be a potential location p on the side joining the nearest and the second nearest corners, and the distance between c and p must be equal to or less than $minMaxDist(c, N_P)$. (Figure 9)

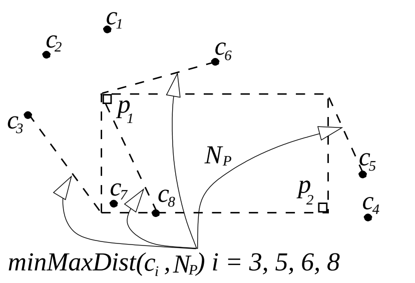


Figure 9: Example of $minMaxDist$

Definition 6. *The minimum upper bound of the distance between a point in the MBR of some node N_1 to its nearest data point o contained in the MBR of another node N_2 ($minExistDNN$).*

We use $minExistDNN(N_C, N_P)$ to denote the minimum upper bound of the distance between a point o within the MBR of N_C, $N_C.mbr$, and its nearest potential location $p \in sub(N_P)$. Formally,

$$minExistDNN(N_C, N_P) = \max\{minMaxDist(o, N_P)|o \in N_C.mbr\}.$$

Xia et al. (2005) propose a method to efficiently compute $minExistDNN(N_C, N_P)$. As shown in Figure 10,

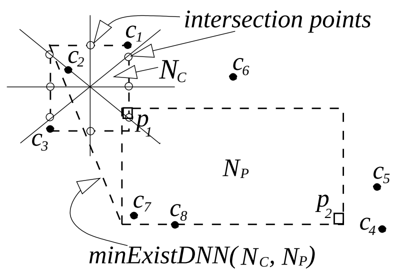


Figure 10: Example of $minExistDNN$

if we draw the four perpendicular bisectors of $N_C.mbr$'s edges and diagonals, they intersect $N_C.mbr$ at eight points. These points are called the *intersection points* (Xia et al. 2005) of N_C. Xia et al. prove that for any point $o \in N_C.mbr$, there is a corner point or an intersection point o' of $N_C.mbr$, such that $minMaxDist(o', N_P) \geq minMaxDist(o, N_P)$. As a result, the computation of $minExistDNN(N_C, N_P)$ requires checking at most twelve points.

Now we have the following theorem to define and prove the lower bound $mindr$.

Theorem 3. *The following expression defines a lower bound for the dr value of all data points indexed in the subtree rooted at a node N_P of R_P.*

$$\max\{maxFDist(N_C) - minExistDNN(N_C, N_P)|N_C \in IN(N_P)\}.$$

Proof. The definition of $minExistDNN$ implies:

$$\begin{aligned}
&\forall c \in sub(N_C)\exists p \in sub(N_P): \\
&\quad minExistDNN(N_C, N_P) \geq dist(c, p) \\
&\Rightarrow \forall c \in sub(N_C)\exists p \in sub(N_P): \\
&\quad -minExistDNN(N_C, N_P) \leq -dist(c, p), \\
&\text{Also, } \exists c \in sub(N_C): \\
&\quad maxFDist(N_C) = dnn(c, F) \\
&\text{Hence, } \exists c \in sub(N_C)\exists p \in sub(N_P): \\
&\quad maxFDist(N_C) - minExistDNN(N_C, N_P) \\
&\quad \leq dnn(c, F) - dist(c, p) \leq dr(p).
\end{aligned}$$

Therefore, the lower bound holds. □

4.5 Discussion

Let us revisit the BB method. Its core idea is that a depth-first traversal is performed on R_P while a global lower bound pd is used to prune the subtrees. The pruning distance pd is updated once $mindr$ of some node or dr of some potential location is found to be larger than it. The pruning power relies on the fast increase of pd. The reason why we do not use a best-first traversal is that $mindr$ is rather small. Thus the main reason for pd to increase is the update of newly found larger dr value. If we use a best-first traversal, dr value will not be found until the traversal reaches the leaf nodes of both trees, which means almost all non-leaf nodes may end up staying in the active page list waiting to be pruned. The space requirement for this process is too high. Hence, we opt to use the depth-first traversal.

5 Cost Analysis

In this section, we analyze the I/O cost and CPU cost of the BB method and compare them with those of the SS method.

We first introduce the notation and equations used in the analysis. We assume an R-tree node has the size of a disk block. Let C_m be the maximum number of entries in a disk block $(i.e., C_m =$ block size / size of a data entry$)$ and C_e be the effective capacity of an R-tree node, i.e., the average number of entries in an R-tree node. Then the average height of an R-tree, h, is computed as $\lceil \log_{C_e} n \rceil$, where n is the cardinality of the dataset (we denote the cardinalities of C, F and P by n_c, n_f and n_p, respectively). The expected number of nodes in an R-tree is the total number of nodes in all tree levels (leaf nodes being level 1 and the root node being level h), which is $\sum_{i=1}^{h} \frac{n}{C_e^i} = n\left(\frac{1}{C_e} + \frac{1}{C_e^2} + \cdots + \frac{1}{C_e^h}\right) = \frac{n}{C_e - 1}(1 - \frac{1}{C_e^h}) \approx \frac{n}{C_e - 1}$.

I/O cost: For the SS method, the data points are retrieved in blocks from the disk, and the I/O cost is $IO_s = \frac{n_p}{C_m}\frac{n_c}{C_m} = \frac{n_p n_c}{C_m^2}$. For the BB method, the I/O cost depends on the number of R-tree nodes accessed. In the method, R_P is traversed in a depth-first order and for every node N_P of R_P, we need to retrieve the nodes in the client R-tree that satisfies certain conditions with N_P. In the worst case, every node of R_P is traversed, and for every node of R_P, the whole client R-tree is traversed. Therefore, the worst-case I/O cost is: $\frac{n_p}{C_e - 1}\frac{n_c}{C_e - 1} = \frac{n_p n_c}{(C_e - 1)^2}$. While this worst-case I/O cost is worse than the I/O cost of the SS method because $C_e < C_m$, in practice, many nodes of the R-trees are pruned during the traversal. We quantify the percentages of the pruned nodes in R_P and R_C^b as the pruning power, and denote them by w_p and w_c, respectively. Then we have the the average I/O cost of the BB method, $IO_b = (1 - w_p)(1 - w_c)\frac{n_p n_c}{(C_e - 1)^2}$. The superiority of the BB method over the SS method lies in w_p and w_c. In our performance study, we will show that the pruning techniques used in the BB method are effective and IO_b is constantly much less than IO_s.

CPU cost: The CPU cost can be considered as the product of the CPU cost per disk block (R-tree node) multiplied by the number of disk blocks (R-tree nodes) accessed. The I/O cost analysis provides the number of nodes accessed. The CPU cost per disk block (R-tree node), typically involves distance metric computations. For every pair of disk blocks accessed, the SS method computes $dist(c, p)$ for every pair of client c and potential location p. So there are C_m^2 distance metric computations. For every pair of R-tree nodes (N_P, N_C), the BB method only computes the values of several distance metrics to determine whether N_C should be put in $IN(N_P)$ for further process. Thus, the BB method has a much smaller number of distance metric computations to process a pair of R-tree nodes than that of the SS method to process a pair of disk blocks. We have also shown that on average, the BB method has a much smaller I/O cost than that of the SS method. Therefore, the CPU cost of the BB method is much smaller than that of the SS method.

6 A Performance Study

In this section, we report the results of our performance study. The experimental setting is presented in Section 6.1. To evaluate the performance of the proposed method under different environments, we conduct experiments on both synthetic and real datasets. Specifically, Section 6.2 presents experiments on datasets with uniform distribution varying the size of the datasets. Section 6.3 presents experiments on datasets with Gaussian distribution varying the variance of the distribution function. Section 6.4 presents experiments on datasets with Zipfian distribution varying the alpha value of the distribution function. Section 6.5 presents experiments on real datasets.

Table 2: Parameters and Their Settings

Parameter	Setting
Data distribution	**Uniform**, Gaussian, Zipfian
Client set size	10K, 50K, **100K**, 500K, 1000K
Existing facility set size	0.1K, 0.5K, 1K, **5K**, 10K
Potential location set size	1K, **5K**, 10K, 50K, 100K
μ (Gaussian distribution)	**0**
σ^2 (Gaussian distribution)	0.125, 0.25, 0,5, **1**, 2
N (Zipfian distribution)	**1000**
α (Zipfian distribution)	0.1, 0.3, 0.6, **0.9**, 1.2

6.1 Experimental Setup

All the experiments were conducted on a personal computer with 3GB RAM and 2.66GHz Intel(R) Core(TM)2 Quad CPU. The disk page size is 4K bytes, and no buffer is used. We measure the running time and the number of I/Os.

We conduct experiments on synthetic and real datasets. Synthetic datasets are generated with a space domain of 1000×1000. The dataset cardinalities range from 0.1K to 1000K. Three types of datasets are used: (i) *Uniform datasets*, where data points are generated randomly according to a uniform distribution; (ii) *Gaussian datasets*, where data points follow the Gaussian distribution; (iii) *Zipfian datasets*, where data points follow the Zipfian distribution. The parameters used in the experiments on synthetic datasets are summarized in Table 2, where values in bold denote default values.

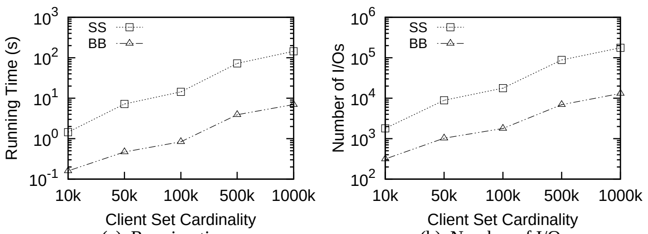
We adopt two groups of real datasets provided by Digital Chart of the World (RtreePortal 2011), which contain the points of populated places and cultural landmarks in the US and in North America. We name them as the US group and the NA group, respectively. For each group of datasets, the populated places are used as the client set C. The cultural landmark dataset is divided into two datasets. Half of the cultural landmarks are chosen randomly to form the existing facility set F, and the remaining are used as the potential location set P. For the US group, the cardinalities of C, F, P are 15206, 3008 and 3009, respectively, while those for the NA group are 24493, 4601 and 4602.

We use the R-tree (Guttman 1984) and its proposed variant as the underlying access methods.

6.2 Experiments on Uniform Datasets

The following experiments focus on the effect of dataset cardinalities. We vary the sizes of C, F and P independently.

6.2.1 Varying the Number of Clients



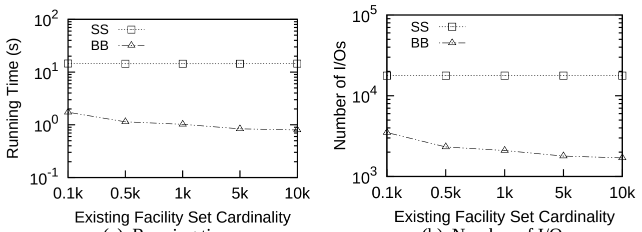
(a) Running time (b) Number of I/Os

Figure 11: The effect of client set cardinality

The results for the experiments that vary the number of clients are shown in Figure 11. From this figure, we can see that the BB method outperforms the SS method by almost ten times in terms of both the running time and the number of I/Os. This is because of the pruning techniques used by the BB method to reduce the search space for the query answer, and this result confirms our cost analysis, where the average cost of the BB method is shown to be much smaller than that of the SS method.

We also see that even with a small set of clients (10K), it takes the SS method seconds to process the query. Considering the capability of human perception, 0.1 seconds may be a preferable choice for processing a query (Morse 1996). Then the SS method is far inferior and is unable to produce the query answer in time, especially for the urban development simulations and the MMOG applications. As for the BB method, it computes the query answer in less than 0.1 seconds for the 10K dataset. Even for a very large dataset (1000K), it computes the query answer within seconds. With some upgrades in hardware, it is still realistic for the BB method to produce the query answer in time.

6.2.2 Varying the Number of Existing Facilities



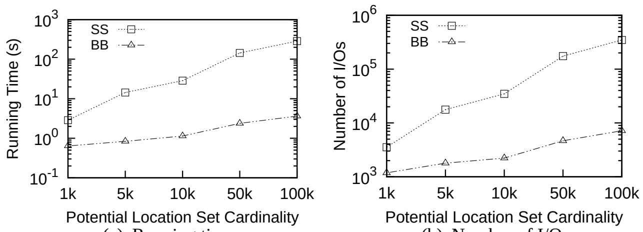
(a) Running time (b) Number of I/Os

Figure 12: The effect of existing facility set cardinality

The results of the experiments varying the number of existing facilities are shown in Figure 12. Again, in this figure, the BB method shows much better performance than the SS method in terms of both the running time and the number of I/Os because of the pruning techniques used to reduce the search space.

Another observation is that an increase in the number of facilities yields a drop in both the running time and the number of I/Os for the BB method. The reason is that on average the more the facilities, the shorter the nearest facility distances for the clients. In other words, $dnn(c, F)$ decreases with the increase of the number of existing facilities. As a result, $maxFDist(N_C)$ decreases and the pruning power of the BB method to prune nodes in R_C^b is enhanced. Therefore, the number of I/Os and running time are reduced. SS is not affected due to its lack of pruning capability and it does not access the set of F (it accesses F for $dnn(c, F)$ computation, which is assumed to be precomputed).

6.2.3 Varying the Number of Potential Locations



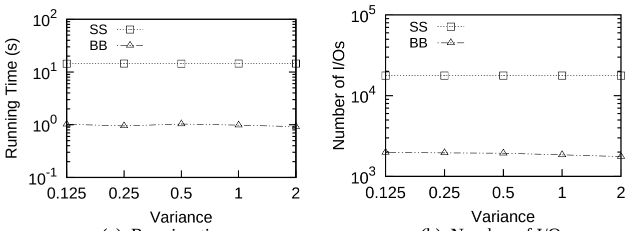
(a) Running time (b) Number of I/Os

Figure 13: The effect of potential location set cardinality

Results of the experiments that vary the number of potential locations are shown in Figure 13. The BB method still shows high efficiency in these experiments.

We observe that, generally, the growth in the number of potential locations has the similar effect on the running time and the number of I/Os as increasing the number of clients. We also notice that, as the number of potential locations increases, the running time and the number of I/Os of the BB method increase much slower than those of the SS method do (please note the logarithmic scale). This is because when the number of potential locations becomes larger, the height of R_P increases and every time a non-leaf node in R_P is pruned, more potential locations are pruned. When the number of potential locations n_p becomes very large (i.e. $n_p \geq 10$K), the proposed pruning techniques function even better and the advantage of the BB method becomes significant.

6.3 Experiments on Gaussian Datasets



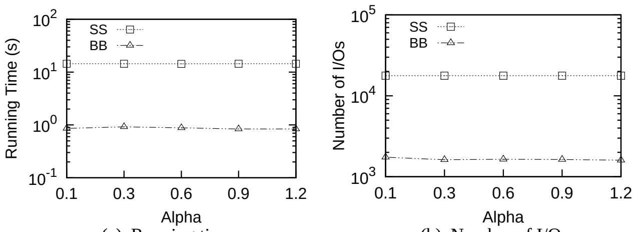
(a) Running time (b) Number of I/Os

Figure 14: The effect of σ^2 in Gaussian distribution

In the following experiments, we vary the distribution of the datasets.

Figure 14 shows the results of experiments conducted on the Gaussian datasets where we vary the value of σ^2. For the Gaussian datasets, varying σ^2 means varying the degree of the inclination for the data points to cluster at the central area of the distribution. Increasing σ^2 leads to less dense data points at the center. We observe that, compared with varying dataset cardinalities, varying σ^2 does not affect much of the algorithm performance. The BB method still outperforms the SS method in terms of both the running time and the number of I/Os, which confirms the results of our cost analysis.

6.4 Experiments on Zipfian Datasets

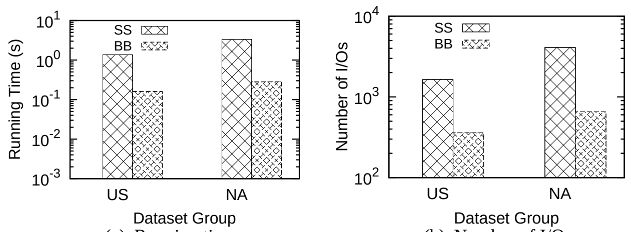


(a) Running time (b) Number of I/Os

Figure 15: The effect of α in Zipfian distribution

We vary the value of α in the experiments conducted on the Zipfian datasets and the results are shown in Figure 15. Like the Gaussian datasets, we notice that the value of α does not affect much of the algorithm performance. We also notice that the resultant running time and number of I/Os are similar to those of the experiments conducted on the Gaussian datasets. We further compare these results with those of the experiments conducted on the uniform datasets with the same dataset cardinalities, and find that the differences among them are small, too. Thus, we can conclude that the effect of different distributions on the proposed method is trivial.

6.5 Experiments on Real Datasets



(a) Running time (b) Number of I/Os

Figure 16: Performance comparison on real datasets

The experimental results on real datasets are shown in Figure 16. The comparative performance of the methods is similar to that of experiments conducted on the synthetic datasets. The BB method still outperforms the SS methods significantly for both US and NA datasets.

Overall, we see that the BB method outperforms the SS methods constantly because of the pruning techniques used to reduce the search space for the query answer. When the dataset cardinalities become large, the advantage of the BB method becomes more significant. These results agree with our cost analysis.

7 Conclusions

We formulated the min-dist location selection query and conducted a comprehensive study on processing this query. We first analyzed the basic properties of this query type and proposed a naive algorithm (SS) to process the query. However, the SS algorithm is inefficient due to repeated scanning on datasets. Motivated by this, we explored geometric properties of spatial data objects, and proposed techniques to prune the search space. This resulted in a branch and bound algorithm (BB). We provided a detailed comparative cost analysis for the BB method and performed extensive experiments to evaluate the empirical performance of the method. The experimental results show that the BB method constantly outperforms the SS method, and when the dataset cardinalities become large, the advantage of the BB method becomes more significant.

References

Achtert, E., Kriegel, H.-P., Kröger, P., Renz, M. & Züfle, A. (2009), Reverse k-nearest neighbor search in dynamic and general metric databases, *in* 'EDBT'.

ArcGIS (2011), 'http://www.esri.com/'.

Cabello, S., Díaz-Báñez, J. M., Langerman, S., Seara, C. & Ventura, I. (2005), Reverse facility location problems., *in* 'CCCG'.

Cheema, M. A., Lin, X., Wang, W., Zhang, W. & Pei, J. (2010), 'Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data', *TKDE* **22**(4), 550–564.

Cheema, M. A., Lin, X., Zhang, W. & Zhang, Y. (2011), Influence Zone : Efficiently Processing Reverse k Nearest Neighbors Queries, *in* 'ICDE'.

Du, Y., Zhang, D. & Xia, T. (2005), The optimal-location query., *in* 'SSTD'.

Gao, Y., Zheng, B., Chen, G. & Li, Q. (2009), 'Optimal-location-selection query processing in spatial databases', *TKDE* **21**, 1162–1177.

Guttman, A. (1984), R-trees: A dynamic index structure for spatial searching., *in* 'SIGMOD'.

Hjaltason, G. R. & Samet, H. (1995), Ranking in spatial databases, *in* 'SSD'.

Huang, J., Wen, Z., Pathan, M., Taylor, K. & Zhang, R. (2011), Ranking Locations for Facility Selection based on Potential Influences, *in* 'the 37th Annual Conference of the IEEE Industrial Electronics Society'.

Huang, J., Wen, Z., Qi, J., Zhang, R., Chen, J. & He, Z. (2011), Top-k Most Influential Location Selection, *in* 'CIKM'.

Korn, F. & Muthukrishnan, S. (2000), Influence sets based on reverse nearest neighbor queries., *in* 'SIGMOD'.

Morse, K. L. (1996), Interest management in large-scale distributed simulations, Technical Report ICS-TR-96-27.

Mouratidis, K., Papadias, D. & Papadimitriou, S. (2005), Medoid queries in large spatial databases., *in* 'SSTD'.

Roussopoulos, N., Kelley, S. & Vincent, F. (1995), Nearest neighbor queries, *in* 'SIGMOD'.

RtreePortal (2011), 'http://www.rtreeportal.org'.

Stanoi, I., Riedewald, M., Agrawal, D. & Abbadi, A. E. (2001), Discovery of influence sets in frequently updated databases, *in* 'VLDB'.

Tao, Y., Yiu, M. L. & Mamoulis, N. (2006), 'Reverse Nearest Neighbor Search in Metric Spaces', *TKDE* **18**(9), 1239–1252.

Wong, R. C.-W., Özsu, M. T., Yu, P. S., Fu, A. W.-C. & Liu, L. (2009), 'Efficient method for maximizing bichromatic reverse nearest neighbor.', *PVLDB* **2**, 1126–1137.

Wu, W., Yang, F., Chan, C. Y. & Tan, K.-L. (2008), Continuous Reverse k-Nearest-Neighbor Monitoring, *in* 'The Ninth International Conference on Mobile Data Management'.

Xia, T., Zhang, D., Kanoulas, E. & Du, Y. (2005), On computing top-t most influential spatial sites., *in* 'VLDB'.

Yang, C. & Lin, K.-I. (2001), An index structure for efficient reverse nearest neighbor queries, *in* 'ICDE'.

Yu, C., Zhang, R., Huang, Y. & Xiong, H. (2010), 'High-dimensional knn joins with incremental updates', *Geoinformatica* **14**, 55–82.

Zhang, D., Du, Y., Xia, T. & Tao, Y. (2006), Progressive computation of the min-dist optimal-location query, *in* 'VLDB'.