World Wide Web manuscript No.
(will be inserted by the editor)

The Min-dist Location Selection and Facility Replacement Queries

**Jianzhong Qi · Rui Zhang · Yanqiu Wang ·
Andy Yuan Xue · Ge Yu · Lars Kulik**

Received: date / Accepted: date

Abstract We propose and study a new type of location optimization problem, the min-dist location selection 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. The problem has a wide range of applications in urban development simulation, massively multiplayer online games, and decision support systems. We also investigate a variant of the problem, where we consider replacing (instead of adding) a facility while achieving the same optimization goal. We call this variant the min-dist facility replacement problem. We explore two common approaches to location optimization problems and present methods based on those approaches for solving the min-dist location selection 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 then utilize the key idea behind MND to approach the min-dist facility replacement problem, which results in two algorithms names MSND and RID. We provide a detailed comparative cost analysis and conduct extensive experiments on the various algorithms. The results show that MND and RID outperform their competitors by orders of magnitude.

Keywords Spatial database · geographic information system · location optimization · min-dist metric

1 Introduction

Location optimization is an important problem for spatial decision support systems. A number of studies [4, 5, 25, 30] proposed solutions to various instances of such problems. In this

J. Qi, R. Zhang, A. Y. Xue and L. Kulik
Department of Computing and Information Systems
University of Melbourne, Australia
Tel.: +61-3-83441332
Fax: +61-3-83441345
E-mail: {jiqi, rui}@csse.unimelb.edu.au, {andy.xue, lkulik}@unimelb.edu.au

Y. Wang and G. Yu
College of Information Science and Engineering
Northeastern University, China
E-mail: {wangyanqiu, yuge}@ise.neu.edu.cn

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 establish a new facility (e.g., fire hydrant, hospital, bus stop) or replace an existing one. When selecting the location for the new facility (or the facility to be replaced), a commonly used criterion is to select the location (or the facility and the location for replacement) 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 many non-player characters (NPCs) like monsters to fight with the players. Very often the game server needs to generate a new NPC for the fighting. If the new NPC is placed randomly, there may be no player around it at all and this will be a waste of the limited computational resource of the game server. A very helpful utility for the game server is selecting a starting point for the NPC from a set of preset locations to minimize the average distance between a player and her nearest NPC, so that the players can find NPCs closer and do not get bored walking around trying to find an NPC to fight with. As the players keep moving around and/or leaving the game, an existing NPC may become too far away to be seen by the players. In this case the game server may want to move the NPC to re-balance the NPCs and the players, i.e., move the NPC to a place to again minimize the average distance between a player and her nearest NPC.

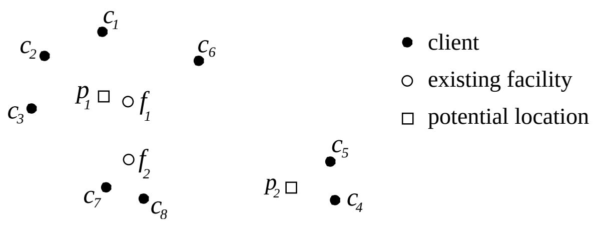


Fig. 1 An example of the studied problems

Fig. 1 gives an example: $\{c_1, c_2, ..., c_8\}$ is a set of clients (residents or players), $\{f_1, f_2\}$ is a set of existing facilities (public facilities or NPCs) and $\{p_1, p_2\}$ is a set of potential locations (candidate locations for facility establishment or NPC placement). Currently, 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.

We consider two scenarios. (i) We select one of the potential locations to establish a new facility. If a new facility is established at p_1, it will become the nearest facility for c_1, c_2 and c_3. If it 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 her nearest facility, so it is selected as the answer. (ii) We select an existing facility and a potential location for facility replacement. There are four possible choices, i.e., $f_1 \rightarrow p_1$, $f_1 \rightarrow p_2$, $f_2 \rightarrow p_1$ and $f_2 \rightarrow p_2$, where "$\rightarrow$" denotes "to be replaced with". Here, the choice is not so obvious because when replacing a facility, some of the clients can get a closer facility while some others may get a further one. For example, if f_1 is replaced with p_1, c_1, c_2 and c_3 will get a closer facility, while c_6 will have a further facility. The crux is to efficiently compute the aggregate effect. In this example, replacing f_2 by p_2 ($f_2 \rightarrow p_2$) can minimize the average distance between a client and the nearest facility. Therefore, it is selected as the answer.

Besides the above described applications, many organizations and businesses face similar decision making problems (e.g., a bank needs to add or replace ATMs; a wireless service provider needs to set up or replace hotspots). This paper studies how to efficiently select a location for new facility establishment or a pair of facility and location for facility replacement, so that the average distance between a client and her nearest facility is minimized. We call the two problems the *min-dist location selection problem* and *the min-dist facility replacement problem*, respectively. In the aforementioned applications, the location selection or facility replacement may be performed frequently. Thus, we formulate the two problems as the following queries.

1.1 Problem Formulation

All data objects (clients, facilities and potential locations) are represented by points in a 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 $p \in P$ for a new facility to be established at, so that $\forall p' \in P$,

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

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

Similarly, the min-dist facility replacement query is defined as follows.

Definition 2 ***Min-dist facility replacement 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 facility replacement query finds a pair of existing facility and potential location, denoted by f and p respectively, so that $\forall \langle f', p' \rangle \in F \times P$,

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

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

Since the denominator is the same on both sides of the inequalities, the problems are 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 [1] can solve several simpler location optimization problems, none can solve the min-dist location selection or facility replacement problems (see Section 2 for more discussion).

1.2 Contributions and Organization of the Paper

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

- We formulate the min-dist location selection problem and the min-dist facility replacement problem.
- We explore two common approaches to location optimization problems and propose methods based on them for solving the location selection problem.
- 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 for the location selection problem, 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 extend the idea of MND further and propose two algorithms named MSND and RID to solve the facility replacement problem.
- 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 superiority of MND and RID over the other methods.

This article is an extended version of our earlier paper [18]. There we proposed the min-dist location selection problem and studied algorithms to solve the problem. In this article, we extend our work by investigating an important problem variant, the min-dist facility replacement problem. We propose two algorithms to solve the problem. The challenge here is a search space cubically proportional to the size of the datasets to be accessed to find the optimal facility-location pair for the replacement. To address the challenge, we transform the expensive search into two operations: a lightweight cubical search plus two lightweight quadratic search. We extend our cost analysis to the newly proposed algorithms and perform additional experiments on them. The results show that the algorithms can solve the min-dist facility replacement problem efficiently.

The rest of the article is organized as follows. Section 2 reviews related work. Section 3 studies the min-dist location selection problem and describes algorithms to solve the problem. Section 4 investigates solutions to the min-dist facility replacement problem. Section 5 analyzes the cost of the proposed algorithms. Section 6 presents the experimental results and Section 7 concludes the article.

2 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 and RNN 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 [19] and best-first [10]. The best-first algorithm can retrieve the nearest neighbors incrementally in order of their distances to the query point. Various studies have been conducted on variants of the NN search, such as visible NN queries [15], kNN joins [29], moving kNN queries [16,17] and predictive kNN queries [31].

As one of the major variants, Korn and Muthukrishnan [13] 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. 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 [27] 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.* [21] propose a method without precomputation. For a query point q, their method dynamically constructs a Voronoi cell that encloses q and contains all its possible RNNs. Only nodes intersecting the Voronoi cell have to be accessed to check for q's RNNs. While these methods work well for a single RNN query, they are not for computing RNNs of large amount of objects at the same time, which is a key difficulty in our study. 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. [24] 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 [32, 33] also correlates multiple sets, but it focuses on intersecting objects with a time-constraint technique rather than closeness. While these approaches work well for a single R(k)NN query, they are not tailored for computing RNNs for a large amount of objects at the same time, which is a key difficulty in our study. There are studies on processing a large amount of objects at the same time, e.g., the group NN query [6] and the convoy query [12], which query the NNs of a group of query objects together and groups of objects that have traveled together, respectively. However, these studies are of quite different settings, and their methods do not apply.

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.* [4] find regions for a new facility to maximize its influence. They introduce

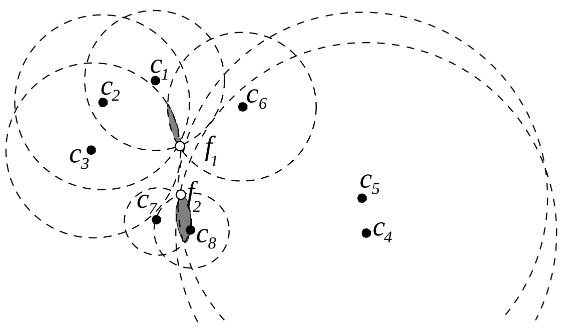


Fig. 2 Example of a max-inf problem [4]

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 the nearest existing facility of c. Since only a facility established within the NLC of c can be a new nearest facility of c, to find the problem solution is to find the regions that are enclosed by the largest number of NLCs. In Fig. 2, 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. Cabello et al.'s study only gave a theoretical analysis for the problem. Wong et al. [23] study this problem further and propose a method to reduce the complexity of finding regions overlapped by the largest number of NLCs. Xia *et al.* [25] 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.* [7] find a location in a region Q for a new facility to maximize its influence. Gao *et al.* [8] find 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 [11] 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. Their solutions do not apply.

Min-dist problems: Zhang *et al.* [30] 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

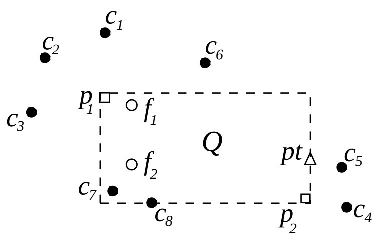


Fig. 3 Example of a min-dist problem [30]

be one of the points in the answer set and it is not the solution p_2 to our location selection problem. To solve the problem, Zhang *et al.* [30] 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. Xiao *et al.* [26] study the min-dist problem in road networks. They have a candidate edge set E for the new facility to be established at. Their key insight is that the optimal location on a candidate edge must be one of the endpoints of the edge. Thus, only the endpoints of the edges in E need to be checked for the problem solution.

These two studies [30, 26] have the same min-dist optimization function as ours, but our study has a set P, the potential locations given as candidates for selection or replacement. In many real applications, we can only choose from some candidate locations, e.g., a bank may only set up a new ATM at or relocate an existing ATM to a place for rent or sale rather than anywhere in a region or on a road. 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. Similarly, the endpoints of the edges in E [26] are different from the points in P. This means that in general their approaches cannot provide a correct answer to our problems, and thus are not applicable.

Related commercial software: As mentioned in Section 1, an existing commercial software [1] 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 the problems we study.

In computational geometry, given a set C of objects (e.g., clients), the k-medoid query [14] 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 variant, where we find k locations called the medians, not necessarily in C, to 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 F or a set P, but we do.

Table 1 The Location Optimization Problems

Problem	Optim. Function	Solution Space	Distance Function	Datasets
[4]	Max-inf	Continuous	L_2	C, F
[23]	Max-inf	Continuous	L_2	C, F, P
[25]	Max-inf	Discrete	L_2	C, F
[7]	Max-inf	Continuous	L_1	C, F
[8]	Max-inf	Discrete	L_2	C, P
[11]	Max-inf	Discrete	L_2	C, F, P
[30]	Min-dist	Continuous	L_1	C, F
[26]	Min-dist	Continuous	Network	C, F, E
[14]	Min-dist	Discrete	L_2	C
[1]	Min-dist	Discrete	L_2	C, P
Proposed	Min-dist	Discrete	L_2	C, F, P

Summary: Table 1 summarizes the differences between our problems and previously studied location optimization problems. Most previous problems are max-inf problems and differ from our problems in optimization functions. For the min-dist problems, they have the same optimization function as ours, but their problem settings are different. As discussed in the second paragraph of the related min-dist problems, they do not choose from a set of given candidate locations, which does not apply to the requirements of our applications.

3 The Min-dist Location Selection Query

In this section we present algorithms to process the min-dist location selection query. Straightforwardly, the query can be processed as follows. We sequentially check all potential locations and 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. *K*NN-join algorithms (e.g., [29]) can do this more efficiently and maintain the results dynamically when clients and facilities are updated. The SS algorithm with precomputation is shown in Al-

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
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 $optLoc$;
```

gorithm 1, where $c.dnn(c, F)$ denotes the precomputed distance between c and her closest existing facility. The distance is stored with the record of c.

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.

We observe that the min-dist location selection query can be redefined in a form that reduces the search space and thus accelerates query processing. Next, we provide the redefinition and a solution framework based on it. Then we present algorithms under the solution framework to process the query.

3.1 Problem Redefinition and a Solution Framework

We start with some basic properties of the problem needed for the redefinition. Table 2 summarizes frequently used symbols.

3.1.1 Problem 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 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.

Table 2 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
R_C, R_P, R_F	R-trees on C, P, and F, respectively
N_C, N_P, N_F	A node of R_C, R_P, and R_F, respectively
e_c, e_p, e_f	An entry of R_C, R_P, and R_F, respectively
$dnn(c, F)$	The nearest facility distance of c
$IS(f)$, $IS(p)$, $IS(\langle f, p\rangle)$	The influence sets of f, p, and $\langle f, p\rangle$, respectively
$dr(f)$, $dr(p)$, $dr(\langle f, p\rangle)$	The distance reduction of f, p, and $\langle f, p\rangle$, respectively
$dr(\langle f, p\rangle, c)$	The distance reduction achieved for c by $\langle f, p\rangle$
$NFC(c)$	The nearest facility circle of c
$MND(c)$	The maximum NFC distance of c
$SNFC(c)$	The second nearest facility circle of c
$MND(c)$	The maximum second NFC distance of c
$RID(c)$	The replacement influence distance of c

Definition 3 *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 location $p \in P$, so that $\forall p' \in P$: $dr(p) \geq dr(p')$.*

3.1.2 A Solution Framework

Definition 3 provides a framework for solving the min-dist location selection problem with the following two steps:

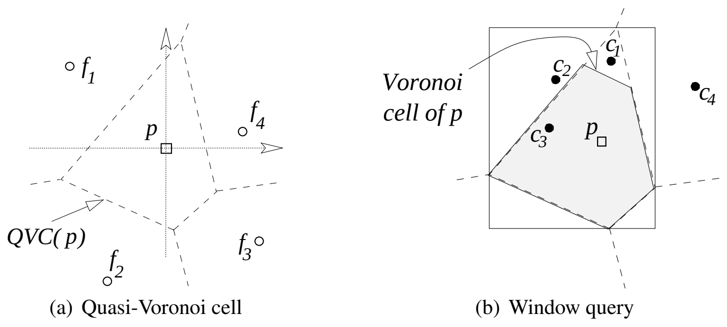
1. Identify $IS(p)$;
2. Compute $dr(p)$ and find the potential location with the largest $dr(p)$.

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 section 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 subsection, we explore two common approaches to location optimization problems and propose methods based on those approaches for solving the min-dist location selection problem under the above framework. When a spatial index is used, we assume an R-tree [9], although any hierarchical spatial index could be used.

3.2 Quasi-Voronoi Cell Method

In this subsection, 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 $o \in V$, $dist(p, o) \leq dist(p', o)$ [2].



(a) Quasi-Voronoi cell (b) Window query

Fig. 4 Examples of the QVC method

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. Stanoi *et al.* [21] 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.

Algorithm 2: QVC(R_C, R_F, F_P)

```
1  $optLoc \leftarrow$ NULL;
2  while not EndOfFile( $F_P$ ) do
3      $B_P \leftarrow$ ReadBlock( $F_P$ );
4      $S_p \leftarrow \emptyset$;
5      for $p \in B_P$ 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              $S_p \leftarrow S_p \cup p$;
10     WQ( $R_C.rootnode$, $S_p$, $optLoc$ );
11 output $optLoc$;
```

In the coordinate system with the origin at p and the two axes parallel to the original axes, we 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). We 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 $QVC(p)$. To find the NN in each quadrant, we use the best-first algorithm [10] 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. Once we have $QVC(p)$, we perform a window query on an R-tree named R_C that indexes the clients with the query range being the *minimum bounding rectangle (MBR)* of $QVC(p)$ (Fig. 4(b)). Then we can further compute $dr(p)$ and determine the optimal potential location. The QVC method is summarized in Algorithms 2 and 3.

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

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

3.3 Nearest Facility Circle Method

In this subsection, 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 $dnn(c, F)$. For a potential location p, $c \in IS(p)$ if and only if p is inside $NFC(c)$. As shown in Fig. 5, p_1 is in the NFCs of c_1, c_2 and c_3, and

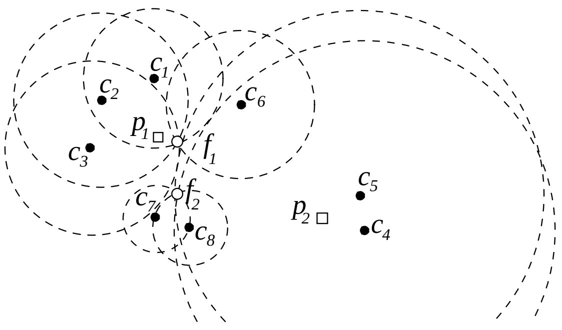


Fig. 5 Example of NFCs

p_2 is in the NFCs of c_4 and c_5. Thus, $IS(p_1) = \{c_1, c_2, c_3\}$ and $IS(p_2) = \{c_4, c_5\}$. 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 [13], denoted as R^n_C, to index the NFCs of all clients. The tree R^n_C 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.

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^n_C 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 P and all NFCs. 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 $dr(p)$. We use a standard R-tree based join algorithm [3] to join R_P and R_C^n, which results in the NFC algorithm. The pseudo-code of the algorithm has been listed in [18] and is omitted here to keep the paper concise.

3.4 Maximum NFC Distance Method

We have presented two methods based on common approaches to location optimization problems. However, those methods both have some drawbacks. The QVC method needs to perform a *K*NN 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 [21,27,22,28]. We also view the extra index for the NFC method as a serious drawback.

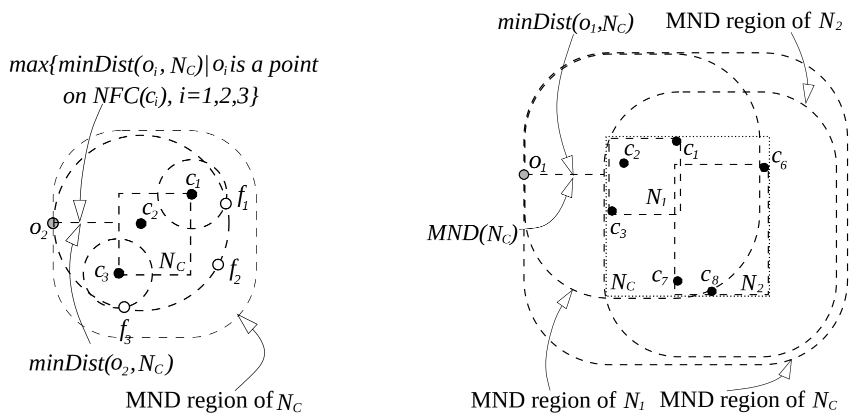
In this subsection, we propose a novel method that is simple and efficient, but requires no extra index, so it overcomes the drawbacks of the QVC and NFC methods. This method still exploits the idea of NFCs. However, unlike the NFC method, which uses an MBR to bound the NFCs of all clients in a node 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 rounded rectangular region around a node N_C that encloses the NFCs of all clients in N_C, and stores that value in the parent entry of N_C in 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*.

3.4.1 The Maximum NFC Distance

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 minimum 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 $MND(N_C)$ (i.e., $minDist(N_C, N_P) \geq MND(N_C)$), 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.

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



(a) MND for a leaf node (b) MND for a non-leaf node

Fig. 6 Examples of MND regions

$MND(N_C)$ as $minDist(o_i, N_C)$, and it delimits the MND region of N_C, the rounded rectangular region in Fig. 6(b).

3.4.2 The Algorithm

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 let $minDist(N_C, N_P)$ 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 See reference [18].

Theorem 1 suggests that we only need to check whether the distance between N_C and 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 use 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 4 details the steps.

3.4.3 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

Algorithm 4: MND(N_P, N_C^m, $optLoc$)

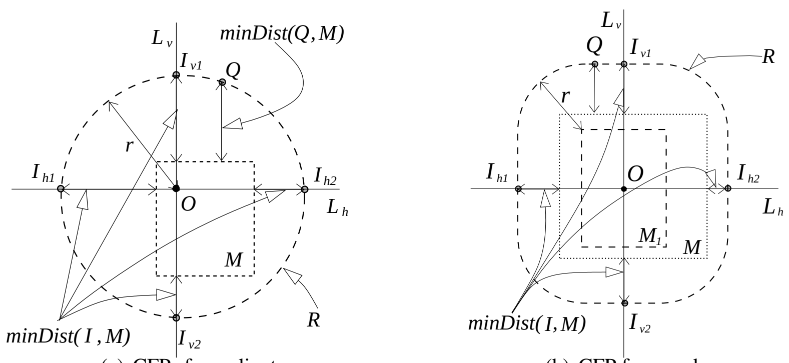
```
1  if $N_P$ and $N_C^m$ are non-leaf nodes then
2      for $(e_p, e_c^m) \in N_P \times N_C^m$, $minDist(e_c^m, e_p) < e_c^m.mnd$ do
3          MND($e_p.childnode$, $e_c^m.childnode$, $optLoc$);
4  else if $N_P$ is a leaf node and $N_C^m$ is a non-leaf node then
5      for $e_c^m \in N_C^m$, $minDist(e_c^m,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

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 children of N_C^m, $MND(N_C^m)$ is computed as the largest $maxMinDist$ value among all these children's $maxMinDist$ values. 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, which 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.

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 "farthest" to N_C^m, and we can limit our search for the "farthest" point within a set of four *candidate farthest 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 $NFC(c)$; the center point O of R is located at c and the radius r denotes the NFD of c. 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 the center point of R.

We denote the largest $minDist(I_i, N_C^m)$ value for the CFPs as $maxMinDist(I, M)$, where I_i denotes a CFP. Theorems 2 and 3 below states that one of the CFPs must be the "farthest" 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 "farthest" 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 farthest points

Theorem 2 *Given an MBR M, a circle $R = (O, r)$ and a set I of four candidate farthest 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 See reference [18].

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 farthest 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 See reference [18].

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:

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

Now we can compute $maxMinDist(c, N_C^m)$ with Equation (1) for a client c indexed in a leaf node N_C^m and further 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.

Discussion. We notice that the RdNN-tree [27] has a similar structure to the tree we use in the MND method to index the clients. In the RdNN-tree, a node N is associated with the largest dnn value of all objects indexed in $sub(N)$, while in the MND method, a node N is associated with the MND value, which by definition is guaranteed to be in general smaller than and at worst the same as the largest dnn value. Therefore, the pruning capability and efficiency of the MND method is guaranteed to be better than a method using the RdNN-tree.

4 The Min-dist Facility Replacement Query

In this section we present algorithms to process the min-dist facility replacement query. We first redefine the query to a similar form to the min-dist location selection query, so that some of the the pruning techniques can be reused.

Definition 4 *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 facility replacement query finds a pair of existing facility and potential location, denoted by f and p respectively, so that $\forall\langle f', p'\rangle \in F \times P$: $dr(\langle f, p\rangle) \geq dr(\langle f', p'\rangle)$.*

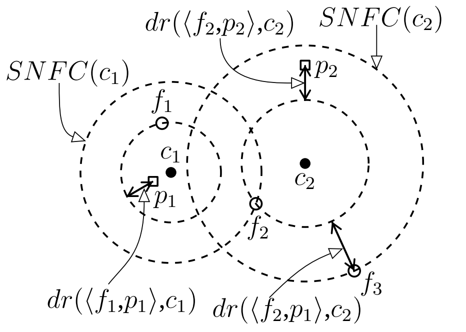
Here $dr(\langle f, p\rangle)$ denotes the distance reduction if f and p are selected for the replacement. Formally, $dr(\langle f, p\rangle) = \sum_{c \in IS(\langle f,p\rangle)}(dnn(c, F) - dnn(c, F \setminus \{f\} \cup \{p\}))$, where $IS(\langle f, p\rangle)$ is a subset of clients whose nearest facilities change when f is replaced by p.

4.1 Sequential Scan Method

The redefined query can be processed with a *sequential scan (SS-FR)* algorithm. It scans the set of clients to compute $dr(\langle f, p\rangle)$ for every pair of existing facility f and potential location p and returns the pair with the largest dr.

For dr computation, we also want to reuse the precomputed dnn of the clients to avoid repetitive client facility distance computation. However, dnn alone are not enough to produce accurate dr values. This is because, when f is replaced with p, the nearest facility of a client c can change to a place that is not p, and thus, the distance reduction achieved for c is not $dnn(c, F)$ - $dist(c, p)$ any more. Fig. 8 gives an example. If we replace f_2 with p_1, then the nearest facility of c_2 changes to f_3 instead of p_1. The distance reduction of $\langle f_2, p_1\rangle$ gained for c_2 is $dnn(c_2, F) - dist(c_2, f_3)$.

We observe that, when f is replaced by p, the nearest facility of c may stay unchanged, or become either p or the existing *second nearest facility (SNF)* of c, depending on the relative position of f, p, c and the SNF of c. We analyze the different cases and summarize the computation of the distance reduction of $\langle f, p\rangle$ gained for c, $dr(\langle f, p\rangle, c)$, in the following



$dr(\langle f_2,p_2\rangle,c_2)$
$SNFC(c_2)$
$SNFC(c_1)$
p_2
f_1
c_1
c_2
p_1
f_2
f_3
$dr(\langle f_1,p_1\rangle,c_1)$
$dr(\langle f_2,p_1\rangle,c_2)$

Fig. 8 Computation of $dr(\langle f,p\rangle)$

equation. Here, $d2nn(c,F)$ denotes the distance between c and its SNF.

$$dr(\langle f,p\rangle,c)=\begin{cases} dnn(c,F)-dist(c,p), & dist(c,p)\le dnn(c,F) \text{ or } \quad \text{(i)} \\ \quad (dist(c,F)=dnn(c,F) \text{ and } dist(c,p)\le d2nn(c,F)) & \\ dnn(c,F)-d2nn(c,F), & dist(c,F)=dnn(c,F) \text{ and } \quad \text{(ii)} \\ \quad dist(c,p)>d2nn(c,F) & \\ 0, \text{ otherwise.} & \text{(iii)} \end{cases} \tag{2}$$

Here, case (i) says that, if p is closer to c than the existing nearest facility of c (cf. Fig. 8, $dr(\langle f_1,p_1\rangle,c_1)$), or p is closer to c than the existing second nearest facility of c while f is the existing nearest facility of c, (cf. Fig. 8, $dr(\langle f_2,p_2\rangle,c_2)$), then p will become the new nearest facility of c after the replacement. Therefore, $dr(\langle f,p\rangle,c)=dnn(c,F)-dist(c,p)$. Cases (ii) and case (iii) say that, if p is not closer to c than even the existing second nearest facility of c, then $dr(\langle f,p\rangle,c)$ is solely determined by f and c. If f is the existing nearest facility of c (case (ii)), then $dr(\langle f,p\rangle,c)=dnn(c,F)-d2nn(c,F)$ (cf. $dr(\langle f_2,p_1\rangle,c_2)$, Fig. 8). Otherwise (case (iii)), then $\langle f,p\rangle$ will not affect c at all. Thus, $dr(\langle f,p\rangle,c)=0$ (e.g., $\langle f_3,p_2\rangle$ and c_1 in Fig. 8).

Based on Equation 2, we can precompute dnn and $d2nn$ at the same time, and then use them in the sequential scan method for dr computation. This will result in an algorithm with a three layered nested loop, where the outer two layers iterate the facility-location pairs while the inner most layer iterates the clients to compute dr values. The algorithm suffers from low efficiency and poor scalability. In the following subsections, we explore pruning techniques to reduce the search space and accelerate query processing.

4.2 Maximum SNFC Distance Method

We investigate techniques to restrict the search space for dr computation. The technique we found mimics that of the MND method but also involves clients' *second nearest facility circles (SNFC)*, which are circles on clients' second nearest facilities (cf. Fig. 8). We call the resultant method the *Maximum SNFC Distance (MSND)* method.

The MSND method is based on that, in Equation 2, only the clients satisfying cases (i) and (ii) can contribute to $dr(\langle f,p\rangle)$. Thus, we can index the clients in an R-tree and then for each facility-location pair, we perform a range query using the conditions in Equation 2 as the predicate to retrieve the relevant clients for dr computation.

The range query will require the clients' dnn and $d2nn$ for predicate computation. Thus, we need to store them in the R-tree and define distance metrics for the tree nodes to enable predicate computation. We apply the technique used in the MND method to build an R-tree variant, R_C^s, to index the clients and bounding regions for their NFCs as well as SNFCs. In R_C^s, the leaf nodes' entries (the client points) store not only the dnn values (radii of the NFCs) but also the $d2nn$ values (radii of the SNFCs). The non-leaf nodes' entries store the MND values as well as the *maximum SNFC distance (MSND)* values for their child nodes. Here, MSND is defined similarly to MND, but bounds the SNFCs instead of NFCs (cf. Fig. 9).

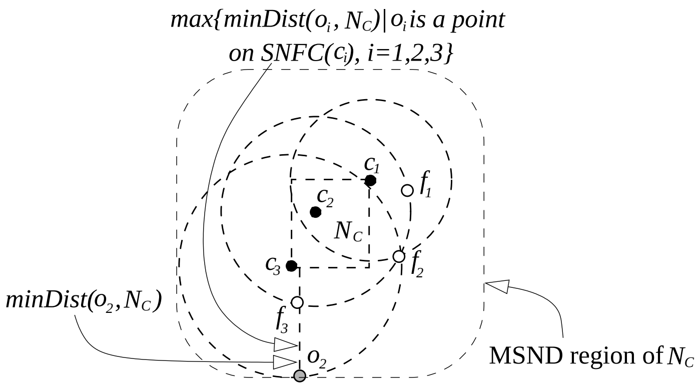


Fig. 9 Example of MSND

We need to do a range query for every pair of facility and location. We batch process the range queries to accelerate query processing. Specifically, we read in the facilities and potential locations in blocks. For each block, we compute a minimum bounding rectangle. Then, for every pair of facility and potential location blocks, we use their bounding rectangles in the range query on R_C^s to find all the relevant clients and therefore reduce the number of tree accesses.

4.3 Replacement Influence Distance Method

The MSND method uses range queries to reduce the search space from $F \times P \times C$ to $F \times P \times \widetilde{C}$, where $\widetilde{C}$ denotes a subset of C that are accessed by the range queries. This is a search cubically proportional to the size of the datasets, which is large. Therefore, MSND is better than Sequential Scan but still expensive. In this subsection, we propose to replace such a cubical search with two lightweight quadratic search plus a lightweight cubical search, so that we can obtain better query processing efficiency. The key is to examine only a small number of promising facility-location pairs. To achieve this, we first compute the dr values for the facilities and potential locations separately (two quadratic search), and then only aggregate the dr values for the facility-location pairs where necessary (a lightweight cubical search) to determine the optimal pair. We use a concept called the *Replacement Influence Distance (RID)* to help identify the facility-location pairs that require dr value aggregation. Therefore, we named the method based on RID the *Replacement Influence Distance (RID)* method.

The motivation behind the RID method is that, instead of considering a pair of facility and potential location $\langle f, p \rangle$ as a unit, we consider it as two independent parts and compute $dr(f)$ and $dr(p)$ separately. This way, we can apply the MND method to efficiently compute $dr(f)$ and $dr(p)$, and only pair up the *promising* facilities and potential locations to find the optimal pair. Here $dr(f)$ denotes the distance reduction incurred by removing f, i.e., $dr(f) = \sum_{c \in IS(f)}(dnn(c,F) - d2nn(c,F))$, where $IS(f)$ is the subset of clients attracted by f (note that $dr(f) \le 0$). Now the problem is to determine which are the promising facilities and potential locations. A pair of facility and potential location $\langle f,p\rangle$ with large $dr(f)$ and $dr(p)$ values might be promising, but this is not always the case because $dr(\langle f,p\rangle)$ is not always simply the sum of $dr(f)$ and $dr(p)$. We need to determine which pairs satisfy $dr(\langle f,p\rangle) = dr(f) + dr(p)$ and which pairs do not, and perform range queries to compute the dr values for the latter case. RID supports this determination process.

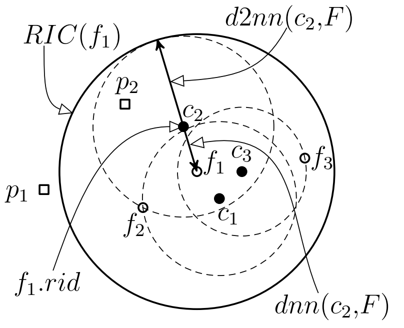


Fig. 10 Example of replacement influence circle

The RID of a facility f is defined as the the largest dnn and $d2nn$ sum of the clients attracted by f. Formally, $f.rid = \max\{dnn(c,F) + d2nn(c,F) | c \in IS(f)\}$. The RID of f defines the *Replacement Influence Circle (RIC)* of f, denoted by $RIC(f)$, which is a circle centered at f with its radius being $f.rid$ (cf. Fig. 10). If a potential location p is not enclosed by $RIC(f)$, then $dr(\langle f,p\rangle) = dr(f) + dr(p)$, as guaranteed by the following Theorem 4.

Theorem 4 *Given a pair of facility and potential location* $\langle f,p\rangle$*,* $dr(\langle f,p\rangle) = dr(f) + dr(p)$ *if* $dist(f,p) > f.rid$*.*

Proof The proof is straightforward. The RIC of f divides the data space into two parts. Inside $RIC(f)$, the clients' new facilities after removing f are irrelevant to any object outside $RIC(f)$, and vice versa. If $dist(f,p) > f.rid$, then p is outside $RIC(f)$. We can then use $dr(f)$ to compute the distance reduction grained for the clients *inside* $RIC(f)$, and $dr(p)$ to compute the distance reduction grained for the clients *outside* $RIC(f)$, separately. Formally,

$$\begin{aligned}
dr(\langle f,p\rangle) &= \textstyle\sum_{c \in IS(\langle f,p\rangle)}(dnn(c,F) - dnn(c, F \setminus \{f\} \cup \{p\})) \\
&= \textstyle\sum_{c \in IS(\langle f,p\rangle) \cap RIC(f)}(dnn(c,F) - dnn(c, F \setminus \{f\} \cup \{p\})) \\
&\quad + \textstyle\sum_{c \in IS(\langle f,p\rangle) \setminus RIC(f)}(dnn(c,F) - dnn(c, F \setminus \{f\} \cup \{p\})) \\
&= \textstyle\sum_{c \in IS(f)}(dnn(c,F) - d2nn(c,F)) \\
&\quad + \textstyle\sum_{c \in IS(p)}(dnn(c,F) - dist(c,p)) \\
&= dr(f) + dr(p)
\end{aligned}$$

Now we can use an R-tree variant R_F^r to index the facilities and the bounding regions of their RICs as what we have done to index the clients and the bounding regions of the

NFCs. We use another R-tree variant R^r_P to index the potential locations, and then perform a synchronous traversal on the two trees to quickly identify all the facility-location pairs that require dr aggregation.

We further enhance the efficiency of the algorithm by storing $dr(f)$ in R^r_F and $dr(p)$ in R^r_P. For the leaf nodes, the entries store the facilities (or potential locations) as well as their respective dr values. For the non-leaf nodes, we store with an entry the largest dr value of the facilities (or potential locations) indexed in the child node of the entry, denoted by $maxdr$. Then we can perform the synchronous traversal on R^r_F and R^r_P in a branch and bound fashion. We start the traversal at the root nodes. For every pair of nodes accessed, we estimate an upper bound dr_u of the dr value for each pair of the entries $\langle e^r_f, e^r_p \rangle$ using the following equation.

$$dr_u(\langle e^r_f, e^r_p \rangle) = \begin{cases} maxdr(e^r_p), mindist(e^r_f, e^r_p) \le e^r_f.rid & \text{(i)} \\ maxdr(e^r_f) + maxdr(e^r_p), \text{ otherwise} & \text{(ii)} \end{cases}$$

In this equation, case (i) says, if the $minDist$ of e^r_f and e^r_p is less than or equal to $e^r_f.rid$, then according to Theorem 4, $dr(\langle f, p \rangle)$ of two objects f and p indexed in the subtrees of e^r_f and e^r_p does not equal to $dr(f) + dr(p)$. In this case, the best situation is that the removal of f does not incur an increase of the dnn value of any client, and thus $dr(\langle f, p \rangle)$ is solely determined by $dr(p)$, i.e., $dr(\langle f, p \rangle) = dr(p)$. Therefore, an upper bound of the dr value for e^r_f and e^r_p is computed as $maxdr(e^r_p)$. In case (ii), the $minDist$ of e^r_f and e^r_p is larger than $e^r_f.rid$. By Theorem 4 we have an upper bound of the dr value for e^r_f and e^r_p computed as $maxdr(e^r_f) + maxdr(e^r_p)$.

Algorithm 5: RID($N^r_F, N^r_P, R^s_C, optPair$)

```
1  if $N^r_F$ and $N^r_P$ are non-leaf nodes then
2      for $(e^r_f, e^r_p) \in N^r_F \times N^r_P$, $dr_u(\langle e^r_f, e^r_p \rangle) > optPair.dr$ do
3          RID($e^r_f.childnode$, $e^r_p.childnode$, $R^s_C$, $optPair$);
4  else if $N^r_F$ is a leaf node and $N^r_P$ is a non-leaf node then
5      for $e^r_p \in N^r_P$, $dr_u(\langle N^r_F, e^r_p \rangle) > optPair.dr$ do
6          RID($N^r_F$, $e^r_p.childnode$, $R^s_C$, $optPair$);
7  else if $N^r_F$ is a non-leaf node and $N^r_P$ is a leaf node then
8      for $e^r_f \in N^r_F$, $dr_u(\langle e^r_f, N^r_P \rangle) > optPair.dr$ do
9          RID($e^r_f.childnode$, $N^r_P$, $R^s_C$, $optPair$);
10 else
11     for $(e^r_f, e^r_p) \in N^r_F \times N^r_P$, $dr_u(\langle e^r_f, e^r_p \rangle) > optPair.dr$ do
12         if $mindist(e^r_f, e^r_p) > e^r_f.rid$ then
13             $dr = dr_u(\langle e^r_f, e^r_p \rangle)$;
14         else
15             $dr$ = PointQuery($e^r_f, e^r_p, R^s_C$);
16         if $dr > optPair.dr$ then
17             $optPair \leftarrow \langle e^r_f, e^r_p, dr \rangle$;
```

We only need to visit the child nodes of the entry pairs whose dr_u values are larger than the optimal dr value found so far, $optPair.dr$. When the traversal reaches the leaf nodes, we issue range queries on R^s_C for the unpruned facility-location pairs to compute the actual

dr values, and update $optPair.dr$. We perform the traversal in a depth-first order so that $optPair.dr$ can be early updated and thus enhance the pruning capability of the algorithm, as summarized in Algorithm 5.

The only problem left is how to efficiently initialize the R-tree variants, R^r_F R^r_P and R^s_C. The R-tree variants are first built with the extra distance metric fields left uninitialized. This is a standard R-tree construction process. Then the extra distance metric fields are initialized as follows. First, an R-tree join like operation is performed on R^r_F and R^s_C to precompute dnn and $d2nn$ for the clients, as well as $dr(f)$ and RID for the facilities all together. The distance metric fields in the non-leaf nodes are initialized during the same process in a bottom up fashion using the technique proposed for the efficient computation of MND. The MND algorithm is then applied to precompute $dr(p)$ for the potential locations in R^r_P and initialize the distance metric fields in the non-leaf nodes in R^r_P, also in a bottom up fashion. The extra cost of the precomputation is very limited as shown in the experiments that verify the efficiency of the MND algorithm ($\leq$ 1 second for 1,000,000 data points, cf. Section 6.2). Once the R-tree variants are set up, they can be maintained incrementally with data updates.

5 Cost Analysis

In this section, we analytically compare for all described methods (SS, QVC, NFC, MND, SS-FR, MSND and RID) the precomputation cost, I/O cost, and CPU cost. Table 3 summarizes the 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.

Table 3 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^n_C, R_P	$(1 - w_n)\frac{n_c n_p}{(C_e - 1)^2}$
MND	dnn	R^m_C, R_P	$(1 - w_m)\frac{n_c n_p}{(C_e - 1)^2}$
SS-FR	dnn, $d2nn$	N/A	$\frac{n_f n_p n_c}{C_m^3}$
MSND	dnn, $d2nn$	R^s_C	$(1 - w_s)\frac{n_f n_p n_c}{C_m^2 (C_e - 1)}$
RID	dnn, $d2nn$ dr, RID	R^s_C, R^r_F, R^r_P	$(1 - w_r)\frac{n_f n_p n_c}{(C_e - 1)^3}$

5.1 Precomputation and Index Cost

We precompute $dnn(c, F)$ for all location selection methods. Computing $dnn(c, F)$ for all clients has the cost of $O(n_c n_f)$ 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. We precompute $dnn(c, F)$ and $d2nn(c, F)$ for the facility replacement methods SS-FR and MSND. Since it is done at the same time, the cost is very similar to that of precomputing $dnn(c, F)$ only. The RID method precomputes more distance metrics (i.e., $dr(f)$, $dr(p)$ and RID), but uses a different computation algorithm, which on average have much lower cost than the nested loop based dnn ($d2nn$) computation used by the other methods. Meanwhile, these values are incrementally maintained. Thus, the cost is amortized.

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. MSND and RID both use an R-tree variant R_C^s, while RID also indexes F and P with two R-tree variants R_F^r and R_P^r, respectively. 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, for the location selection query, 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. For the facility replacement query, MSND maintains an R-tree variant while RID maintains three. SS-FR and MSND have the same distance metric precomputation costs. RID computes more distance metrics, but the computation is more efficient due to the use of the R-tree variants.

5.2 I/O Cost

For SS, the data points are retrieved in blocks from the disk; the I/O costs is $IO_s = \frac{n_p}{C_m}\frac{n_c}{C_m} = \frac{n_p n_c}{C_m^2}$. For the other location selection methods, they involve R-tree (and variant) traversals. 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, all nodes are paired up and traversed. Therefore, the worst-case I/O costs for these two methods are the same: $\frac{n_c}{C_e-1}\frac{n_p}{C_e-1} = \frac{n_c n_p}{(C_e-1)^2}$. In average case, some of the nodes are pruned from traversal. We quantify the percentage of pruned nodes 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, which is associated with the metrics used in the determination of the influence sets, i.e., $dnn(c, F)$ and MND, respectively. Since MND is defined to approximate $dnn(c, F)$ closely, the pruning power quantifiers have similar values, i.e., $w_m \approx w_n$ 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 $QVC(p)$: 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 $QVC(p)$, 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.

For the facility replacement methods, SS-FR retrieves all data points in blocks and its I/O cost $IO_{sf} = \frac{n_f}{C_m}\frac{n_p}{C_m}\frac{n_c}{C_m} = \frac{n_f n_p n_c}{C_m^3}$. MSND examines every pair of facility and potential location. Thus, its I/O cost $IO_{ms} = \frac{n_f}{C_m}\frac{n_p}{C_m} IO_{ms}^q = \frac{n_f n_p}{C_m^2} IO_{sf}^q$, where IO_{ms}^q denotes the I/O cost of the range query required for dr computation. For the range query, we need to retrieve the nodes in the client R-tree R_C^s that satisfy certain conditions with the bounding rectangles of a facility block and a potential location block. In the worst case, every node of R_C^s is traversed. The I/O cost is $\frac{n_f n_p}{C_m^2}\frac{n_c}{C_e - 1} = \frac{n_f n_p n_c}{C_m^2 (C_e - 1)}$. This worst-case I/O cost is slightly worse than the I/O cost of SS-FR. Meanwhile, some of the nodes may be pruned during the range query. We quantify the percentage of the pruned nodes as the pruning power, denoted by w_s. The number of nodes accessed is then $(1 - w_s)\frac{n_c}{C_e - 1}$, and the I/O cost becomes $(1 - w_s)\frac{n_f n_p n_c}{C_m^2 (C_e - 1)}$. This I/O cost can be very close to the worst-case I/O cost because the bounding rectangles are usually large, since they bound randomly grouped facilities and potential locations.

RID method further reduces the number of facility and potential locations accessed by performing a branch and bound based traversal on R_F^r and R_P^r. Its I/O cost IO_r is denoted as $(1 - w_r)\frac{n_f}{C_e - 1}\frac{n_p}{C_e - 1}\frac{n_c}{C_e - 1} = (1 - w_r)\frac{n_f n_p n_c}{(C_e - 1)^3}$, where w_r quantifies the pruning power of the branch and bound traversal as well as the range query on R_C^s. Note that even though the range queries of the RID method are on facility-location pairs instead of block pairs, which means there may be more I/Os for the range query part, the advantage of RID over MSND is still explicit because of the high pruning capability of the branch and bound traversal. Thus, we have $IO_r < IO_{ms} \approx IO_{sf}$.

5.3 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 $IO_q = IO_{q1} + IO_{q2} + IO_{q3}$. 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 $QVC(p)$ and t_{q3} indicates the CPU cost per pair of $QVC(p)$ 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 the MBR of $QVC(p)$ or $N_P.mbr$, respectively. Due to the additional QVC 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. The same conclusion applies to the SS-FR and the MSND methods, while MSND method has comparatively smaller CPU cost per pair of facility and potential location because the range query used help prune many clients from distance comparison for dr computation. Therefore, MSND has smaller overall CPU cost (i.e., $CPU_{ms} < CPU_{sf}$).

For RID, the distance metrics computed include (i) $minDist$, for determining which equation to use for dr upper bound computation and further which pair of nodes to be accessed next; (ii) $dist(c, f)$ and $dist(c, p)$, for actual dr value computation. The first part is similar to the computations in the MND method, which is small because it is for per pair of blocks. The second part is similar to the range query of the MSND method. Meanwhile, IO_r is much smaller. Therefore, the CPU cost of RID (CPU_r) is small compared to the other facility replacement methods.

In summary, we have $CPU_s > CPU_q > CPU_m \approx CPU_n$ and $CPU_{sf} > CPU_{ms} > CPU_r$. These inequalities will be validated by experiments in the next section.

6 Experimental Results

This section reports our experimental results. We start with the experimental settings in Section 6.1. Then we present experiments on the algorithms proposed for the location selection and facility replacement queries in Sections 6.2 and 6.3, respectively.

6.1 Experimental Setup

All experiments were conducted on a desktop PC with 3GB RAM and 2.66GHz Intel$^{®}$ 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 1,000,000. 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 4, where values in bold denote default values.

Table 4 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

We use two groups of real datasets provided by Digital Chart of the World [20], 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 [9] (or its variants as proposed in this paper) as the underlying access methods.

We evaluate the performance of four methods for the min-dist location selection query:

- SS, where the potential locations and the clients are sequentially scanned for dr value computation.
- QVC, where quasi-Voronoi cells are used to reduce the search space for dr value computation.
- NFC, where nearest facility circles are used to reduce the search space for dr value computation and the dr values of different potential locations are computed synchronously.
- MND, where MND regions are used to reduce the search space for dr value computation and the dr values of different potential locations are computed synchronously.

We evaluate the performance of three methods for the min-dist facility replacement query:

- SS-FR, where the facilities, the potential locations and the clients are sequentially scanned for dr value computation.
- MSND, where MSND regions are used to reduce the search space for dr value computation.
- RID, where RID values are used to reduce the number of facility-potential location pairs required to be checked to find the optimal pair and MSND regions are used to reduce the search space for dr value computation.

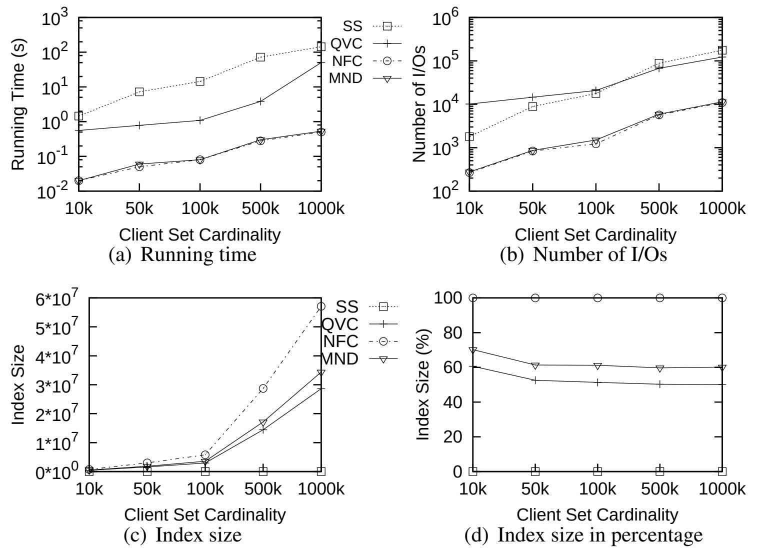
6.2 The Min-dist Location Selection Query

In this subsection, we show that, for the min-dist location selection query, MND is the only method that performs as good as NFC in terms of the running time and the number of I/Os, while MND has a much smaller index size.

6.2.1 Varying Dataset Cardinality

The results for the experiments that vary the number of clients are shown in Fig. 11. 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. 11(a) and (b)). Meanwhile, the MND method has a much smaller index size compared to the NFC method (cf. Fig. 11(c)). Fig. 11(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.

From Fig. 11, 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



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

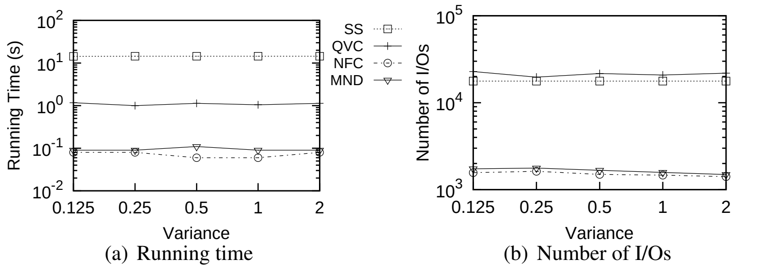
Fig. 11 The effect of client set size

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.

We have also conducted experiments that vary the number of facilities and potential locations. The results are similar to the above and thus omitted. See reference [18] for more details.

6.2.2 Varying Dataset Distribution

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. 12 The effect of σ^2 in Gaussian distribution

Fig. 12 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 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.

6.2.3 Experiments on Real Datasets

The experimental results on real datasets are shown in Fig. 13. 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 other methods.

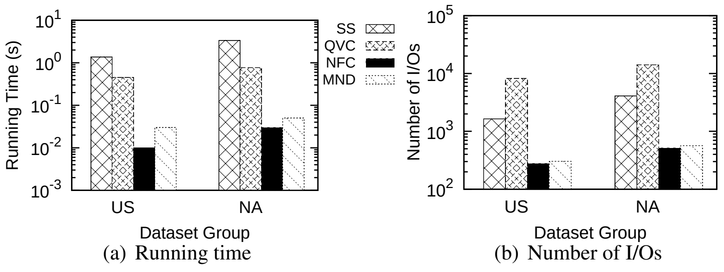


Fig. 13 Performance comparison on real datasets

6.3 The Min-dist Facility Replacement Query

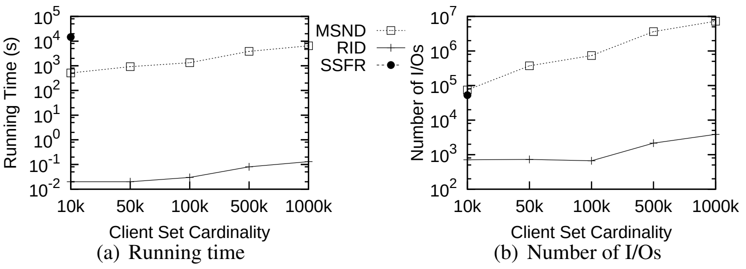


Fig. 14 The effect of client set size

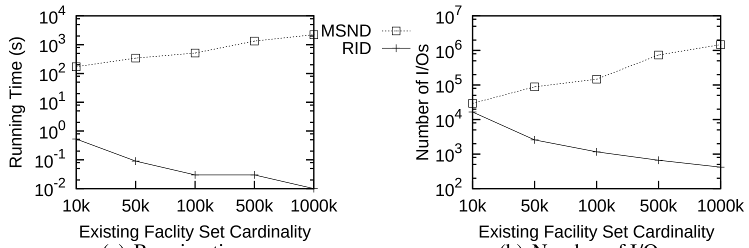
In this subsection, we compare the performance of the facility replacement methods, i.e., SS-FR, MSND and RID. We show that RID constantly outperforms SS-FR and MSND under various settings by up to five orders of magnitude.

6.3.1 Varying Dataset Cardinality

We first vary the size of the client dataset. As Fig. 14 shows, when the number of clients increases, the running time and I/O for all methods increase. The advantage of RID is clear, and the superiority is up to five orders of magnitude. It can process the query within 0.1 seconds for very large datasets (e.g., $|C| = 1,000,000$), while neither SS-FR nor MSND can process the query within 100 seconds for much smaller datasets (e.g., $|C| = 10,000$). In particular, SS-FR requires $14,676$ seconds (≈ 4 hours) to process a dataset of $10,000$ clients. Since SS-FR scans every client, its running time increases linearly with the number of clients. It will need 20 hours to produce a query answer for a dataset of $50,000$ clients. Due to this extremely low efficiency, we omit SS-FR when $|C|$ is larger than $10,000$ as well as in the following experiments, and focus on the comparison between MSND and RID. We also observe that the performance difference on I/O is relatively small compared with that on running time. When $|C| = 10,000$, MSND even has slightly higher I/O cost than SS-FR does. This confirms our cost analysis in Section 5.

We also perform experiments where the facility set size and the potential location set size are varied. From Fig. 15 we can see that RID outperforms MSND constantly in terms of both running time and I/O when the facility set size is varied. An interesting observation is that, while the costs of MSND increase with the dataset sizes because MSND accesses every pair of facility and potential location, the costs of RID decrease. This is because when the number of facilities increases, the dnn and $d2nn$ values of the clients decrease and as a result, the facilities have smaller RID regions, which enhances the pruning capability of the method and hence the algorithm performance.

We omit the result of varying potential location set size because it is very similar to that of varying client set size.



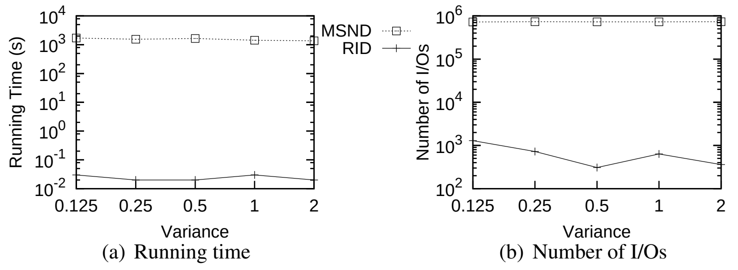
(a) Running time (b) Number of I/Os

Fig. 15 The effect of existing facility set size

6.3.2 Varying Dataset Distributions

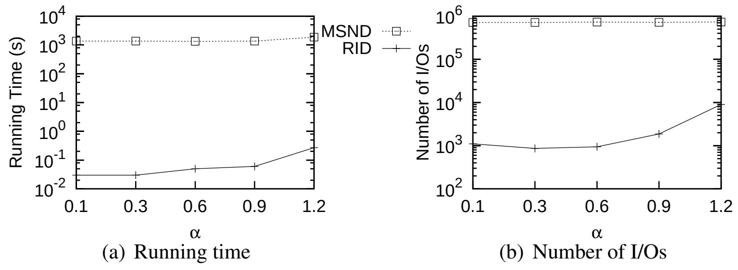
Next, we vary the data distribution by varying the values of σ^2 and α in the Gaussian and Zipfian datasets, respectively. From Figs. 16 and 17 we can see that RID again outperforms MSND in datasets with different distributions by several orders of magnitude. While data distribution does not affect much of the performance of MSND, the performance of RID varies when data distribution changes. This is explained as follows. Altering the values of

σ^2 and α effectively alters the skewness of the data distribution, which has a two-fold effect on the algorithm performance. On one hand, more skewed data results in lower selectivity of the range queries needed by MSND and RID. On the other hand, more skewed data also means smaller dnn and $d2nn$ values as well as RID regions, which results in higher selectivity of the range queries. The combined effect depends on whether the former or the latter is the dominating factor. As the figures show, the combined effect on the MSND method is more balanced while it is varying on the RID method.



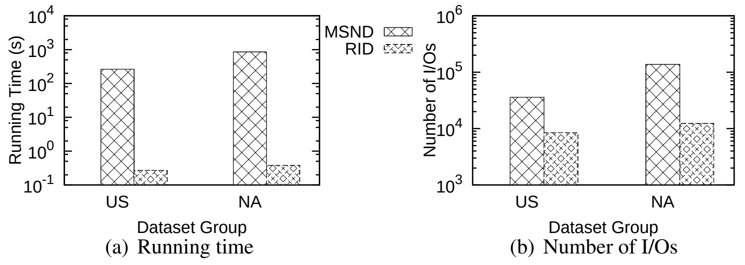
(a) Running time (b) Number of I/Os

Fig. 16 The effect of σ^2 in Gaussian distribution



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

Fig. 17 The effect of α in Zipfian distribution



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

Fig. 18 Performance comparison on real datasets

6.3.3 Experiments on Real Datasets

We also evaluate the performance of MSND and RID on real datasets. The result in Fig. 18 again confirms the superiority of the RID method over the MSND method in terms of both running time and I/O. The comparative performance of the two method is similar to that of the experiments on synthetic datasets.

7 Conclusions

We formulated the min-dist location selection problem and an important variant, the min-dist facility replacement problem. For the location selection problem, 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. For the facility replacement problem, we first apply the MND method to achieve an effective solution called the MSND method. To obtain even better efficiency, we transform the facility replacement operation into a two-stage operation, and thus significantly reduce the search space, which results in a highly efficient method called RID. 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 and the RID method.

Acknowledgements This work is supported by the Australian Research Council's Discovery funding scheme (project numbers DP130104587 and DP130103705). Rui Zhang is supported by the Australian Research Council's Future Fellow funding scheme (project number FT120100832).

References

1. ArcGIS: http://www.esri.com/ (2013)
2. Aurenhammer, F.: Voronoi diagrams - a survey of a fundamental geometric data structure. ACM CS **23**, 345–405 (1991)
3. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: SIGMOD, pp. 237–246 (1993)
4. Cabello, S., Díaz-Báñez, J.M., Langerman, S., Seara, C., Ventura, I.: Reverse facility location problems. In: CCCG, pp. 68–71 (2005)
5. Delfos, J., Tan, T., Veenendaal, B.: Design of a web-based lbs framework addressing usability, cost, and implementation constraints. World Wide Web **13**, 391–418 (2010)
6. Deng, K., Xu, H., Sadiq, S., Lu, Y., Fung, G.P.C., Shen, H.T.: Processing group nearest group query. In: ICDE, pp. 1144–1147 (2009)
7. Du, Y., Zhang, D., Xia, T.: The optimal-location query. In: SSTD, pp. 163–180 (2005)
8. Gao, Y., Zheng, B., Chen, G., Li, Q.: Optimal-location-selection query processing in spatial databases. TKDE **21**, 1162–1177 (2009)
9. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
10. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: SSD, pp. 83–95 (1995)
11. Huang, J., Wen, Z., Qi, J., Zhang, R., Chen, J., He, Z.: Top-k most influential locations selection. In: CIKM (2011)
12. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. PVLDB **1**(1), 1068–1080 (2008)
13. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: SIGMOD, pp. 201–212 (2000)
14. Mouratidis, K., Papadias, D., Papadimitriou, S.: Medoid queries in large spatial databases. In: SSTD, pp. 55–72 (2005)
15. Nutanong, S., Tanin, E., Zhang, R.: Incremental evaluation of visible nearest neighbor queries. TKDE **22**(5), 665–681 (2010)
16. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: The v*-diagram: a query-dependent approach to moving knn queries. PVLDB **1**(1), 1095–1106 (2008)
17. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: Analysis and evaluation of v*-*k*nn: an efficient algorithm for moving *k*nn queries. VLDB J. **19**(3), 307–332 (2010)
18. Qi, J., Zhang, R., Kulik, L., Lin, D., Xue, Y.: The min-dist location selection query. In: ICDE, pp. 366–377 (2012)

19. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD, pp. 71–79 (1995)
20. RtreePortal: http://www.chorochronos.org/ (2013)
21. Stanoi, I., Riedewald, M., Agrawal, D., Abbadi, A.E.: Discovery of influence sets in frequently updated databases. In: VLDB, pp. 99–108 (2001)
22. Tao, Y., Papadias, D., Lian, X.: Reverse knn search in arbitrary dimensionality. In: VLDB, pp. 744–755 (2004)
23. Wong, R.C.W., Özsu, M.T., Yu, P.S., Fu, A.W.C., Liu, L.: Efficient method for maximizing bichromatic reverse nearest neighbor. PVLDB **2**, 1126–1137 (2009)
24. Wu, W., Yang, F., Chan, C.Y., Tan, K.L.: Continuous Reverse k-Nearest-Neighbor Monitoring. In: MDM (2008)
25. Xia, T., Zhang, D., Kanoulas, E., Du, Y.: On computing top-t most influential spatial sites. In: VLDB, pp. 946–957 (2005)
26. Xiao, X., Yao, B., Li, F.: Optimal location queries in road network databases. In: ICDE, pp. 804–815 (2011)
27. Yang, C., Lin, K.I.: An index structure for efficient reverse nearest neighbor queries. In: ICDE, pp. 485–492 (2001)
28. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbors in large graphs. TKDE **18**, 540–553 (2006)
29. Yu, C., Zhang, R., Huang, Y., Xiong, H.: High-dimensional knn joins with incremental updates. Geoinformatica **14**, 55–82 (2010)
30. Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive computation of the min-dist optimal-location query. In: VLDB, pp. 643–654 (2006)
31. Zhang, R., Jagadish, H.V., Dai, B.T., Ramamohanarao, K.: Optimized algorithms for predictive range and knn queries on moving objects. Inf. Syst. **35**(8), 911–932 (2010)
32. Zhang, R., Lin, D., Kotagiri, R., Bertino, E.: Continuous intersection joins over moving objects. In: ICDE, pp. 863–872 (2008)
33. Zhang, R., Qi, J., Lin, D., Wang, W., Wong, R.C.W.: A highly optimized algorithm for continuous intersection join queries over moving objects. VLDBJ **21**, 561–586 (2012)