The Min-dist Location Selection Query

Jianzhong Qi [†1], Rui Zhang [†2], Lars Kulik [†3], Dan Lin [‡4] Yuan Xue [†5]

[†]*University of Melbourne, Victoria, Australia*
[1,2]{jiqi,rui}@csse.unimelb.edu.au, [3,5]{lkulik,yuaxue}@unimelb.edu.au

[‡]*Missouri University of Science and Technology, Missouri, USA*
[4]lindan@mst.edu

***Abstract*— We propose and study a new type of location optimization problem: given a set of clients and a set of existing facilities, we select 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. We call this problem the min-dist location selection problem, which has a wide range of applications in urban development simulation, massively multiplayer online games, and decision support systems. We explore two common approaches to location optimization problems and propose methods based on those approaches for solving this new problem. However, those methods either need to maintain an extra index or fall short in efficiency. To address their drawbacks, we propose a novel method (named MND), which has very close performance to the fastest method but does not need an extra index. We provide a detailed comparative cost analysis on the various algorithms. We also perform extensive experiments to evaluate their empirical performance and validate the efficiency of the MND method.**

I. Introduction

Location optimization is an important problem for spatial decision support systems. A number of studies [1], [2], [3] proposed solutions to various instances of such problems. In this paper, we consider a new location optimization problem that cannot be efficiently solved by existing techniques. The problem is motivated by the following applications.

In urban development simulation, urban planners need to consider the influence of public infrastructure or business centers on the residents. Very often they need to select a location for establishing a new facility (e.g., fire hydrant, public phone booth, hospital, bus stop, etc.) and a commonly used criteria is to select the location that can minimize the average distance between a resident and her nearest facility so that people can access the facilities in the shortest time.

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 team, which has moved on to another region. It is 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 completing quests rather than walking.

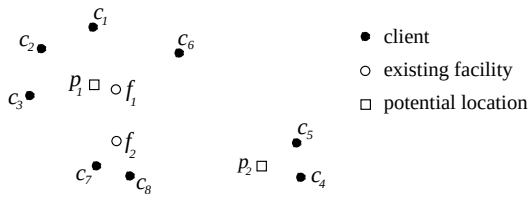


Fig. 1. An example for the problem

The example in Fig. 1 illustrates the problem: $\{c_1, c_2, ..., c_8\}$ 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.

Besides the above described applications, many organizations and businesses face similar decision making problems (e.g., McDonald's needs to add new restaurants; a wireless service provider needs to set up new hotspots). This paper studies how to efficiently select a potential location for a new facility, so that the average distance between a client and her nearest facility is minimized. We call it the *min-dist location selection problem*. In the aforementioned applications, the min-dist location selection is usually performed frequently. Therefore, we formulate the problem as the following query.

A. 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*. 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 $p_i \in P$ for a new facility to be established at, so that $\forall p_j \in$

P and $p_j \neq p_i$:

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

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.

Although an existing commercial software [4] can solve several simpler location optimization problems, none can solve the min-dist location selection problem (see Section II).

B. 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 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. KNN-join algorithms (e.g., [5]) 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.

Algorithm 1: SS(C, P)

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

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.

C. 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 problem and analyze its basic properties.
- We explore two common approaches to location optimization problems and propose methods based on them for solving this new problem, the quasi-Voronoi cell method and the nearest facility circle method.
- As methods based on the common approaches either need to maintain an extra index or fall short in efficiency, we propose a method called MND, which uses a single value to describe a region that encloses the nearest existing facilities of a group of clients, so that the search of influenced clients for a potential location can be done groupwise. This results in an algorithm that has very close performance to the fastest of the previous algorithms without the need for an extra index.
- We provide a thorough cost analysis of all methods.
- We conduct extensive experiments to evaluate the empirical performance of all methods. The results validate the efficiency of the MND method.

Section II reviews related work. Section III discusses the basic properties of the problem and presents a solution framework. Sections IV, V and VI present the quasi-Voronoi cell method, the nearest facility circle method and the MND method, respectively. Section VII analyzes the cost of the methods. Section VIII presents the experimental results and Section IX concludes the paper.

II. Related Work

Location optimization problems are mostly characterized by optimization functions, based on which they can be classified into two categories: *max-inf* problems and *min-dist* problems. Both categories are closely related to *nearest neighbor (NN)* search and *reverse nearest neighbor (RNN)* search. Therefore, we first review studies of NN search and RNN search, and then review studies of max-inf problems and min-dist problems.

NN search: Given a set of objects S and a query object q, the NN search returns q's nearest objects in S. Two popular NN search algorithms are depth-first [6] and best-first [7]. The best-first algorithm can retrieve the nearest neighbors incrementally in order of their distances to the query point.

RNN search: Korn and Muthukrishnan [8] first propose the RNN query and define the RNNs of an object o to be the objects whose respective NN is o. They propose to use an R-tree variant, named the *RNN-tree*, in addition to the original R-tree that maintains the data points to answer RNN queries. In an RNN-tree, the data entries are stored in the form of NN circles. An NN circle of a point o is a circle centered at o with the distance between o and its NN as the radius. The bounding boxes of these NN circles are indexed in the RNN-tree. Using this tree, an RNN query is answered with the data points whose NN circles enclose the query point. To avoid maintaining the RNN-tree, Yang and Lin [9] propose the *RdNN-tree*, which effectively combines the original R-tree and the RNN-tree. While these methods require the precomputation of the distance between an object and its NN, Stanoi et al. [10] propose to process the RNN queries without precomputation. For a query point q, their method dynamically constructs a Voronoi cell that encloses q

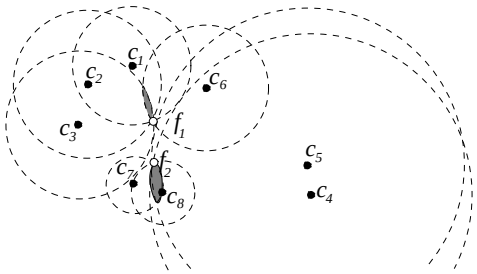


Fig. 2. Example of a max-inf problem [1]

and contains all its possible RNNs. Only nodes intersecting the Voronoi cell have to be accessed to check for q's RNNs.

There are studies on RNN query variants under different settings. For example, the *reverse k nearest neighbor (RkNN)* query finds objects whose k nearest neighbors include the query object. Wu et al. [11] study the RkNN query on continuously moving objects, which correlates two sets of moving objects according to their closeness. The continuous join query on extended moving objects [12], [13] also correlates multiple sets, but it focuses on intersecting objects with a time-constraint technique rather than closeness. While these approached work well for a single R(k)NN query, they are not tailored for computing RNNs for large amount of objects at the same time, which is a key difficulty in our study.

Max-inf problems: Max-inf problems maximize the *"influence"* of a facility, where influence is typically defined as the number of clients who are the RNNs of the facility. Cabello et al. [1] propose the *MAXCOV* problem that finds a location for a new facility to maximize its influence. They introduce the *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 existing nearest facility. Since only a facility established within the NLC of c can be a new nearest facility of c, to find the solution for the MAXCOV problem is to find the regions that are enclosed by the largest number of NLCs. Fig. 2 shows an example, where the gray regions are the problem solution because each of them is covered by four NLCs while no region is covered by more than four NLCs.

Xia et al. [2] propose the top-t most influential sites problem. They use a branch and bound method to find top-t facilities in F with the largest *influence* within a continuous spatial region Q, where the influence of a facility is defined as the total weight of its RNNs. Du et al. [14] also work on the max-inf problem with weighted clients and a continuous region Q. Using L_1 as the distance metric, this problem finds a location for a new facility so that its influence is maximized. Gao et al. [15] define a facility location problem that finds a location p outside a region Q (instead of inside a region) for a new facility so that its *"optimality"* is maximized. Here, the optimality of p is defined as a function of the number of clients in Q whose distance to p is within a certain threshold d_c (attracted by p). Intuitively, the more clients p attracts, the greater its optimality. A more recent study [16] selects top-k candidate locations with the largest influence values for a new facility. These studies differ from ours in optimization functions and other settings, and do not apply.

Min-dist problems: Zhang et al. [3] propose the min-dist optimal-location problem. Given a client set C, an existing facility set F and a region Q, it finds points within 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. Fig. 3 gives an example, where pt may

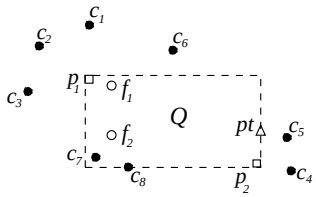


Fig. 3. Example of a min-dist problem [3]

be one of the points in the answer set and it is not the solution p_2 to our problem. To solve the problem, Zhang et al. [3] propose a method that first identifies a set L of candidate locations from Q and then divides L progressively until the answer set is found.

Zhang et al.'s problem definition [3] has the same min-dist optimization function as ours, but our problem definition has an additional set P, the potential locations given as candidates for selection. In many real applications, we can only choose from some candidate locations, e.g., McDonald's may only open a new restaurant at a place for rent or sale rather than anywhere in a region. The main idea of Zhang et al.'s solution is the fast identification of a small set L of candidate locations from Q. However, the candidate locations in L could be any point from Q, which may not even contain a potential location from our potential location set P. This means that in the general case their approach cannot provide a correct answer to our problem, and thus is not applicable.

Related commercial software: As mentioned in Section I-A, an existing commercial software [4] 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 points in the set of existing facilities F. Therefore, this software does not solve our problem.

TABLE I
THE LOCATION OPTIMIZATION PROBLEMS

Problem	Optim. Function	Solution Space	Distance Function	Datasets
[1]	Max-inf	Continuous	L_2	C, F
[2]	Max-inf	Discrete	L_2	C, F
[14]	Max-inf	Continuous	L_1	C, F
[15]	Max-inf	Discrete	L_2	C, P
[16]	Max-inf	Discrete	L_2	C, F, P
[3]	Min-dist	Continuous	L_1	C, F
[4]	Min-dist	Discrete	L_2	C, P
Proposed	Min-dist	Discrete	L_2	C, F, P

Summary: Table I summarizes the differences between

our problem and previously studied location optimization problems. Most previous problems are max-inf problems and differ from our problem in optimization functions. For the min-dist problems, they have the same optimization function as our problem does, but their problem settings are different. For example, Zhang et al. [3] consider finding locations for the new facility from a continuous region. As discussed in the second paragraph of the related min-dist problems, their problem does not choose from a set of given candidate locations, which does not apply to the requirements of our applications.

III. Basic Properties and a Solution Framework

The min-dist location selection query can be redefined in a form that reduces the search space. This section provides such a redefinition and a solution framework based on it. Next, we start with some basic properties of the problem needed for the redefinition. Table II summarizes frequently used symbols.

TABLE II
Frequently Used Symbols

Symbols	Explanation
o	A 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

A. Basic Properties

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 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 clients that will reduce their NFD if a new facility is established at p. For example, in Fig. 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 $p_i \in P$, so that $\forall p_j \in P$ and $p_j \neq p_i$: $dr(p_j) \leq dr(p_i)$.

B. A Solution Framework

Definition 2 provides a framework for solving the min-dist location selection problem with the following two steps:

1) Identify $IS(p)$;
2) Compute $dr(p)$ and find the potential location with the largest $dr(p)$.

Since the cardinality of $IS(p)$ is usually much smaller than that of C, we do not have to access the whole client dataset for every potential location p. Thus, the above framework has a great potential to improve performance. All methods presented in this paper will follow this framework. The key issues are: (i) how to efficiently identify $IS(p)$ and (ii) how to prune more potential locations from consideration. We will see that in all methods $dnn(c,F)$ of every client is used many times in both steps of the framework. Computing $dnn(c,F)$ on-the-fly will repeatedly access the datasets of the clients and the existing facilities, which will incur significant costs. Therefore, we precompute $dnn(c,F)$ for every client and store it with the client's record for all methods (including the SS method).

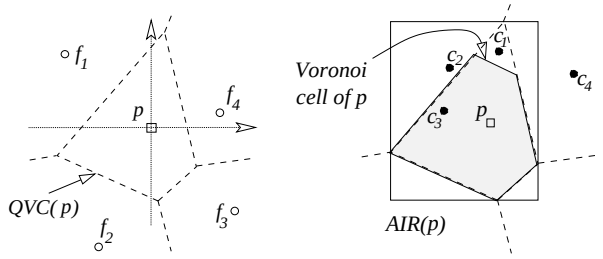
In the next two sections, we explore two common approaches to location optimization problems and propose methods based on those approaches for solving the min-dist location selection query under the above framework. When a spatial index is used for a method, we assume an R-tree [17], although any hierarchical spatial index could be used.

IV. Quasi-Voronoi Cell Method

In this section, we propose a so-called "quasi-Voronoi cell" (QVC) method. For any potential location p, the *Voronoi cell* of p on the set $F\cup p$ is a region V that satisfies that for any point $p' \in F\cup p$, $p' \neq p$, and for any point $o \in V$, $dist(p,o) \leq dist(p',o)$ [18]. It is guaranteed that the Voronoi cell of p encloses all and only the clients in $IS(p)$. We can use the Voronoi cell to quickly identify $IS(p)$. However, computing the Voronoi cell of p is an expensive operation itself. Interestingly, this algorithm only needs to identify a superset of $IS(p)$ instead of the exact $IS(p)$. Stanoi et al. [10] show a relatively straightforward way to compute a region that encloses the Voronoi cell and this region is a good approximation of the Voronoi cell. We call this region the *quasi-Voronoi cell* (QVC). First, we find a superset of $IS(p)$ through the QVC of p. Then, we can use the precomputed NFD to quickly identify the exact $IS(p)$. Finally, we compute $dr(p)$ and compare it for all potential locations. Next, we give details of constructing QVC and the algorithms.

The QVC of a potential location p is formed as follows. In the coordinate system with the origin at p and the two axes parallel with the original axes, find the nearest facility to p in each of the four quadrants and let these nearest facilities be f_1, f_2, f_3 and f_4 as shown in Fig. 4(a). Draw the bisector between each f_i ($i = 1,2,3,4$) and p, and the four bisectors form a polygon. This polygon is the QVC of p, denoted as $QVC(p)$. Stanoi et al. [10] prove that $QVC(p)$ encloses the Voronoi cell of p. To find the NN in each quadrant, we use the best-first algorithm [7] to retrieve the NNs until each quadrant

has one. Since this algorithm is based on a spatial index, we use an R-tree to index the facilities, denoted as R_F.



(a) Quasi-Voronoi cell (b) Approximate influence region

Fig. 4. Examples of the QVC method

To facilitate finding clients in $QVC(p)$, we index C using an R-tree and denote this R-tree by R_C. We perform a window query on R_C with the query range being the *minimum bounding rectangle (MBR)* of $QVC(p)$. We call the MBR of $QVC(p)$ the *approximate influence region* of p and denote it by $AIR(p)$ (Fig. 4(b)). The window query finds all clients in $AIR(p)$. For each client c in $AIR(p)$, we compare $dist(p,c)$ with $dnn(c,F)$, which has been precomputed.

If $dist(p,c) \geq dnn(c,F)$, then we know that c is not in $IS(p)$. Otherwise, c is in $IS(p)$. Since we can identify all the clients in $IS(p)$, we can then compute $dr(p)$. We compute $dr(p)$ for every potential location and the one with the largest distance reduction is the answer. The QVC method is summarized in Algorithms 2 and 3.

Algorithm 2: QVC(R_C, R_F, F_P)

```
1  optLoc ← NULL;
2  while not EndOfFile( F_P ) do
3      B_P ← ReadBlock( F_P );
4      S_p ← ∅;
5      for p ∈ B_P do
6          Contruct QVC(p) from R_F;
7          Contruct AIR(p), stores it as p.mbr;
8          if p.mbr intersects R_C.rootnode.mbr then
9              S_p ← S_p ∪ p;
10     WQ( R_C.rootnode, S_p, optLoc );
11 output optLoc;
```

Algorithm 3: WQ(N_C, S_p, $optLoc$)

```
1  if N_C is a leaf node then
2      for p ∈ S_p do
3          for e_c ∈ N_C, dist(p,e_c) < e_c.dnn(c,F) do
4              p.dr ← p.dr + e_c.dnn(c,F) − dist(p,e_c);
5          if optLoc = NULL or p.dr > optLoc.dr then
6              optLoc ← p;
7  else
8      for e_c ∈ N_C do
9          S'_p ← ∅;
10         for p ∈ S_p, p.mbr intersects e_c.mbr do
11             S'_p ← S'_p ∪ p;
12         WQ(e_c.childnode, S'_p, optLoc);
```

V. Nearest Facility Circle Method

In this section, we propose a method that exploits the *nearest facility circle (NFC)*, and we call it the NFC method. The nearest facility circle of a client c, denoted by $NFC(c)$, is a circle centered at c with the radius being $dnn(c,F)$. It can be observed that for a potential location p, $c \in IS(p)$ if and only if p is inside $NFC(c)$. An example is shown in Fig. 5, where

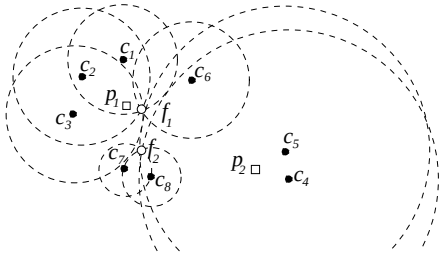


Fig. 5. Example of NFCs

p_1 is in the NFCs of c_1, c_2 and c_3, and p_2 is in the NFCs of c_4 and c_5. Thus, $IS(p_1) = \{c_1, c_2, c_3\}$ and $IS(p_2) = \{c_4, c_5\}$. Therefore, we only need to check which NFCs enclose p to identify the clients in $IS(p)$. Motivated by this observation, we build an RNN-tree [8], denoted as R_C^n, to index the NFCs of all clients. As discussed in the related work, R_C^n is basically an R-tree that indexes the MBRs of the NFCs of the clients. It can be built based on R_C and maintained in accordance to the updates of R_C. Note that there is no extra computation to get the clients' NFCs, since $dnn(c,F)$ has been precomputed for all clients and stored in R_C. Besides having an RNN-tree to index the NFCs, this method also uses an R-tree to index the potential location set P, denoted as R_P. Then for every potential location p, we can use R_C^n to quickly identify all NFCs that enclose p, which is essentially a point query on an R-tree. We need to do this for all the potential locations indexed in R_P, which makes the process a spatial join between

Algorithm 4: NFC(N_P, N_C^n, $optLoc$)

```
1  if N_P and N_C^n are non-leaf nodes then
2      for (e_p, e_c^n) ∈ N_P × N_C^n, e_p.mbr intersects e_c^n.mbr do
3          NFC(e_p.childnode, e_c^n.childnode, optLoc);
4  else if N_P is a leaf node and N_C^n is a non-leaf node then
5      for e_c^n ∈ N_C^n, e_c^n.mbr intersects N_P.mbr do
6          NFC(N_P, e_c^n.childnode, optLoc);
7  else if N_P is a non-leaf node and N_C^n is a leaf node then
8      for e_p ∈ N_P, e_p.mbr intersects N_C^n.mbr do
9          NFC(e_p.childnode, N_C^n, optLoc);
10 else
11     for e_p ∈ N_P, e_p intersects N_C^n.mbr do
12         for e_c^n ∈ N_C^n,
           dist(centerOf(e_c^n.mbr), e_p) < ½(edgeLength(e_c^n.mbr))
           do
13             e_p.dr ← e_p.dr + ½(edgeLength(e_c^n.mbr)) −
               dist(centerOf(e_c^n.mbr), e_p) ;
14         if e_p.dr > optLoc.dr or optLoc = NULL then
15             optLoc ← e_p;
```

P and the set of all NFCs of C. The spatial join operation finds out all intersected pairs between two sets of objects. In our case, when P is a set of points, the spatial join returns for every p, the set of NFCs that enclose p. Then we can identify $IS(p)$ using the clients corresponding to the NFCs that enclose p and compute $dr(p)$ for every p. We use a standard R-tree based join algorithm [19] to join R_P and R_C^n, which results in the NFC algorithm, as summarized in Algorithm 4. Note that in this algorithm, since the MBR of an entry e_c^n bounds $NFC(c)$ of e_c^n's corresponding client c, we can compute $dnn(c,F)$ as $\frac{1}{2}edgeLength(e_c^n.mbr)$, where $edgeLength()$ returns the length of an edge of a square MBR (lines 12, 13).

VI. Maximum NFC Distance Method

We have proposed two methods based on common approaches to location optimization problems. However, those methods both have some drawbacks. The QVC method needs to perform a kNN search to find a nearest facility in each quadrant for every potential location, which is expensive. The NFC method is simple and efficient, but needs to maintain an extra index, R_C^n. In dynamic environments, insertions and deletions on data occur frequently. Maintaining two indexes on the dataset C makes database management such as concurrency control more complicated and brings significant overheads. Therefore, having the extra index has been considered as a serious drawback in the solutions to other types of location optimization problems [10], [9], [20], [21]. We also view the extra index for the NFC method as a serious drawback.

In this section, we propose a novel method that is simple and efficient, but requires no extra index, so it overcomes the drawbacks of the QVC and NFC methods. This method still exploits the idea of NFCs. However, unlike the NFC method, which uses an MBR to bound the NFCs of all clients in a node of R_C and physically stores all these MBRs in a separate tree (R_C^n), this method uses just one value to describe a region that encloses the NFCs of all clients in a node and stores that value in the parent entry of the node of R_C. Therefore, this method avoids using another tree but achieves the same purpose. A challenge in this method is to define a value for delimiting a region that can enclose the NFCs of all clients in a node N_C of R_C *as tight as possible*.

We propose to use a value with respect to a node called the *maximum NFC distance (MND)*, denoted as $MND(N_C)$ for a node N_C. The intuition is that given the NFCs of the clients indexed by a node N_C, we find a point from these NFCs whose distance to the MBR of N_C is the largest. This largest distance defines $MND(N_C)$. If the distance between N_C and a node N_P in R_P (the R-tree on the set of potential locations) is larger than or equal to $MND(N_C)$, then for any potential location p in N_P, no client in $IS(p)$ is from $sub(N_C)$ since no point in the MBR of N_P will be enclosed by the NFC of any client in $sub(N_C)$, where $sub(N_C)$ denotes the set of clients contained in the subtree rooted at N_C. In what follows, we first formally define MND and then explain it in detail.

Given a leaf node N_C in R_C and the clients indexed in N_C, we find a client c_i indexed in N_C and a point o_i on the boundary of $NFC(c_i)$, so that for any other point o_j on the NFC of any client indexed in N_C, $minDist(o_i,N_C) \geq minDist(o_j,N_C)$, where $minDist(o,N)$ denotes the minimum distance between two objects (either points or MBRs). Then we define $MND(N_C)$ as $minDist(o_i,N_C)$. The metric $MND(N_C)$ delimits a rounded rectangular region such that for any point o on its boundary, $minDist(o,N_C) = MND(N_C)$ (cf. Fig. 6(a)). We call this region the *MND region* of N_C.

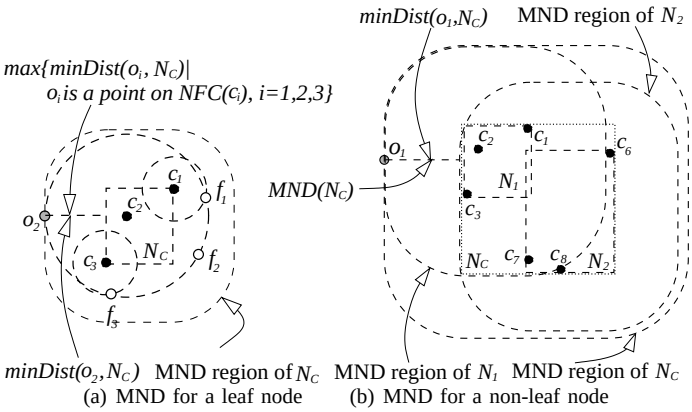


Fig. 6. Examples of MND regions

For non-leaf nodes, MND is defined recursively in a bottom-up manner. Given a non-leaf node N_C in R_C and the child nodes of N_C, we find a point o_i on the boundary of the MND region of a child node N_i, so that for any other point o_j on the boundary of the MND region of a child node N_j, $minDist(o_i,N_C) \geq minDist(o_j,N_C)$. Then we define $MND(N_C)$ as $minDist(o_i,N_C)$, and it delimits the MND region of N_C, the rounded rectangular region in Fig. 6(b).

The definition of the MND region of N_C guarantees that this region must be intersected by a node N_P in R_P if $sub(N_C) \cap IS(p) \neq \emptyset$, where p is a potential location in the subtree rooted at N_P. If this region is not intersected by N_P, then $sub(N_C) \cap IS(p) = \emptyset$ and we can discard the whole subtree of N_C when identifying $IS(p)$. This observation, formalized in Theorem 1, is the pruning strategy of the MND method.

Theorem 1: Let p be a potential location indexed in the subtree rooted at N_P, and $minDist(N_C,N_P)$ be the minimum distance between the MBRs of two nodes N_C and N_P. Then, $sub(N_C) \cap IS(p) = \emptyset$ if $minDist(N_C,N_P) \geq MND(N_C)$.

Proof: By definition, $minDist(N_C,N_P)$ is the minimum distance between a point in the MBR of N_C and a point in the MBR of N_P. For any point p indexed in the subtree rooted at N_P, p is enclosed by the MBR of N_P. Thus, $minDist(p,N_C) \geq minDist(N_C,N_P)$. If $minDist(N_C,N_P) \geq MND(N_C)$, then $minDist(p,N_C) \geq MND(N_C)$. According to the definition of $MND(N_C)$, p is not inside the NFC of any client indexed by N_C. Thus, $sub(N_C) \cap IS(p) = \emptyset$. ■

Theorem 1 suggests that we only need to check whether a node N_C's distance to N_P is less than $MND(N_C)$ to determine whether any client $c \in sub(N_C)$ is in $IS(p)$ for any potential location p enclosed by N_P. Like the other methods, we uses an R-tree to index the clients, but in addition, we store the MND value of a node N_C^m in its parent entry e_c^m, denoted as

$e_c^m.mnd$. To distinguish this R-tree from the normal R-tree on C, we denote it as R_C^m. The algorithm for processing the query mimics a spatial join on the two R-trees, R_C^m and R_P. We traverse the two trees simultaneously and compare every node from R_C^m with every node from R_P, starting from the roots. As we traverse down the tree, we compare a node pair (N_P, N_C^m) only if $minDist(N_P, N_C^m) < MND(N_C^m)$; this condition can be checked before retrieving N_C^m since $MND(N_C^m)$ is stored in the parent entry of N_C^m. When the traversal of the two trees finishes, all nodes that may contain points in $IS(p)$ are checked and hence we obtain $IS(p)$. Algorithm 5 details the steps.

Algorithm 5: MND(N_P, N_C^m, $optLoc$)

```
1  if NP and N_C^m are non-leaf nodes then
2      for (e_p, e_c^m) ∈ N_P × N_C^m, minDist(e_c^m, e_p) < e_c^m.mnd do
3          MND(e_p.childnode, e_c^m.childnode, optLoc);
4  else if NP is a leaf node and N_C^m is a non-leaf node then
5      for e_c^m ∈ N_C^m, minDist(e_c^m, NP) < e_c^m.mnd do
6          MND(NP, e_c^m.childnode, optLoc);
7  else if NP is a non-leaf node and N_C^m is a leaf node then
8      for e_p ∈ NP, minDist(N_C^m, e_p) < N_C^m.mnd do
9          MND(e_p.childnode, N_C^m, optLoc);
10 else
11     for (e_p, e_c^m) ∈ N_P × N_C^m, minDist(e_c^m, e_p) < e_c^m.mnd do
12         e_p.dr ← e_p.dr + e_c^m.dnn(c,F) − dist(e_c^m, e_p);
13     if e_p.dr > optLoc.dr or optLoc = NULL then
14         optLoc ← e_p;
```

A. Efficient computation of the Maximum NFC Distance

The definition of MND does not give an efficient way for its computation. According to the definition, MND can be computed straightforwardly as follows. Suppose N_C^m is a leaf (or non-leaf) node. We compute for every client c (or child node N) indexed by N_C^m the largest $minDist(o, N_C^m)$ value for a point o on the boundary of $NFC(c)$ (or MND region of N), denoted as $maxMinDist(c, N_C^m)$ (or $maxMinDist(N, N_C^m)$). Since the MND region of N_C^m should enclose the NFCs (or MND regions) of all N_C^m's children, $MND(N_C^m)$ must be the largest among all these children's $maxMinDist$ values, that is:

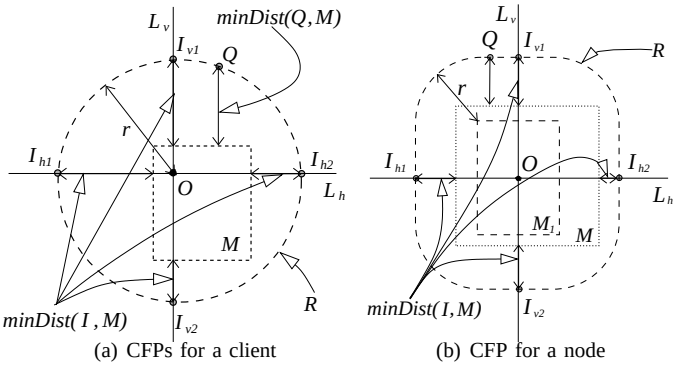
$$MND(N_C^m) = \begin{cases} \max\{maxMinDist(c, N_C^m) | c \in N_C^m\}, \\ \qquad \text{if } N_C^m \text{ is a leaf node} \\ \max\{maxMinDist(N, N_C^m) | \\ \qquad N \text{ is a child node of } N_C^m\}, \\ \qquad \text{if } N_C^m \text{ is a non-leaf node.} \end{cases}$$

However, $minDist(o, N_C^m)$ is a piecewise function based on the relative position of a point o and the MBR of N_C^m. The computation of the $maxMinDist$ values requires computing the maxima of a piecewise function with two variables. This is typically obtained by numerical methods, specifically, by finding the stationary points of a Lagrange function. However, numerical methods are iterative methods and there is no guarantee on the number of iterations needed to find the solution. Therefore, the computation cost is very high and unpredictable for the straightforward way of computing MND.

Next, we propose a much more efficient method to compute the MND. The key observation is that the MND can be derived from those points on the boundary of $NFC(c)$ (MND region of a child node N) that are the "furthest" to N_C^m, and we can limit our search for the "furthest" point within a set of four *candidate furthest points (CFPs)* described as follows.

Fig. 7(a) illustrates the CFPs for a client c indexed in a leaf node N_C^m. In the figure, M denotes the MBR of N_C^m, R denotes $NFC(c)$, R's center point O is located at c and its radius r denotes r's NFD value. A horizontal line L_h and a vertical line L_v intersect each other at O, and they intersect R at I_{h1}, I_{h2}, I_{v1} and I_{v2}, respectively. The four points I_{h1}, I_{h2}, I_{v1} and I_{v2} are the CFPs of c. Similarly, Fig. 7(b) illustrates the CFPs for a child node N of a non-leaf node N_C^m. In the figure, M_1 denotes the MBR of N, R denotes the MND region of N, r denotes $MND(N)$ and O is R's center point. The four points I_{h1}, I_{h2}, I_{v1} and I_{v2} are the CFPs of N.

We denote the largest $minDist(I_i, N_C^m)$ value for the CFPs as $maxMinDist(I, M)$, where I_i denotes a CFP. We will prove below (Theorems 2 and 3) that one of the CFPs must be the "furthest" point from the boundary of $NFC(c)$ (or the MND region of a child node N) to N_C^m, i.e., $maxMinDist(I, M) = maxMinDist(c, N_C^m)$ (or $maxMinDist(N, N_C^m)$) if N_C^m is a leaf node (or a non-leaf node). The intuition here is that we can divide the boundary of $NFC(c)$ (or the MND region of a child node N) into a set of arc segments, and for each segment, there must be a CFP I_i such that for any point o on the segment, $minDist(I_i, N_C^m) \geq minDist(o, N_C^m)$. We find the CFP with the largest $minDist(I_i, N_C^m)$ value and it is the "furthest" point from $NFC(c)$ (or the MND region of N) to N_C^m.



(a) CFPs for a client (b) CFP for a node

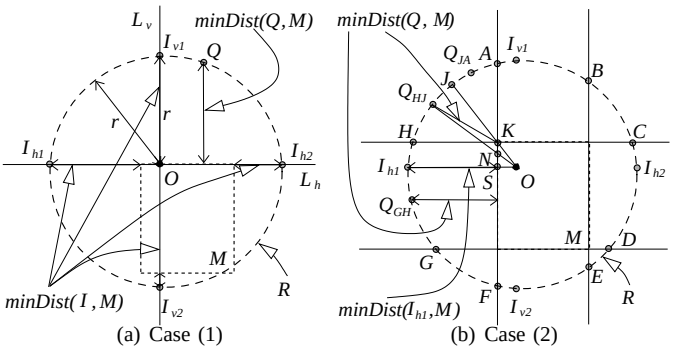
Fig. 7. Candidate furthest points

Theorem 2: Given an MBR M, a circle $R = (O, r)$ and a set I of four candidate furthest points, the largest $minDist$ value from a point Q on the boundary of R to M, $maxMinDist(R, M)$, equals to $maxMinDist(I, M)$.

Proof: If R is enclosed by M, then for every point Q on R, $minDist(Q, M) = 0$. Thus, $maxMinDist(R, M) = maxMinDist(I, M) = 0$.

Otherwise, there are the following two cases.

(1) The center point O is on the boundary of M (cf. Fig. 8(a)). Without loss of generality, we assume O is on the top edge of M. Then $minDist(I_{v1}, M) = r$. For any other point Q on R, $minDist(Q, M) < r$. Thus, $maxMinDist(R, M) =$



(a) Case (1) (b) Case (2)

Fig. 8. Examples of Theorem 2

$maxMinDist(I,M) = r$.

(2) The center point O is inside M (cf. Fig. 8(b)). In this case, we divide the boundary of R into eight arc segments using four lines overlapping the four edges of M. A resultant arc segment is categorized into group 1 if it passes through one of the CFPs (e.g., $\overset{\frown}{GH}$ passing through I_{h1}), and into group 2 if it does not (e.g., $\overset{\frown}{HA}$).

For the arc segments in group 1, the theorem holds because for a point Q on any of these arc segments, $minDist(Q,M) \leq minDist(I_i,M)$, where I_i denotes the CFP passed through by this arc segment. For example, for a point Q_{GH} on $\overset{\frown}{GH}$, $minDist(Q_{GH},M) \leq minDist(I_{h1},M)$.

For group 2, without loss of generality, we prove that for a point Q on $\overset{\frown}{HA}$, $minDist(Q,M) \leq minDist(I_{h1},M)$ or $minDist(Q,M) \leq minDist(I_{v1},M)$, where I_{h1} and I_{v1} are passed through by the two arc segments $\overset{\frown}{GH}$ and $\overset{\frown}{AB}$ that are adjacent to $\overset{\frown}{HA}$, respectively.

Let the top left corner of M be K. As Fig. 8(b) shows, radius OJ passes through K and intersects R at J. Then $minDist(J,M) = |JK|$. Radius OI_{h1} is perpendicular to the left edge of M and they intersect at S. Then $minDist(I_{h1},M) = |I_{h1}S|$. We prove $minDist(J,M) < minDist(I_{h1},M)$ as follows.

$$\left.\begin{array}{l} minDist(I_{h1},M) = |I_{h1}O| - |SO| = r - |SO| \\ minDist(J,M) = |JO| - |KO| = r - |KO| \\ OI_{h1} \perp SK \Rightarrow |KO| > |SO| \end{array}\right\} \Rightarrow$$

$$minDist(J,M) < minDist(I_{h1},M).$$

Point J further divides $\overset{\frown}{HA}$ into two arc segments $\overset{\frown}{HJ}$ and $\overset{\frown}{JA}$. For a point Q_{HJ} on $\overset{\frown}{HJ}$, we prove $minDist(Q_{HJ},M) < minDist(I_{h1},M)$. Similarly, we can prove $minDist(Q_{JA},M) < minDist(I_{v1},M)$ for a point Q_{JA} on $\overset{\frown}{JA}$ and the proof is omitted due to space limit.

Proving $minDist(Q_{HJ},M) < minDist(I_{h1},M)$ equals to proving $|Q_{HJ}K| < |I_{h1}S|$. First, we draw radius OQ_{HJ}, which intersects M at N. Then we prove $|Q_{HJ}K| < |Q_{HJ}N|$ and $|Q_{HJ}N| < |I_{h1}S|$ so as to prove $|Q_{HJ}K| < |I_{h1}S|$. The proof of $|Q_{HJ}K| < |Q_{HJ}N|$ is straightforward based on the law of sines, since $\angle Q_{HJ}KN > \angle Q_{HJ}NK$. Meanwhile, $|Q_{HJ}N| < |I_{h1}S|$ holds because $|Q_{HJ}N| = r - |NO|$, $|I_{h1}S| = r - |SO|$ and $|NO| > |SO|$. Thus, $|Q_{HJ}K| < |Q_{HJ}N| < |I_{h1}S|$ and $minDist(Q_{HJ},M) < minDist(I_{h1},M)$.

Now we have proved that for a point Q on an arc segment in either group 1 or group 2, $minDist(Q,M) \leq maxMinDist(I,M)$. Therefore, the theorem is proved. ■

Theorem 3: Given two MBRs M and M_1, the MND region R of M_1 centered at O and a set I of four candidate furthest points, the largest *minDist* value from a point Q on the boundary of R to M, $maxMinDist(R,M)$, equals to $maxMinDist(I,M)$.

Proof: The proof of this theorem is very similar to that of Theorem 2. Thus, we only provide a sketch of the proof.

If R is enclosed by M, the theorem holds because $minDist(Q,M) = 0$ for every point Q on R's boundary.

Otherwise, there are the following two cases.

(1) The center point O is on the boundary of M. Like case (1) in Theorem 2, the theorem holds because $maxMinDist(R,M) = maxMinDist(I,M) = r$.

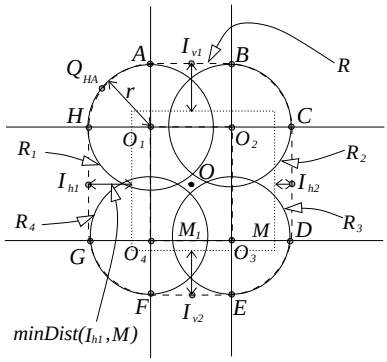


Fig. 9. Theorem 3, case (2)

(2) The center point O is inside M. As shown in Fig. 9, we draw four circles $R_1 = (O_1, r)$, $R_2 = (O_2, r)$, $R_3 = (O_3, r)$ and $R_4 = (O_4, r)$ centering at the four corners O_1, O_2, O_3 and O_4 of M_1, respectively. For each circle, a horizontal line and a vertical line intersecting at its center point also intersect the boundary R. This results in eight candidate furthest points of the four circles, denoted by A, B, ..., H. The CFPs divide R into eight segments. The resultant segments are categorized into two groups. Every segment in group 1 passes through one of the CFPs, while no segment in group 2 does.

For a point Q on a segment in group 1, the theorem holds because there is a CFP of M_1 in the segment.

For group 2, by definition, each segment in it must be overlapped by one of the circles in $\{R_1, R_2, R_3, R_4\}$. Then Theorem 2 guarantees the theorem's correctness. For example, $\overset{\frown}{HA}$ belongs to this group and it is overlapped by R_1. Theorem 2 guarantees that, for any point o on $\overset{\frown}{HA}$, $minDist(o,M) \leq minDist(A,M)$ or $minDist(H,M)$. At the same time, $minDist(A,M) \leq minDist(I_{v1},M)$ and $minDist(H,M) \leq minDist(I_{h1},M)$. Therefore, $minDist(o,M) \leq minDist(I,M)$, where I is one of the CFPs of M_1. ■

Theorems 2 and 3 provide an efficient way to compute the MND, which requires the computation of the *minDist* values for the four CFPs. The specific steps are as follow.

We denote the coordinates of O as (O_x, O_y), and the coordinates of I_{h1}, I_{h2}, I_{v1} and I_{v2} as $(O_x - r, O_y)$, $(O_x + r, O_y)$,

$(O_x, O_y + r)$ and $(O_x, O_y - r)$, respectively. Let M be (M_{x-}, M_{x+}, M_{y-}, M_{y+}) ("$-$" and "$+$" stand for lower bound and upper bound, respectively). Then, $minDist(I_{h1}, M) = M_{x-} - (O_x - r)$, $minDist(I_{h2}, M) = (O_x + r) - M_{x+}$, $minDist(I_{v1}, M) = (O_y + r) - M_{y+}$ and $minDist(I_{h2}, M) = M_{y-} - (O_y - r)$. As a result, according to Theorem 2, we have:

$$\begin{aligned} &maxMinDist(R, M) = \max\{M_{x-} - (O_x - r), \\ &(O_x + r) - M_{x+},\ M_{y-} - (O_y - r),\ (O_y + r) - M_{y+}\} \end{aligned} \quad (1)$$

Now we can compute $maxMinDist(c, N_C^m)$ using Equation (1) for a client c indexed in a leaf node N_C^m. Further, we can compute $MND(N_C^m)$ as follows since it is defined as $max\{maxMinDist(c, N_C^m) | c$ is a client indexed by $N_C^m\}$.

$$\begin{aligned} &MND(N_C^m) = \max\{d_1, d_2, d_3, d_4\}, \text{ where} \\ &d_1 = \max\{c_y + dnn(c,F) | c \in N_C^m\} - \max\{c_y | c \in N_C^m\}, \\ &d_2 = \max\{c_x + dnn(c,F) | c \in N_C^m\} - \max\{c_x | c \in N_C^m\}, \\ &d_3 = \min\{c_y | c \in N_C^m\} - \min\{c_y - dnn(c,F) | c \in N_C^m\}, \\ &d_4 = \min\{c_x | c \in N_C^m\} - \min\{c_x - dnn(c,F) | c \in N_C^m\}. \end{aligned}$$

According to Theorem 3, we can replace c by N and replace dnn by MND in the above equation to obtain an equation for computing the MND value of a non-leaf node N_C^m, where N denotes a child node of N_C^m and MND denotes its MND.

Compared with the straightforward MND computation approach, which requires an expensive iterative method for computing maxima, the above proposed method requires only several arithmetic operations, which has a constant low cost. As the MND computation is performed recursively in a bottom up manner, it resembles the procedure of MBR computation for R-tree construction and maintenance. Therefore, the MND computation can be integrated straightforwardly into the standard R-tree procedures with negligible overhead.

VII. Cost Analysis

In this section, we analytically compare for all described methods (SS, QVC, NFC and MND) the precomputation cost, I/O cost, and CPU cost. Table III summarizes the analytical results, but omits CPU cost as it is just the product of I/O cost and processing cost per node (block).

We first introduce the notation and equations used in the analysis. Let C_m be the maximum number of entries in a disk block ($i.e., C_m$ = block size / size of a data entry). Let C_e be the effective capacity of an R-tree, i.e., the average number of entries in an R-tree node. The average height of an R-tree is $h = \lceil \log_{C_e} n \rceil$ where n is the cardinality of the dataset; the cardinalities of C, F and P are denoted 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 $\sum_{i=1}^{h} \frac{n}{C_e^i} = n\left(\frac{1}{C_e} + \frac{1}{C_e^2} + \cdots + \frac{1}{C_e^h}\right) = \frac{n}{C_e - 1}(1 - \frac{1}{C_e^h}) \approx \frac{n}{C_e - 1}$. We assume an R-tree node has the size of a disk block.

A. Precomputation and Index Cost

We precompute $dnn(c,F)$ for all methods. Computing $dnn(c,F)$ for all clients has the cost of $O(n_c \cdot n_f)$ since $dist(c, f)$ for each pair of client c and existing facility f needs to be computed. The result of $dnn(c,F)$ may be incrementally maintained and therefore the cost is amortized.

TABLE III
Summary of Costs

Method	Precomp	Indexes	I/O Cost
SS	dnn	N/A	$\frac{n_p n_c}{C_m^2}$
QVC	dnn	R_C, R_F	$\frac{n_p}{C_m} + k\frac{n_p n_f}{C_e - 1} + n_p(1 - w_q)\frac{\log_{C_e} n_c}{C_m}$
NFC	dnn	R_C, R_C^n, R_P	$(1 - w_n)\frac{n_c n_p}{(C_e - 1)^2}$
MND	dnn	R_C^m, R_P	$(1 - w_m)\frac{n_c n_p}{(C_e - 1)^2}$

QVC uses R_C and R_F. NFC and MND all use R_P. In addition, NFC uses R_C and the RNN-tree R_C^n, while MND uses the R-tree variant R_C^m. The cost of maintaining any of the R-tree variants is very similar to the cost of maintaining a traditional R-tree. For example, R_C^n has the same C_m and C_e as R_C, so it has almost the same maintenance cost as R_C. R_C^m has an additional attribute in each entry, which reduce C_e a little bit. However, the effect on the height of the tree is very small. For example, in our experiments, where every entry of R_C stores only its MBR and a child node pointer, the height of R_C^m is less than 10% larger than that of R_C. The difference in height will be even smaller in practical databases where an entry is much larger than just an MBR. Therefore, we do not distinguish C_m (C_e) of different R-tree variants.

In summary, except for the costs of building indexes, all methods have the same precomputation cost. QVC and MND have similar R-tree maintenance costs and the NFC method maintains one more R-tree.

B. I/O Cost

For SS, the data points are retrieved in blocks from the disk; 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 other three methods, the I/O costs depend on the number of R-tree nodes accessed. In NFC and MND, 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 (R_C^n or R_C^m) that satisfy 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 costs for these two methods are the same: $\frac{n_c}{C_e - 1}\frac{n_p}{C_e - 1} = \frac{n_c n_p}{(C_e - 1)^2}$. While this worst-case I/O cost is worse than the I/O cost of SS, in practice, many nodes of the R-trees are pruned during traversals. We quantify the percentage of pruned nodes in the simultaneous traversal of the two R-trees as the pruning power, denoted by w; the number of nodes accessed is then $(1 - w)\frac{n_c n_p}{(C_e - 1)^2}$, where w should be replaced by w_n and w_m for NFC and MND, respectively. The cost difference among the two methods lies in the different pruning powers of the two algorithms. Next, we focus on their pruning power differences.

The pruning power is associated with the metrics used in the determination of whether the subtree rooted at a client R-tree node indexes the clients in $IS(p)$ of some potential location p, which are $dnn(c,F)$ and MND for NFC and MND, respectively. The affected regions corresponding to

these metrics are the MBR of the NFCs and the MND region. According to the definitions of these metrics, the area covered by the MND region is very similar to that covered by the MBR of the NFCs. This means that $w_m \approx w_n$ and hence $IO_m \approx IO_n$. This relationship is also observed in our experiments.

QVC involves the following I/O costs. (i) Fetch P from the disk in blocks, $IO_{q1} = \frac{n_p}{C_m}$. (ii) For each potential location p, perform a best-first NN query to construct $AIR(p)$: the I/O cost is $IO_{q2} = n_p \cdot k\frac{n_f}{C_e-1}$ where k indicates the average percentage of R_F nodes accessed in the NN query. (iii) For every $AIR(p)$, perform a window query on R_C: the I/O cost is $IO_{q3} = \frac{n_p}{C_m} \cdot (1-w_q)\log_{C_e} n_c$. Therefore, the I/O cost of QVC is $IO_q = IO_{q1} + IO_{q2} + IO_{q3} = \frac{n_p}{C_m} + k\frac{n_p n_f}{C_e-1} + \frac{n_p}{C_m}(1-w_q)\log_{C_e} n_c$.

The I/O cost of SS is much larger than that of NFC or MND due to its lack of pruning capability. The I/O cost of QVC depends on C_m and can be larger than SS under certain circumstances as follows. Let $IO_{nn} = k\frac{n_f}{C_e-1}$ (i.e., the I/O cost of the NN query discussed above). Based on the I/O costs of SS and QVC, if $C_m^2 IO_{nn} > n_c$, we obtain $C_m IO_{nn} > \frac{n_c}{C_m}$. Hence, $\frac{n_p}{C_m}\left(1 + C_m k\frac{n_f}{C_e-1} + (1-w_q)\log_{C_e} n_c\right) > \frac{n_p n_c}{C_m^2}$ and thus, $IO_q > IO_s$. For example, in our experiments, when $n_c = 10$K and $C_m = 204$, $IO_q > IO_s$ whenever $IO_{nn} > 2.4$. This is a situation where NN query only accesses 2.4 nodes in R_F. In general, $IO_s > IO_q$ when n_c is huge or n_f is small.

C. CPU Cost

The CPU cost can be considered as the product of the CPU cost per block (node) multiplied by the number of blocks (nodes) accessed. The I/O cost analysis provides the number of nodes accessed. The CPU cost per block, denoted by t, involves MBR intersection check and/or metric computation.

The NFC method requires the intersection examination of the MBRs, and MND requires only the computation of *minDist* and the comparison of *minDist* and *MND*. Therefore $t_m \approx t_n$. For QVC, recall that $IO_q = IO_{q1} + IO_{q2} + IO_{q3}$. Since the first part only involves disk block retrieval, there is very little CPU cost; the CPU cost of QVC is mainly $t_{q2} IO_{q2} + t_{q3} IO_{q3}$ where t_{q2} corresponds to the CPU cost per pair of R_C and R_F nodes during the construction of $AIR(p)$ and t_{q3} indicates the CPU cost per pair of $AIR(p)$ block and R_C node. The third part, $t_{q3} IO_{q3}$, is comparable with the CPU costs of NFC and MND. In fact $t_{q3} \approx t_n$ because both methods perform a window query with the query window being either $AIR(p)$ or $N_P.mbr$, respectively. Due to the additional quasi-Voronoi cell construction stage, QVC has higher CPU cost in general compared with NFC and MND.

While the other methods only compute the values of several metrics for each pair of accessed nodes, SS computes $dist(c,p)$ for every pair of client c and potential location p for each pair of blocks of the client set and the potential location set. Hence, the CPU cost per pair of blocks of SS, t_s, is much higher than that of any other method. Also, IO_s is not smaller than other I/O costs. Thus, SS has the highest CPU cost.

In summary, we have $CPU_s > CPU_q > CPU_m \approx CPU_n$. Our experimental study will also validate this inequality.

TABLE IV
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

VIII. EXPERIMENTAL STUDY

In this section, we report the results of our experiments. Section VIII-B studies the behavior of the different methods using uniform datasets, varying the sizes of the different datasets used in the query. Section VIII-C studies the performance of the methods using datasets of Gaussian and Zipfian distributions, varying the skewness of the data distribution. Section VIII-D presents the experimental results on real datasets.

A. Experimental Setup

All experiments were conducted on a desktop PC with 3GB RAM and 2.66GHz Intel(R) Core(TM)2 Quad CPU. The disk page size is 4K bytes. We measure the running time, the number of I/Os and the index size.

We conduct experiments on synthetic and real datasets. Synthetic datasets are generated with a space domain of 1000×1000. The dataset cardinalities range from 100 to 1000000. Three types of datasets are used: (i) *Uniform datasets*, where data points are distributed randomly; (ii) *Gaussian datasets*, where data points follow the Gaussian distribution; (iii) *Zipfian datasets*, where data points follow the Zipfian distribution. The parameters of the synthetic data experiments are summarized in Table IV, where values in bold denote default values.

We use two groups of real datasets provided by Digital Chart of the World [22], 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.

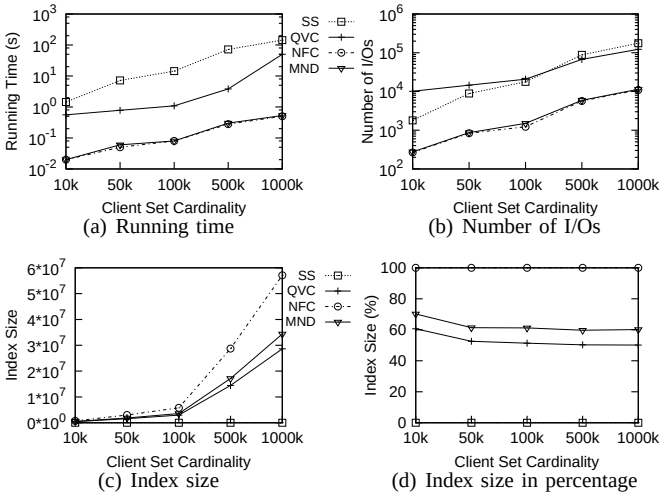
We use the R-tree [17] (or its variants as proposed in this paper) as the underlying access methods.

B. Experiments on Uniform Datasets

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

1) Varying the Number of Clients: In our experiments, we show that MND is the only method whose performance is as good as NFC in terms of the running time and the number of I/Os, while MND has a much smaller index size.

The results for the experiments that vary the number of clients are shown in Fig. 10. From this figure, we can see that the NFC method and the MND method perform best in terms of the running time and the number of I/Os (cf. Fig. 10(a) and (b)). Meanwhile, the MND method has a much smaller index size compared to the NFC method (cf. Fig. 10(c)). Fig. 10(d) gives a different representation of the index size requirements using the measure relative to the index size of the NFC method. For example, for the 10K datasets the index size of the MND method is about 70% of that of the NFC method, and for the 100K datasets, the index size of the MND method drops to about 60% of that of the NFC method.



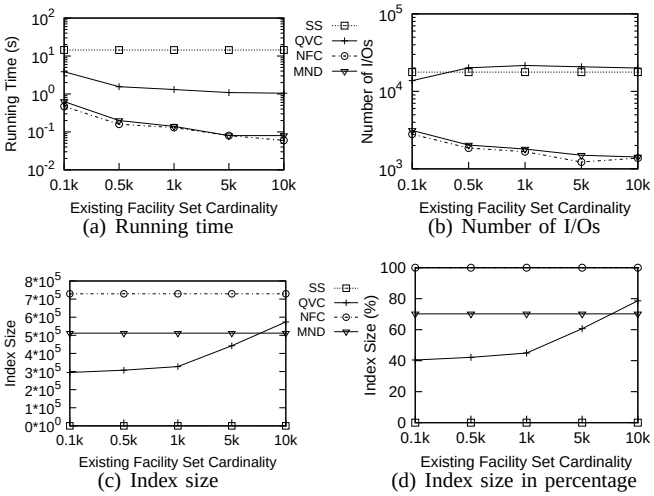
(a) Running time (b) Number of I/Os
(c) Index size (d) Index size in percentage

Fig. 10. The effect of client set size

From Fig. 10, we also observe that, compared with other methods, the SS and QVC methods have significantly higher running time and larger numbers of I/Os, although the QVC method requires slightly less index size than the MND method does and the SS method does not require any index. When the cardinality of the client set is large enough (e.g. 500K), the number of I/Os of SS exceeds that of QVC. The observations above are in accordance with the cost analysis. QVC traverses R_F for each potential location, while either NFC or MND only traverses the R-trees once on average for the entire potential location set. Thus, QVC has larger number of I/Os and higher running time. For SS, $IO_s > IO_q$ whenever n_c is large. It is slow because it does not have any pruning strategy.

2) Varying the Number of Existing Facilities: The running time, the number of I/Os and the index size of the methods, with respect to the number of existing facilities, are shown in Fig. 11. Again, MND and NFC are the most efficient methods in terms of the running time and the number of I/Os, while MND outperforms NFC in terms of index size due to NFC's extra index tree for client indexing.

Other observations can be made from the figure are as follows. First, in terms of the running time and the number of I/Os, the comparative performance of the methods is similar to that of the experiments varying the number of clients. Second, an increase in the number of facilities yields a drop in both the running time and the number of I/Os. The effect is more explicit for the NFC and MND methods. The reason is that on

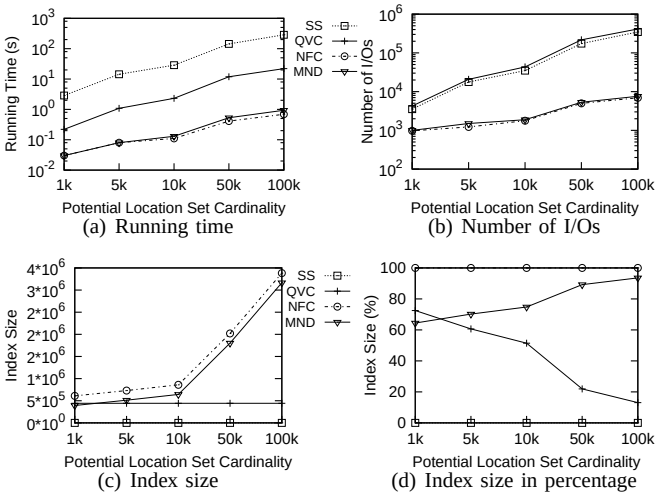


(a) Running time (b) Number of I/Os
(c) Index size (d) Index size in percentage

Fig. 11. The effect of existing facility set size

average the more the facilities, the shorter the nearest facility distance. In other words, $dnn(c,F)$ decreases with the increase of the number of existing facilities. As a result, the areas of NFCs and MND regions decrease and the pruning power 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). Third, when the number of facilities is small enough, the number of I/Os of QVC is less than that of SS, which is in accordance with our cost analysis. Fourth, varying the number of existing facilities only affects the index size of the QVC method, since only this method requires an index on F.

3) Varying the Number of Potential Locations: Experiments that vary the number of potential locations also give results that are very similar to those of the experiments varying the number of clients, as shown in Fig. 12. MND still shows high efficiency in terms of the pruning time, the number of I/Os and the index size.



(a) Running time (b) Number of I/Os
(c) Index size (d) Index size in percentage

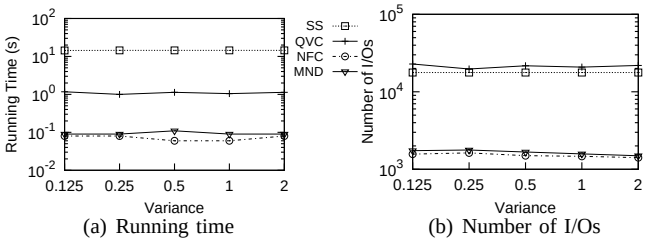
Fig. 12. The effect of potential location set size

From Fig. 12(c) and (d), we notice that the index sizes of the SS and QVC methods are not affected by the increase in the number of potential locations. This is because these two

methods do not index the potential locations, and as a result, they are both slow and have large numbers of I/Os. We also observe that the growth in the number of potential locations has the same effect on the running time and the number of I/Os as increasing the number of clients. When the number of potential locations n_p becomes very large (i.e. $n_p \geq 10$K), the advantages of NFC and MND in terms of the number of I/Os become much significant (cf. Fig. 12(b)).

C. Experiments on Gaussian and Zipfian Datasets

In the following experiments, we vary the distribution of the datasets. We focus on performance of the algorithms in terms of the running time and the I/O cost rather than the index size because the influence of detail data distribution on the index size requirement is not the major concern of this paper.



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

Fig. 13. The effect of σ^2 in Gaussian distribution

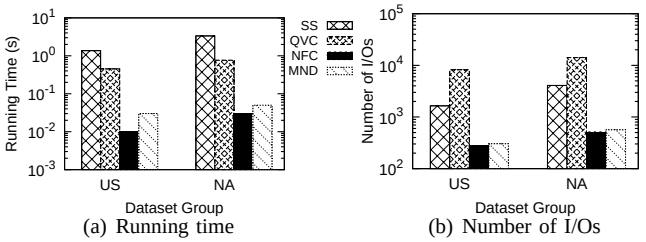
Fig. 13 shows the results of experiments on Gaussian datasets varying the value of σ^2. For the Gaussian datasets, varying σ^2 means varying the degree of the inclination for the data points to cluster at the central area of the distribution. Increasing σ^2 leads to less dense data points at the center. We see that, compared with varying the dataset cardinalities, varying σ^2 does not affect much of the algorithm performance. NFC and MND are still the two most efficient methods. These results follow our cost analysis.

Experimental results on datasets of Zipfian distribution have similar behavior to the above results and are omitted.

D. Experiments on Real Datasets

The experimental results on real datasets are shown in Fig. 14. The comparative performance of the methods is similar to that of experiments conducted for the synthetic datasets. QVC shows the worst performance in terms of the number of I/Os. While the number of I/Os of SS is close to that of QVC, it has the largest running time due to the lack of pruning capability. NFC and MND outperform other methods in terms of both the number of I/Os and the running time.

Overall, the MND method outperforms the other methods.



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

Fig. 14. Performance comparison on real datasets

IX. Conclusions

We formulated the min-dist location selection problem and conducted a comprehensive study. We proposed two methods, QVC and NFC, based on common approaches to location optimization problems. Our experiments show that they significantly outperform the sequential scan algorithm. However, they both have some drawbacks. NFC performs the best but requires maintaining an additional index. QVC requires fewer indexes, but is not as efficient as NFC. We further proposed the MND method, which has very close efficiency to NFC without the need of maintaining an additional index. We provided a detailed comparative cost analysis for all methods and performed extensive experiments to evaluate the empirical performance of them. The results agree with our analysis and validate the advantages of the MND method.

Acknowledgment

This work is supported by the Australian Research Council's Discovery funding scheme (project numbers DP0880250 and DP0880215).

References

[1] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura, "Reverse facility location problems." in *CCCG*, 2005, pp. 68–71.
[2] T. Xia, D. Zhang, E. Kanoulas, and Y. Du, "On computing top-t most influential spatial sites." in *VLDB*, 2005, pp. 946–957.
[3] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progressive computation of the min-dist optimal-location query," in *VLDB*, 2006, pp. 643–654.
[4] ArcGIS. (2011) http://www.esri.com/.
[5] C. Yu, R. Zhang, Y. Huang, and H. Xiong, "High-dimensional knn joins with incremental updates," *Geoinformatica*, vol. 14, pp. 55–82, 2010.
[6] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *SIGMOD*, 1995, pp. 71–79.
[7] G. R. Hjaltason and H. Samet, "Ranking in spatial databases," in *SSD*, 1995, pp. 83–95.
[8] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries." in *SIGMOD*, 2000, pp. 201–212.
[9] C. Yang and K.-I. Lin, "An index structure for efficient reverse nearest neighbor queries," in *ICDE*, 2001, pp. 485–492.
[10] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi, "Discovery of influence sets in frequently updated databases," in *VLDB*, 2001, pp. 99–108.
[11] W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan, "Continuous Reverse k-Nearest-Neighbor Monitoring," in *MDM*, 2008.
[12] R. Zhang, D. Lin, R. Kotagiri, and E. Bertino, "Continuous intersection joins over moving objects." in *ICDE*, 2008, pp. 863–872.
[13] R. Zhang, J. Qi, D. Lin, W. Wang, and R. C.-W. Wong, "A highly optimized algorithm for continuous intersection join queries over moving objects," *to appear in the VLDB Journal*.
[14] Y. Du, D. Zhang, and T. Xia, "The optimal-location query." in *SSTD*, 2005, pp. 163–180.
[15] Y. Gao, B. Zheng, G. Chen, and Q. Li, "Optimal-location-selection query processing in spatial databases," *TKDE*, vol. 21, pp. 1162–1177, 2009.
[16] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He, "Top-k most influential locations selection," in *CIKM*, 2011.
[17] A. Guttman, "R-trees: A dynamic index structure for spatial searching." in *SIGMOD*, 1984, pp. 47–57.
[18] F. Aurenhammer, "Voronoi diagrams - a survey of a fundamental geometric data structure," *ACM CS*, vol. 23, pp. 345–405, 1991.
[19] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using r-trees," in *SIGMOD*, 1993, pp. 237–246.
[20] Y. Tao, D. Papadias, and X. Lian, "Reverse knn search in arbitrary dimensionality," in *VLDB*, 2004, pp. 744–755.
[21] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao, "Reverse nearest neighbors in large graphs," *TKDE*, vol. 18, pp. 540–553, 2006.
[22] RtreePortal. (2011) http://www.rtreeportal.org.