Processing of Top-K Most Influential Location Selection Queries

Rui Zhang, Jin Huang, Zeyi Wen
Department of Computing and Information Systems
University of Melbourne, Australia
rui@csse.unimelb.edu.au

Jian Chen
School of Software Engineering
South China University of Technology, China

Kerry Taylor
Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia

Zhen He
Department of Computer Science
La Trobe University, Australia

A Technical Report

June 2013

Abstract

Facility location selection queries help to evaluate the popularity of different facility locations for a to-be-added facility. Such queries have wide applications in marketing and decision support systems. In this report, we propose and investigate a new type of queries aiming to retrieve the *top-k most influential locations* from a candidate set in a given context of customers and existing facilities. The *influence* in the query, which models the potential popularity of the new facility, is defined as the number of *reverse nearest* customers the new facility can attract if it was added. Specifically, given a candidate set C, an existing facility set F, and a customer set M, the proposed query returns the top-k candidates in C with the greatest influences. The most naive solution for the query employs sequential scan on all data sets and is thus expensive and not scalable to large data sets. To improve the solution, two R-Tree based branch-and-bound algorithms are presented. One of them, named *Estimation Expanding Pruning (EEP)*, uses distance metrics between nodes to tighten the search space, while the other, named *Bounding Influence Pruning (BIP)*, relies on half plane styled geometric properties to achieve the same goal. Both algorithms follow the best-first access strategy guided by "hints" computed during the pruning and meanwhile gradually refine these "hints". BIP generally outperforms EEP since it avoids repeated estimations on F and M. Yet due to the extensively accesses on R-tree indexes, the complexity in the worst case of both algorithms is unsatisfactory, causing their performance to degrade dramatically when the data set grows. To achieve better scalability, an algorithm named *Nearest Facility Circle (NFC)* is proposed. Rather than computing all the influence relationships from scratch as EEP and BIP, NFC first pre-computes the influence relationships between customers and existing facilities, then indexes these relationships with an R-Tree, finally processes the query using multiple cheap point enclosure queries. Furthermore, a *NFC join (NFCJ)* algorithm is propose to construct an R-tree on candidate set and share the common traversal cost of point enclosure query by using R-tree join algorithm. We theoretically and experimentally compare all proposed algorithms. The results show that NFCJ is the the best solution for the proposed query.

Chapter 1

Introduction

A common problem for many business and organizations is to find a suitable place to establish *a new facility*. For instance, McDonald's may want to introduce a new restaurant into a booming community to compete with other fast food restaurants. A wireless carrier may want to construct a new base station or hotspot for wireless Internet access to a densely populated area to improve its service quality. A scientific organization may want to select a location for a new environmental sensor to capture particular mobile wildlife as the materials for research. A city planner may want to find where to introduce a new public infrastructure such as a drop-in clinic to a flourishing suburb. In most cases, the selection of locations must be made from *a given candidate set*, e.g., trulia.com lists more than 43,100 locations for sale in Los Angele, CA, USA [30] and Soufun.com lists more than 339,330 locations for rental in Beijing, China [26]. For business directors, one of the most important indicators used to evaluate a candidate location is the number of customers the newly added facility could attract. In this report, we investigate the problem of finding the top-k candidate locations that attract the largest number of customers, where k is a user define integer. The top-k locations are of interest because in real applications there are additional factors such as safety and popularity of a brand in a region (for example, fast food is relatively unpopular in suburbs with mostly European habitants). These may be important factors that will affect the decision but are difficult to quantify. Therefore, top-k results returned by the query can serve as the primary candidates based on which further consideration can be made. In this study, we assume a customer is attracted by his or her nearest facility and the business has the knowledge of customer and existing facility distributions from surveys or past sales records.

An example corresponding to the above problem is shown in Figure 1.1, where circles and squares represent customers and facilities, respectively. To distinguish existing facility locations and candidate locations, we use white squares to denote the existing facilities and black squares to denote the candidate facilities. In the figure, the candidate locations are labeled as $c_1, c_2, c_3, ..., c_6$. A customer is connected to a candidate location with a dashed line if and only if

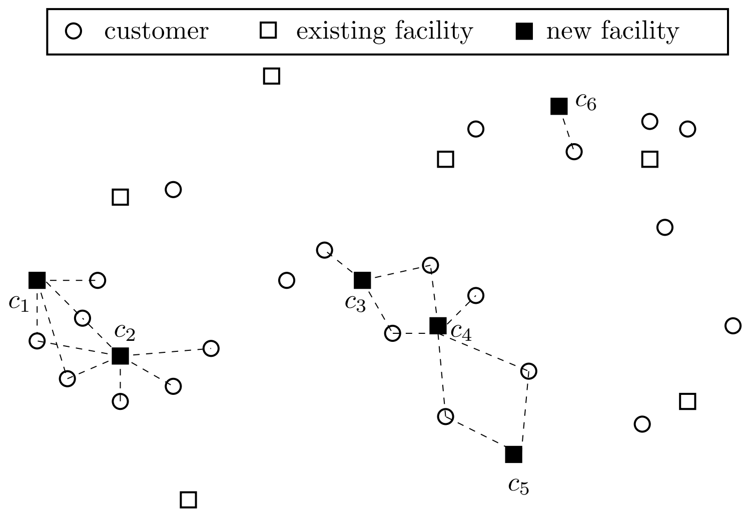


Figure 1.1: $c_1, c_2, c_3, c_4, c_5, c_6$ are of influence values $4, 6, 3, 5, 2, 1$, respectively.

the customer would be influenced by a new facility established at that candidate location. For each candidate location c_i, the number of customers it attracts can be computed by counting the number of customers connected to it with a dashed line. In this example, the numbers corresponding to $c_1, c_2, c_3, c_4, c_5, c_6$ are $4, 6, 3, 5, 2, 1$. If user inputs three as the parameter k, then the query would return the answer set c_2, c_4, c_1.

Please note the number of customers an existing facility location can attract may be reduced by a newly added facility. This follows the idea of competition as McDonald's may want to attract customers from other restaurants. Therefore in this case the existing facilities may represent a competing company like KFC. Another example is that a wireless carrier adds a new base station to take load off existing base stations since existing ones are out of capacity or ill-balanced. After adding the new facility, this situation can be improved. In an extreme case, a company may even consider replacing the existing facility with the new facility due to the maintenance cost of keeping the existing facility. In all scenarios, our method can easily address the requirements.

The aforementioned facility location selection problem aims at maximizing the *influence* of the new facilities, where the influence is defined as the number of customers who perceive the new facility as their nearest facility. In a previous poster paper [15], we formulate the above described problem as the top-k most influential location selection query. In this report, we detail the solutions for answering this query and evaluate their performance with experiments and analysis.

The remainder of the report is organized as follows. Chapter 2 defines the related concepts and top-k most influential location selection query. Chapter 3 reviews previous studies on related topics. Chapter 4 progressively describes sequential scan, estimation expanding pruning, bounding influence pruning, and nearest facility location algorithms with the analysis on their complexities. Chapter 5 presents the experimental results. Finally, the report is concluded in Chapter 6.

Chapter 2

Preliminary Concepts and Studied Problem

In this chapter, we will first introduce the related concepts and definition of location influence, based on which we then propose a novel query to select the optimal location for a new facility.

2.1 Location Influence

We present formal definition of reverse nearest neighbor and location influence first. Table 2.1 lists frequently used symbols in the report.

Definition 1 (Reverse Nearest Neighbors)**.** *The customers who perceive a facility as their nearest facility are* reverse nearest neighbors *of this facility. Let $d(f, m)$ denote the Euclidean distance between f and m, $min(m, F)$ denote the minimum distance between m and any $f \in F$, $f.RNN(F, M)$ denote the reverse nearest neighbors of $f \in F$, then $f.RNN(F, M) = \{m \in M | d(m, f) = min(m, F)\}$.*

Note this definition presents the bichromatic version of the reverse nearest neighbors problem, where objects are divided into two categories. The reverse nearest neighbors of an object in such scenario always come from the opposite category [16].

When evaluating the popularity of a facility, counting the number of its reverse nearest neighbors would be a sensible indicator because in many scenarios such as marketing and city planning, specific individuals are of less interest than the overall number of customers. We have:

Definition 2 (Location Influence)**.** *The* influence *of a location is the number of its reverse nearest neighbors. Let I_f be the influence of facility f, then $I_f(F, M) = |f.RNN(F, M)|$.*

Table 2.1: Frequently used symbols

Symbol	**Explanation**
C, F, M	Sets of candidate, existing facility and customer locations, respectively
c, f, m	A candidate location, an existing facility and a customer, respectively
p	A point in the data space
t_C, t_F, t_M	R-trees on C, F and M, respectively
n_C, n_F, n_M	A node in t_C, t_F and t_M, respectively
r_C, r_F, r_M	The MBRs of nodes n_C, n_F, and n_M, respectively
I_f, I_c	The influence of an existing facility and a candidate facility, respectively
I_C^u	The upper bound of influence for all c indexed by a node n_C
I_C^l	The lower bound of influence for all c indexed by a node n_C
I_δ	The k^{th} greatest influence value of candidates seen so far
L_C, L_F, L_M	The priority list of nodes n_C, n_F, and n_M, respectively
$n_C.S_M$, $n_F.S_M$	The unpruned customers of nodes n_C and n_F, respectively
$n_M.S_C$, $n_M.S_F$	The unpruned candidates and existing facilities of node n_M, respectively
$n_C.R$	The influence region of a node n_C
$n_C.S_F$	The relevant existing facilities set of a node n_C
$n_C.S_F^+$	The outer relevant existing facilities set of a node n_C
$n_C.S_F^-$	The inner relevant existing facilities set of a node n_C

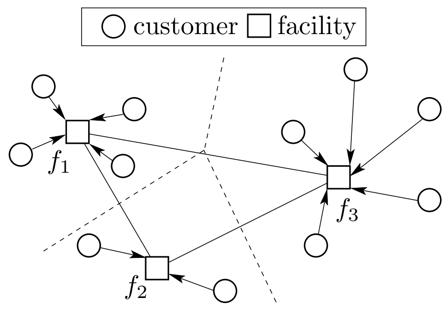


Figure 2.1: $I_{f1} = 4$; $I_{f2} = 2$, $I_{f3} = 5$

Figure 2.1 demonstrates an example of facility influence, where circles denote customers and squares denote facilities. In the figure, the dash lines are perpendicular bisectors between each pair of facilities. Easily, $I_{f1} = 4$, $I_{f2} = 2$, $I_{f3} = 5$.

Here we assume each customer location contributes the same 1 unit influence. Yet, our problem setting can be generalized to take variable units into consideration. For the sake of brevity, we follow the 1 unit setting for the rest of this report.

Instead of focusing on the specific reverse nearest neighbors, Definition 2 can be used as the criteria for ranking facilities based on their attractiveness for various applications. The influence based location problem has wide applications in fields like commercial marketing and community planning. Selecting an optimum location is of great interest when different locations offer diverse potential profits. In addition, the influence location selection problem generally faces massive data sets in the real world where there can be numerous facilities and customers. This calls for an efficient solution to make the query viable for integration into decision making systems.

2.2 Influence Maximization for A New Facility

When selecting the location for a new facility, we aim to find the one with highest potential to be popular among all existing facilities. Using location influence as an indicator, it is possible to select the optimum location for a to-be built facility. Specifically, we can evaluate the potential popularity of a location by *adding a new facility* on it, and then *computing the influence of that new facility.* The influence of the new facility turns out to be an indicator of desirability of the location. Obviously, the more customers the new facility can influence, the better the location. This leads to the following definitions:

Definition 3 (Potential Influence)**.** *Let C denote a candidate set of available locations for a to-be built facility. The potential influence I'_c of a candidate location $c \in C$ is the influence it earns when it is added into the existing facility location set, i.e. $I'_c = I_c(F \bigcup \{c\}, M)$.*

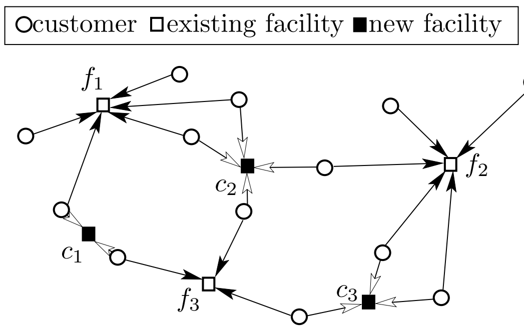


Figure 2.2: $C = \{c_1, c_2, c_3\}$; $I_{c1} = 2$, $I_{c2} = 4$, $I_{c3} = 3$; Top-2 influential candidates are c_2, c_3

In the remainder of the report, when there is no ambiguity, we use I_f to denote $I_f(F, M)$ and I_c to denote $I_c(F \bigcup \{c\}, M)$ for brevity.

Definition 4 (Top-k Most Influential Locations)**.** *Given a set C of candidate locations for the new facility, the top-k most influential locations are k locations in C with the largest* potential influence*.*

If multiple candidate locations have the same potential influence value, there might be ties when selecting the top-k candidates. To resolve this, we arbitrarily choose some of them as the answer. For example, when more than k candidates are of same largest potential value, we simply pick k of them as the answer.

Figure 2.2 gives an example for this potential influence. In the figure, the circles denote customers, the white squares denote existing facilities, black squares denote potential new facilities. As illustrated, if c_1 is added into existing facilities, it can influence two customers from f_1 and f_3 (customers with white arrows pointing to c_1); if c_2 is added, it can influence four customers from f_1, f_2, and f_3; if c_3 is added, it can influence three customers from f_2 and f_3 . Therefore, the potential influence of c_1, c_2, and c_3 are $2, 4, 3$, respectively. According to definition 4, the top-2 most influential locations among C are c_2 and c_3.

Chapter 3

Related Work

3.1 Reverse Nearest Neighbor

Korn and Muthukrishnan [16] first proposed the reverse nearest neighbor (RNN) query and define the RNNs of an object o to be the objects whose respective nearest neighbor is o. In the same paper [16], Korn and Muthukrishnan propose an RNN-tree based solution to the RNN query, where the RNN-tree is an R-Tree [14] variant that indexes nearest neighbor (NN) circles of the data objects rather than the data objects themselves. Here, the NN circle of an object is defined to be a circle that centers at o with its radius being the distance between o and o's nearest neighbor. Based on the NN circles, to find the RNN of an object o only requires checking which objects' NN circles enclose o. Applying this idea to our top-k most influential query gives the NFC algorithm. However, the RNN-tree based solution has two major drawbacks. One is that it requires the extra maintenance of an RNN-tree. The other is that it requires precomputing the NN circles. Therefore, this solution can not handle objects with frequent updates. To solve the first problem, Yang and Lin [37] propose to integrate the NN circle information into an R-Tree, so that the resultant R-Tree can be used to process RNN queries as well as other common types of queries, thus avoiding the maintenance of an extra RNN-tree. To solve the second problem, Stanoi et al. [27] propose an approximation-refinement framework to compute the RNNs on the fly, so that no precomputation is needed. While these methods work well for processing a single RNN query, they are not designed to compute RNNs for a large number of objects at the same time, which is one of the key difficulties in many facility location problems. Thus, the RNN problem can be viewed as a sub problem of the facility location problems. Recent progress on improving the efficiency of answering RkNN query can be found in [29], [34], and [1]. Techniques proposed for similarity (nearest neighbor) search [17, 19, 20, 38, 41] such as bulk loading index construction [3] and pre-computing key function values for similarity search [9] can also be helpful in RNN search.

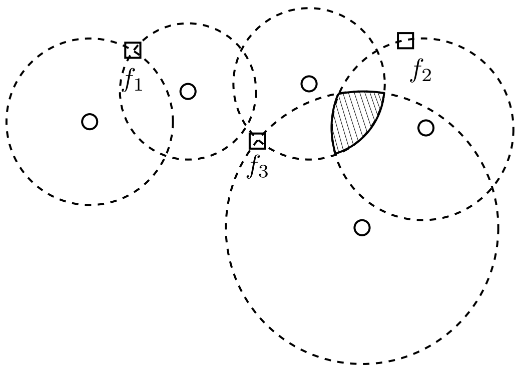


Figure 3.1: Example of the MAXCOV problem

3.2 Location Distance Minimization

Min-dist facility location problems aim to minimize the average distance between customers and their respective nearest facilities. Zhang et al. [39] propose to find an optimal location c in a given region Q such that if a new facility is built on c, the average distance between the customers and their respective nearest facilities is minimized. Mouratidis et al. [18] study the k medoid query and the k median query, which aim at finding a set C' ($C' \subset C$) of k locations from a set C to minimize the average distance between locations in C' and locations in C. Similar to our problem settings, given client set and existing facility set, Qi et al. [22, 23] propose a new min-dist location selection query which aims to pick the best location from a candidate set to minimize the average distance between a client and its nearest facility. Besides also introducing the NFC algorithm to solve the problem, they propose a novel method to answer the query without the need to construct a spatial index in advance. The performance of this method is verified to be close to the best algorithm via extensive experiments. All these studies are distance based optimization problems and are different from our influence based optimization problem because they focus on optimizing the overall performance of all the facilities while our problem focuses on optimizing the performance of one particular facility. Hence, their solutions are not applicable.

3.3 Location Influence Maximization

Max-inf facility location problems aim at maximizing the influence values of the locations, where the influence of a location c is defined by the number of customers c attracts. Cabello et al. [5] propose a facility location problem based on the MAXCOV optimization criterion, which is to find regions in the data space that maximize the numbers of RNNs for the points in these regions. Figure 3.1 gives an example, where the gray region is the optimal region. Points in this region have three RNNs, while any point outside of this region has at

Table 3.1: Existing studies on location influence maximization problem

Study	**Input**	**Output**	**Influence Definition**	**Space**	**Solution**
[5]	M	Regions	Number of RNNs	ℓ_2	NFC
[35]	M, F	Top-k $f \in F$	Number of RNNs	ℓ_2	Branch and bound
[33, 32]	M, F	Regions for a new facility	Number of RNNs	ℓ_p	Region-to-point transformation
[36]	M, F	Regions for a new facility	Number of $m \in M$ within distance of their $(1+\alpha)$NN	ℓ_2	Greedy and grid partitioning
[7, 8]	M, F	Index structure	Number of RkNN	ℓ_2	Voronoi cell
[28]	M, F	Regions for new facilities	Number of RNN constrained by capacity	ℓ_2	NFC and heuristic pruning
[13]	M, F	Network segments for new facilities	Number of RNNs	Spatial network	Branch and bound
[25]	M, F	Top-k $f \in F$	Number of RNNs	Path trajectory	Branch and bound
[42]	M, F	Top-k $f \in F$	Expectation of RNNs	ℓ_2	Branch and bound
[11]	M, F, C as a region	Top $c \in C$	Number of RNNs	ℓ_1	Branch and bound
[12]	M, Q, d	Top $f \notin Q$	Number of $m \in M$ within distance of d	ℓ_2	Branch and bound

most two RNNs. They introduce the concept of *nearest location circle (NFC)* to solve the problem, where the NFC of a customer m is a circle centered at m with its radius being the distance between m and m's existing nearest facility. To find the solution for the MAXCOV criterion based problem is to find the regions that are enclosed by the largest number of NFCs, which requires complex computations. The study give a theoretical analysis, but no efficient algorithm is presented.

Xia et al. [35] propose the top-t most influential sites problem and a branch and bound algorithm to solve it. This problem finds the top-t most influential *existing* sites within a given region Q. It does not consider any candidate locations for **a new** facility. That is to say, in their work, the influence computation is based on all existing facilities, and the influence comparison is between all **existing facilities**. In our work, the influence computation is based on the set obtained by adding each candidate location into the existing facility set, and the influence comparison is between all **candidate locations**. The only possible way to reuse their solution is to first add each candidate into the existing set to compute the top-t influential locations in this new set and then rank all candidate locations by their influences to return the top-k locations. Yet because the added candidate location is not necessarily among the top-t answer of its corresponding new set, the adapted solution cannot guarantee the correct answer unless we set t to the size of the new set, i.e., $|F| + 1$. Hence, there is no straightforward way to adapt their solution to solve our problem.

One highly related problem is also studied in [33]. Yet the expected answer for their most influential location query is **a region** where the new facility being added could earn the same maximum influence values. To achieve an efficient solution, they take advantage of *region-to-point transformation* to tighten the search space dramatically. The proposed method is further extended in [32] to handle similar problem in any $\ell_p - norm$ space of two and three dimensionality. However, since they focus on **selecting the optimal region** rather then **selecting a set of optimal candidates from a given data set**, and the candidates in our problem may not locate in the returned optimal region in their solution, their solutions do not directly apply.

Another similar study is [36], where assumption that customers will only visit their nearest facilities is relaxed such that all facilities within the distance of $(1 + \alpha)d$ from the customer might be visited, where d is the distance between a customer and her nearest neighbor and α is a user input parameter indicating how much further a user would like to travel for a non-nearest facility. Furthermore, the study also gives a greedy solution for finding the k locations for adding k new facilities simultaneously to gain the overall maximum influence. A grid-based technique is used to divide the space and return the grid with highest potential influence as the answer. Again, since the problem setting does not consider the candidate set, applying their method to our problem will not give direct solutions since we would need to return multiple grids until k candidates are found enclosed in these returned grids.

Du et al. [11] propose to find a point from a continuous candidate region that can maximize the influence value. They use ℓ_1 distance and have a strong assumption that all the roads are either horizontal or vertical. We consider ℓ_2 distance, which is a more general problem setting. More importantly, we consider **a candidate location set** instead of **a candidate region**. This is a more practical problem setting because in many real applications, we can only choose from some candidate locations (e.g. a McDonald's restaurant can only be opened at a place for lease or sale, rather than anywhere in a region). Cheema et al. [7] propose to find an influence zone for a query location c, where the customers inside this zone form exactly the *reverse k nearest neighbor (RkNN)* query result of c. Here, a RkNN query retrieves all the data points that have c as one of their k nearest neighbors. They use a method to compute *Voronoi cells* on the fly for the query location to obtain its RkNNs. The proposed method is further rigorously analyzed in [8], and shown to be available when the dimensionality is more than two and there is data update. Compared to this problem, our problem focuses on **the number of RNNs** of the candidate locations instead of **specific locations of the RNNs**. Recently, a similar influential location selection problem with capacity limit is studied [28]. The major difference between this study and the study presented in this paper is twofolds: first, the capacity of facility is out of consideration in the study here, therefore the optimization goal is for the new facility only instead of for all facilities to achieve maximum influence; only facilities in the candidate set is considered here, while the study in [28] attempts to find all locations in the data space that would achieve the optimization. This difference makes the solution there inapplicable to our problem.

While most studies deal with problem in either Euclidean space or $\ell_p - norm$ space, the location influence maximization problem is also studied in spatial networks. Ghaemi et al. [13] tackle the problem where both query objects and sites reside on the spatial network. Shang et al. [25] propose to represent the facility locations by path trajectories. This way, the most accessible locations could be selected based on the number of path trajectories that perceive the location as their nearest facilities. When data sets contain uncertain instances, Zheng et al. [42] formulate the most influential locations as those with highest expected ranks. To efficiently answer the query, the authors propose several pruning rules and a divide-and-conquer paradigm to eliminate search space in terms of locations to be computed and the number of possible worlds needed to be checked. Clearly, none of these studies consider an additional candidate set for the to-be added facility, making their solutions inapplicable to our problem.

Unlike the above problems, which define the influence values based on the cardinalities of RNN sets, Gao et al. [12] propose to find the optimal location f outside a given region Q based on the number of customers in Q whose distances to f is within a given threshold d. We consider neither the specific region Q nor the given threshold d in our study, which makes their solution inapplicable to our problem.

Overall, Table 3.1 summarizes above studies based on their inputs, outputs, assumptions on formulation the problem and their proposed solutions.

Chapter 4

Solutions

In this chapter, we will comprehensively study four solutions for top-k most influential location selection query. To begin with, we follow the definition of the problem and present a rather naive solution, which is based on performing a sequential scan (SS) on all data sets. Although this solution returns correct answer, its efficiency is unsatisfactory since it accesses the data sets intensively and performs repeated computations. To improve the efficiency, we propose two R-Tree based branch-and-bound solutions. The R-Tree is a widely used index structure designed specifically for spatial data[14]. Each spatial object is associated with a Minimum Bounding Rectangle (MBR). Multiple MBRs are then grouped as nodes in upper levels of the tree, which are again associated with bigger MBRs bounding all MBRs in the corresponding group. To perform queries on an R-Tree, typically we traverse the tree using the MBR corresponding to a node as the indicator to decide whether its children of that node should be accessed. Both proposed methods index all three data sets with R-Trees or its variant, and rely on estimating the influence bounds for candidate locations to tighten the search space. One of them, named Estimation Expanding Pruning (EEP), uses distance metrics between MBRs to gradually refine the estimation during traversing all three trees. The other algorithm, named Bounding Influence Pruning (BIP), maintains a heap for nodes in the tree corresponding to candidate locations and takes advantages of *Voronoi-cell* styled geometric properties to reduce distance computations. Both EEP and BIP are of complexity $O(n \log n)$ in the best case, which is better than the $O(n^2)$ complexity of SS. Yet, the worst case complexities of EEP and BIP are $O(n^4)$ and $O(n^3)$, respectively, which are far from competitive. To overcome this, we further study another R-Tree based solution. This solution indexes the *nearest facility circle* (NFC) instead of the location for each customer and transforms the most influential location selection query into multiple points enclosure queries on that R-Tree. Additionally, we may construct a R-tree for the candidate set and employ join algorithm on R-tree to share the cost of point queries which result to NFC join (NFCJ) algorithm. Both NFC and NFCJ achieve $O(n \log n)$ in the best case and $O(n^2)$ in the worst case, while NFCJ outperforms NFC in terms of I/O operations. Hence, NFCJ is the best solution for the most influential location selection query.

4.1 Sequential Scan

As defined in Definition 4, the problem can be solved in a straightforward manner following the idea in the definition. In order to select the k most influential locations in the candidate set M, we first compute the exact influence for each candidate location $c \in C$ then simply return the k largest. The most naive implementation of this idea is to use sequential scan to obtain the candidate influence. For each candidate c, we obtain its influence value by adding it to the existing facility set F and scanning the customer set M to find reverse nearest neighbors, namely ms which have c as their nearest facility. This requires $|C||F||M|$ number of scans.

Algorithm 1: Sequential Scan (SS)

```
Input: $k$, customer set $M$, existing facility set $F$, candidate location set $C$
Output: TopInf(k, M, F, C)
1  foreach $m \in M$ do
2      foreach $f \in F$ do
3          if $m.nfd > d(f,m)$ then
4              $m.nfd \leftarrow d(f,m)$
5  foreach $c \in C$ do
6      foreach $m \in M$ do
7          if $m.nfd > d(c,m)$ then
8              $I_c$++
9  Sort $C$ by $I_c$
10 TopInf(k, M, F, C) $\leftarrow$ First $k$ locations in $C$
```

Notice that the set M is repeatedly scanned for existing facilities $f \in F$ when computing influence for each c. These repeated scans can be avoided by first scanning the customer set M and the existing facility F, then storing the distance between each c and its nearest facility for further use. Let this distance be called $n$earest $f$acility $d$istance (nfd). When computing the influence for candidate c, we can scan M again to find which $m \in M$ perceives c as the nearest facility by simply checking whether the distance between c and m is within the nfd corresponding to m. Due to the fact this method only needs sequential scans on data sets, we name it the *Sequential Scan* algorithm.

The algorithm can be summarized as Algorithm 1, where $m.nfd$ is the nearest facility distance of m and $d(a,b)$ is the distance between location a and location b. As shown in the pseudo-code, it is easy to find that the sequential scan algorithm requires $|F||M| + |C||M|$ distance computations, leading to a time complexity of $O(n^2)$.

The sequential scan algorithm is far from efficient of solving top-k most influential location problems due to the fact it relies on intensive scans on all of three data sets and computes unnecessary influence for weakly influential candidates, which are of less interest in the problem. Also, repeated scans on set M is undesirable since in reality, this set tends to be the largest among all sets. In following sections, we come up with two algorithms relying on heuristics to prune search space and improve efficiency in terms of execution time.

4.2 Estimation Expanding Pruning

To prepare for the pruning of the search space, we index sets C, F with R-Trees and M with an aggregate R-Tree for more efficient access and heuristic operations. Let t_C, t_F, t_M denote these R-Trees. Recalling that our problem asks for top-k most influential locations, it is desirable to estimate the influence for candidates before computing their exact influence and to use this information to eliminate unpromising candidates at an early stage. This way, we can avoid exhaustive computations by taking advantage of influence value distribution among candidates.

In this section, we propose a distance based technique to help us estimate the influence of candidate locations. The distances between customers and their nearest existing facilities as well as the possible number of customers that could be influenced by the candidate facility are estimated. With these estimations, the solution traverses t_C in a best first order determined by *importance* of that node, which depends on both estimated influences and the number of candidates the node's Minimum Bounding Rectangle (MBR) encloses, to quickly find the top-k candidates.

When an internal node of t_C is accessed, each of its child nodes is evaluated for its importance. Since this operation naturally expands the search space, we call it the expanding operation on an index tree. t_F and t_M are also expanded so that the estimations on distances and influences can be gradually refined, which brings more effective pruning in return. As estimation and expanding operations play the major role in the solution, it is named the *Estimation Expanding Pruning* (EEP) algorithm.

Trees t_C, t_F, t_M are traversed in a best first order by maintaining priority lists L_C, L_F, and L_M, which contain entries corresponding to the to-be accessed nodes. These entries also record the importance values for the nodes and some *influence relation information* presented by related entries in other lists. Initially, only the roots of trees are stored in the corresponding lists. As the traversal proceeds, lists are accessed in L_M, L_F, L_C order, most important entries in lists are expanded and entries corresponding to their child nodes are re-inserted into the lists. The repeated accesses terminate when the top-k candidates have been found. Specifically, we maintain a sorted list for all computed influence values in descending order. Once all upper bound I_C^u of influences for nodes in the tree are smaller then the k^{th} influence value in that list, no more candidate location remaining in the tree could serve as the query answer since none of them could have an influenced value greater than the k^{th} computed one. Hence, the algorithm could terminate at an early stage. Algorithm 2 shows the high level algorithm of EEP.

Algorithm 2: Estimation Expanding Pruning (EEP)

```
Input: Root nodes $root_M$, $root_F$ and $root_C$ of the three trees
Output: TopInf(k, M, F, C)
1 Insert $root_M$, $root_F$ and $root_C$ into $L_M$, $L_F$ and $L_C$ respectively
2 Initialize the influence relation information for the nodes in the three lists
3 while $\forall n_C \in L_C$, the $k^{th}$ largest computed $I_c < I_C^u$ do
4     ProcessEntry( $L_M$ )
5     ProcessEntry( $L_F$ )
6     ProcessEntry( $L_C$ )
7 return TopInf(k, M, F, C)
```

On Lines 4-6 in Algorithm 2, the entries with greatest importance value are selected, the corresponding nodes are expanded, and the importance values and influence relation information of their child nodes are computed so that promising nodes are re-inserted into the lists as entries while unpromising nodes are discarded directly. Details for this procedure is given in Section 4.2.3. Before taking a more detailed look, we will first give the definitions on hints used in the procedure, which are stored in the entry with corresponding node. Specifically, influence relation information is introduced in Section 4.2.1 and importance values of nodes are defined in Section 4.2.2. The run-time complexity of EEP is given in Section 4.2.4.

4.2.1 Influence relation information of nodes

The influence relation information of a node $n \in t$ contains sets of nodes from trees other than t which either potentially influence n if n represents customers or potentially influenced by n if n represents facilities. These sets are named *influence relation sets*. In addition, some distances between node n and nodes in influence relation sets are also stored for efficient access and computation. This information together helps in efficiently determining the importance of a node, as described in Section 4.2.2, as well as pruning the search space when processing entries, as elaborated in Section 4.2.3.

For brevity, in the remainder of the report, we denote n_C, n_F, and n_M as nodes in t_C, t_F, and t_M, respectively; r_C, r_F, r_M as the corresponding MBRs of node n_C, n_F, n_M, respectively.

Specifically, for n_C, in order to estimate the influence for candidates enclosed in its r_C, a set named $n_C.S_M$ is maintained as the set of nodes n_M which might be influenced by any c enclosed by r_C. For n_F, similar to that for n_C, a set $n_F.S_M$ is maintained for estimating the influence of facility loca-

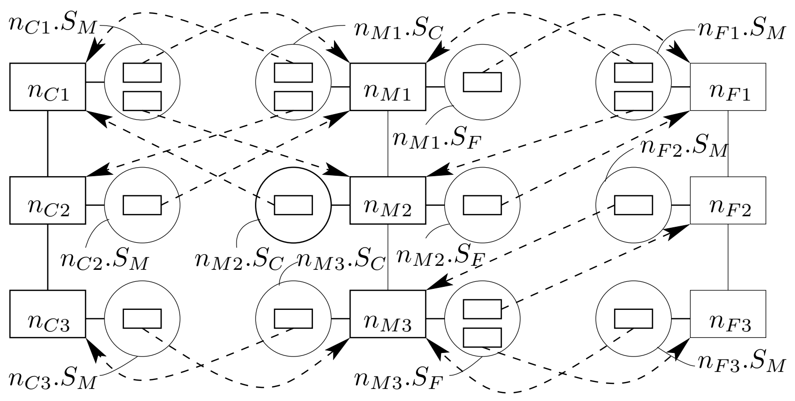


Figure 4.1: An example for influence relation sets on list L_C, L_F, L_M

tion represented by n_F. For n_M, two sets $n_M.S_F$ and $n_M.S_C$ are maintained. Set $n_M.S_F$ contains n_F such that facility f enclosed by r_F may influence any customer m enclosed by r_M. Similarly, set $n_M.S_C$ contains nodes n_C such that candidate facility c enclosed by r_C may influence any customer m enclosed by r_M. The method to determine these sets are elaborated in the later part of this section. Figure 4.1 shows an example, where $L_C = \{n_{C1}, n_{C2}, n_{C3}\}$, $L_F = \{n_{F1}, n_{F2}, n_{F3}\}$, $L_M = \{n_{M1}, n_{M2}, n_{M3}\}$. As demonstrated in the figure, $n_{C1}.S_M = \{n_{M1}, n_{M2}\}$, $n_{C2}.S_M = \{n_{M1}\}$, $n_{C3}.S_M = \{n_{M3}\}$; $n_{F1}.S_M = \{n_{M1}, n_{M2}\}$, $n_{F2}.S_M = \{n_{M3}\}$, $n_{F3}.S_M = \{n_{M3}\}$; $n_{M1}.S_C = \{n_{C1}, n_{C2}\}$, $n_{M1}.S_F = \{n_{F1}\}$, $n_{M2}.S_C = \{n_{C1}\}$, $n_{M2}.S_F = \{n_{F1}\}$, $n_{M3}.S_C = \{n_{C3}\}$, $n_{M3}.S_F = \{n_{F2}, n_{F3}\}$.

In order to compute the influence relation sets for a given node efficiently, we introduce three distance metrics between nodes. Given two nodes, which are represented by their MBRs r_1 and r_2, the **MinDist**(r_1, r_2), and the **MaxDist**(r_1, r_2) are respectively the minimum distance and the maximum distance between any pair of points, one enclosed by r_1 and the other in r_2. The distance metric **MinExistDist**$_{r_2}(r_1)$, which is introduced in [35], is defined as the minimum upper bound of the distance for a point in r_1 to its nearest point in r_2. In other words, it is possible to find the nearest point in r_2 for any point in r_1 within distance $MinExistDist_{r_2}(r_1)$. The influence relation sets $n_C.S_M$, $n_F.S_M$, $n_M.S_C$, and $n_M.S_F$ can be determined with the following two theorems,

Theorem 1. *Given $n_C \in L_C$, if $\exists n_F \in \{n_F | n_M \in n_C.S_M, n_F \in n_M.S_F\}$,*

$$MinDist(r_M, r_C) \geq MinExistDist_{r_F}(r_M),$$

then for m enclosed by r_M, m is not influenced by any c enclosed by r_C.

Proof. We prove by contradiction. Suppose there is an m enclosed by r_M who is influenced by a specific c enclosed by r_C, then according to Definition 2, $d(c, m) < MinExistDist_{r_F}(r_M)$; but since $d(c, m) \geq MinDist(r_M, r_C)$ by the definition of $MinDist$, this contracts with $MinDist(r_M, r_C) \geq MinExistDist_{r_F}(r_M)$. □

Theorem 2. *Given $n_C \in L_C$, if $\forall n_F \in \{n_F | n_M \in n_C.S_M, n_F \in n_M.S_F\}$,*

$$MaxDist(r_M, r_C) < MinDist(r_M, r_F),$$

then for m enclosed by r_M, for c enclosed by r_C, m is influenced by c.

Proof. Since $MaxDist(r_M, r_C) < MinDist(r_M, r_F)$, for any m, c, f enclosed by n_M, n_C, and n_F, respectively, $d(m, c) < d(m, f)$. According to Definition 2, f enclosed by n_F cannot influence m enclosed by n_M due to the existence of n_C. Also, since $n_M.S_F$ contains all n_Fs those could influence n_M with the absence of n_C, m enclosed by n_M can only be influenced by some c enclosed by n_C when c is added. □

Theorem 1 states that if node n_C is so far away from node n_M that there are other n_Fs much nearer to n_M, then customer ms represented by n_M are not influenced by any candidate location c represented by n_C due to the existence of n_F. To complete this intuition, Theorem 2 states that if node n_C is so close to node n_M that no other n_Fs can be closer, then customers represented by n_M shall be influenced by some candidate location c in n_C.

While Theorems 1 and 2 help determine the influence relations between node pairs, we need to compute and store further distance thresholds for node $n_M \in L_M$ to improve efficiency in determining influence relation sets for nodes in lists. Specifically, one of these needed thresholds, denoted as $n_M.d_{low}$, stores the lower bound for the distance between r_M and its nearest r_F, while the other, denoted as $n_M.d_{upp}$, stores the upper bound for the distance between r_M and its nearest r_F. Formally, we have

$$n_M.d_{low} = min(\{MinDist(r_M, r_F) | \forall n_F \in n_M.S_F\})$$

and

$$n_M.d_{upp} = min(\{MinExistDist_{r_F}(r_M) | \forall n_F \in n_M.S_F\}).$$

With the theorems and stored distances introduced above, three rules are available for pruning and determining the influence relation set for child nodes $n_{M'}$ of a given node n_M. The rules are as follows:

1. Given n_C, if $\exists n_M \in n_C.S_M, n_M.d_{low} > MaxDist(r_C, r_M)$, then according to Theorem 2, $\forall m$ enclosed by r_M and $\forall c$ enclosed by r_C, m is influenced by c. Since we can ensure this customer in r_M will be influenced, node n_M should be removed from $n_C.S_M$ and n_C should be removed from $n_M.S_C$ as well. In the meantime, for c enclosed by r_C, I^l_C should be increased by $|O_M|$.

2. Given n_C, if $\exists n_M \in n_C.S_M, MinDist(r_M, r_C) \geq n_M.d_{upp}$, according to Theorem 1, $\forall m$ enclosed by r_M and $\forall c$ enclosed by r_C, m is not influenced by r_C. Hence, n_M and n_C should be removed from $n_C.S_M$ and $n_M.S_C$, respectively.

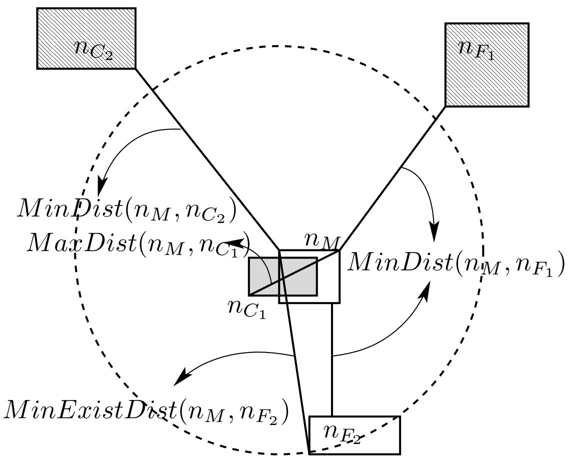


Figure 4.2: n_{F_1} can be pruned from $n_M.S_F$, n_{C_2} can be pruned from $n_M.S_C$

3. Similar to rule 2, given n_F, if $\exists n_M \in n_F.S_M$, $MinDist(r_M, r_F) \geq n_M.d_{upp}$, then $\forall m$ enclosed by r_M and $\forall f$ enclosed by r_F, m is not influenced by f. Thus, n_M and n_F should be removed from $n_F.S_M$ and $n_M.S_F$, respectively.

Figure 4.2 shows an example of these pruning rules on influence relation sets. In this example, $n_M.d_{upp} = MinExistDist(r_M, r_{F_2})$, $n_M.d_{low} = MinDist(r_M, r_{F_1})$. According to rule 2, n_{C_2} is not in $n_M.S_C$ since $MinDist(r_{C_2}, r_M) \geq n_M.d_{upp}$; according to rule 3, n_{F_1} is not in $n_M.S_F$ since $MinDist(r_{F_1}, r_M) \geq n_M.d_{upp}$; according to rule 1, any c enclosed by r_{C_1} should influence all m enclosed by r_M, since $MaxDist(r_M, r_{C_1}) < n_M.d_{low}$.

Intuitively, the influence relation sets are initialized as the roots of corresponding trees, namely, $n_{root_C}.S_M = \{root_M\}$, $n_{root_F}.S_M = \{root_M\}$, $n_{root_M}.S_C = \{root_F\}$, and $n_{root_M}.S_F = \{root_F\}$. As mentioned in algorithm 2, roots are stored in the lists with the influence relation information as entries. Each time an entry is processed, its child nodes are re-inserted into the corresponding lists with their influence relation information computed. Before giving the details on how the entries are processed in Section 4.2.3, we will first introduce the criteria for determining the order of processing the entries in the next section.

4.2.2 Importance of nodes

We use the importance values of nodes to determine the access order for their corresponding entries in the lists. In EEP, different trees have different uses in pruning the search space: the candidate R-tree is traversed for the goal of computing influence value for candidates while the existing facility R-tree and the customer R-tree are traversed for the goal to prune unnecessary computations during computing the influence. Therefore, we have defined “importance” differently for different trees to guide the tree traversals towards different goals.

To simplify the narration, we denote $area(n)$ and $|n|$ as the area of MBR and the number of locations enclosed in the MBR corresponding to node n; $|n.S|$ as the number of nodes in n's influence relation set; $n.imp$ is the importance value of node n.

- $n_C.imp$. Since only the most influential candidates are desired in our problem, we want to always access the most promising candidates first. Recalling from Section 4.2.1, each entry in L_C holds I_C^l for node n_C in it, we define the importance of a candidate node as the maximum number of customers it can influence estimated by influence relation information in that entry, i.e. $n_C.imp = I_C^l + \sum_{n_M \in n_C.S_M} |n_M|$.
- $n_F.imp$. If the corresponding MBR of a facility node has a larger area, it may affect the estimation of the influence of more candidates. Also, a larger influence relation set on C suggests a facility node being more relevant for candidates. Hence, $n_F.imp$ is defined as $area(n_F) \cdot |n_F.S_C|$.
- $n_M.imp$. Larger $area(o_M)$ and larger $|o_M|$ suggest a greater possibility for a customer node affecting the influence relation information of nodes in other lists, therefore the customer node is more useful for pruning the search space. With this rationale, $n_M.imp$ is defined as $area(n_M) \cdot |n_M| \cdot |n_M.S_F| \cdot |n_M.S_C|$.

4.2.3 Processing entries

As described in Lines 4-6 in Algorithm 2 and the above sections, each time an entry with the greatest importance value on a list is picked, its child nodes are evaluated. If the node in the picked entry is a leaf node in t_C, denoted as n_C, then for c in n_C we compute their exact influence values by checking I_C^l, $n_C.S_M$ and $n_M.S_F$ for all $n_M \in n_C.S_M$. The computation is similar to the solution given in Section 4.1, however, here sets M and F are dramatically smaller thanks to stored influence relation information in the entry. If the node in the picked entry is an internal node, then we compute the influence relation information by inheriting it from the parent node and prune it using rules described in Section 4.2.1. Please note we can always inherit the information instead of computing it from scratch because the following property holds.

Property 1. For any child node $n_{M'}$ of n_M, $n_{M'}.S_F \subset n_M.S_F$, and $n_{M'}.S_C \subset n_M.S_F$; for any child node $n_{C'}$ of n_C, $n_{C'}.S_M \subset n_C.S_M$; for any child node $n_{F'}$ of n_F, $n_{F'}.S_M \subset n_F.S_M$.

After obtaining the influence relation information for child node n', we should update the stored information in other lists related to the parent n. Again, we use the three rules based on distances between MBRs to prune unpromising nodes in the influence relation sets. If child node n' is still related to other nodes in the lists, then its importance value is computed and the node shall be re-inserted to the corresponding list. If the node picked is a leaf node in t_F or t_M, we update its influence relation set, assign an importance value of -1 to it and re-insert it into list L_F or L_M, respectively.

The algorithm terminates when there are at least k computed candidate location cs' influence values and all nodes remaining in L_C have maximum influence values smaller than the k^{th} largest influence values of candidates computed. The maximum influence values of n_C in L_C could be computed as $I^u_C = I^l_C + |n_C.S_M|$. Also, when all entries on L_M or L_F have importance values smaller than 0, Line 4 or Line 5 is omitted in Algorithm 2. Algorithm 3 gives the pseudo-code for the procedure in Lines 4-6 of Algorithm 2. Note that in line 8 whether n' is related can be determined by checking the three rules introduced in Section 4.2.1.

Algorithm 3: ProcessEntry(L)

```
   Input: List $L$
1  Pick $n$ with the largest $n.imp$ from $L$
2  if $n$ is an internal node then
3      Expand $n$
4      foreach child object $n' \in n$ do
5          Compute the influence relation information of $n'$
6          Update the influence relation information of nodes related to $n'$
7          Prune unpromising objects with updated influence relation
           information
8          if $n'$ is related then
9              Insert $n'$ into $L$
10 if $n$ is a leaf node then
11     if $n \in L_C$ then
12         $I^u = I^l + |n.S_M|$
13         if $I^u > k^{th}$ largest $I_c$ computed so far then
14             foreach $c \in n$ do
15                 Compute $I_c$
```

4.2.4 Complexity of EEP

The construction of the R-Tree indexes incurs $O(n \log n)$ cost. With the constructed R-Tree, the major cost of EEP lies in the processing entry procedure, which computes the importance and influence relation information for children nodes of a node in the entry. Let r denote the average size of the influence relation sets. As described in Algorithm 3, the cost of processing an entry is contributed by computing the influence relation information and updating influence relation information for other related nodes. Though in practical settings, M tends to be much larger than both F and C, we first denote their sizes as $O(n)$ for simplicity. For each entry, computing influence relation information costs $O(r)$ since a traversal on its parent's influence relation set is adequate. Updating influence relation information for related nodes is more complicated. For each entry on L_M, we need to check $n_C \in n_M.S_C$ and $n_F \in n_M.S_F$ to see whether they can be pruned, for n_C updated, we also check $n_M \in n_C.S_M$ for further updates, resulting in $O(r^2)$ cost. For each entry on L_C, we need to check $n_M \in n_C.S_M$ to see whether n_M can be pruned, also leading to an $O(r)$ cost. For each entry on L_F, we need to first check $n_M \in n_F.S_M$, for n_M not pruned in this procedure, $n_M.d_{low}$ and $n_M.d_{upp}$ are updated. Additionally, we must evaluate $n_C \in n_M.S_C$ and $n_F \in n_M.S_F$ to see whether they can be pruned with the new distances. For n_C updated, we check $n_M \in n_C.S_M$ to further compute the influence upper bound for it. This procedure leads to a cost of $O(r^3)$.

In the best case, only $O(k)$ candidates with greatest influence values are accessed, therefore $O(k \cdot \log n)$ node n_Cs are accessed. The while loop of Line 3 in Algorithm 2 only executes $O(\log n)$ times. Since influence relation sets are pruned smoothly, the relevant set size $O(r) = O(1)$. The overall cost of EEP in the best case is $O(n \log n) + O(\log n) \cdot (O(1) + O(1^2) + O(1) + O(1^3)) = O(n \log n)$.

In the worst case, all candidates are accessed so the t_M is actually also fully traversed. The while loop therefore executes $O(n)$ times. Here, the techniques eliminate few nodes from influence relation sets, thus $O(r) = O(n)$. To summarize, the overall cost of EEP in the worst case is $O(n \log n) + O(n) \cdot (O(n) + O(n^2) + O(n) + O(n^3)) = O(n^4)$.

Clearly, the major factor determining the efficiency of EEP is the average size r of the influence relation sets. In Chapter 5 we will see that in both synthetic and real-world data sets, EEP outperforms SS in terms of executing time, indicating r is generally much less then $O(n)$. As previously mentioned, $|M|$ can be larger than $|F|$ and $|C|$ in reality. Thus if $|M|$ dominate the scale of the problem, the complexity of EEP can be represented by $O(|M| \log |M|)$ in the best case and $O(|M|^3)$ in the worst case. Yet as illustrated in Section 5.4 the growth of EEP when $|M|$ increases is similar to that of SS, suggesting in most cases the complexity tends to be $O(|M| \log |M|)$ thanks to effective pruning techniques.

4.3 Bounding Influence Pruning

EEP needs to maintain additional lists storing information of relation sets for not only nodes in the candidate tree t_C, but also for the customer tree t_M and the existing facility tree t_F. This can consume a large amount of memory space when the data sets are huge. Also, as a result of maintaining influence relation lists for all data sets, its complexity in the worst case is at an undesired $O(n^4)$. Since we ultimately care only about most influential candidates in answering queries of the form defined in Definition 4, it is desired to focus on the candidate tree t_C rather than extensively studying all three trees.

In this section, we introduce another strategy to estimate and refine influence values for candidate locations without storing redundant information for customer locations and existing facility locations. Similar to EEP, we again index C and F with R-Trees and M with an aggregate R-Tree.

Specifically, the method traverses t_C in a best first order, using a max-heap to order currently available nodes by their influence upper bounds computed before their insertions. For each node on the heap, we store an influence region

$n_C.R$, locating where all customers can only be influenced by the candidates in that node, a set $n_C.S_F$ of relevant F node and a set $n_C.S_M$ of relevant M node. A node n_F is called relevant to n_C if its existence potentially helps refining the influence regions for child node $n_{C'}$ of n_C. A node n_M is called relevant to n_C if its MBR r_M intersects the influence region $n_C.R$, namely, m in r_M might be influenced by c in r_C.

Each time, the top node on the max-heap is picked. When the picked node is an internal one, the algorithm relies on geometric properties together with stored information in the picked node to refine its children nodes' influence regions and relevant sets. With this computed information, the influence upper bounds of child nodes can be retrieved by performing point enclosure queries on t_M. Next, each child node is evaluated with a global influence threshold, which is maintained globally as the k^{th} greatest influence value of candidates computed so far. If it cannot be pruned, it is inserted into the max-heap for further consideration. When the picked node is a leaf node, the method uses further techniques to tighten the search space and performs exact influence computation for each location enclosed by the leaf node. Because the method focuses on bounding candidates and uses influence upper bounds to prune unpromising candidates, it's named the *Bounding Influence Pruning* (BIP) algorithm. Algorithm 4 lists the pseudo-code for BIP, where I_δ maintains the k^{th} maximum influence values seen so far.

Algorithm 4: Bounding Influence Pruning (BIP) Algorithm

```
Input: Root nodes $root_M$, $root_F$ and $root_C$ of the three trees
Output: TopInf(k,M,F,C)
1  $root_C.S_F \leftarrow \{root_F\}$, $root_C.S_M \leftarrow \{root_M\}$, $H_C.insert(root_C)$
2  while $H_C \neq \varnothing$ do
3      $n_C = H_C.pop()$
4      if $n_C$ is leaf then
5          foreach $c$ enclosed by $n_C$ do
6              Compute exact influence value $I_c$
7              if $I_c > I_\delta$ then
8                  Update $TopInf(k, M, F, C)$ and $I_\delta$
9      else
10         foreach child node $n_{C'}$ of $n_C$ do
11             Construct $n_{C'}.R$, $n_{C'}.S_F$, $n_{C'}.S_M$;
12             Compute the influence value upper bound $I^u_{c'}$
13             if $I^u_{c'} > I_\delta$ then
14                 $H_C.insert(n_{C'})$
15 return TopInf(k, M, F, C)
```

We will first introduce the methods for computing influence region and relevant sets (Lines 11-12) in Section 4.3.1, and then elaborate on the techniques involved in computing exact influence values for candidates (Line 6) in Section

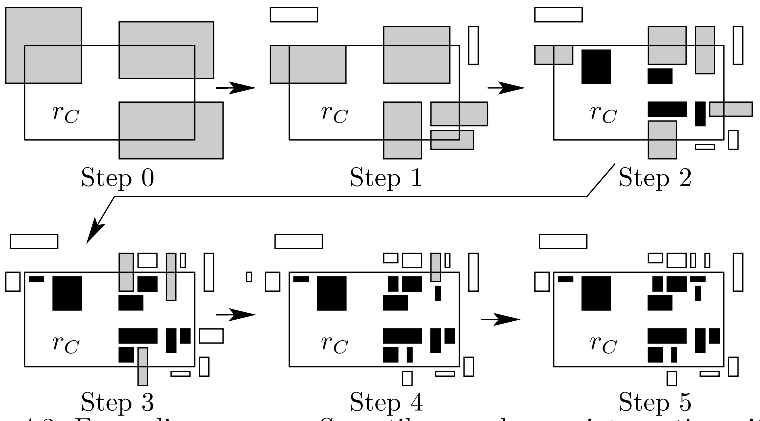


Figure 4.3: Expanding $n_F \in n_C.S_F$ until no n_F has r_F intersecting with r_C

4.3.2. The section ends with an analysis of the complexity of BIP in Section 4.3.3.

4.3.1 Constructing influence region and relevant sets

In this section, we describe the details for computing the influence region and relevant sets for a given internal node n_C. As described in Algorithm 4, the root nodes of t_M and t_F are inserted into the relevant sets $root_C.S_M$ and $root_C.S_F$, respectively. Then, $root_C$ is inserted into the max-heap H_C.

Each time we pick the top node n_C from H_C, which has the maximum I_C^u among the nodes on the heap. To compute the influence region for a given node n_C, we first expand every $n_F \in n_C.S_F$ that has r_F intersecting with r_C. Obviously, after a series of expanding operations, some r_F will lie outside r_C while other r_Fs will lie inside r_C. We call those r_Fs lying inside r_C as inner relevant F nodes, denoted as $r_C.S_F^-$. Since we only need to compute influence upper bound for n_C, we will only use $n_C.S_F \setminus r_C.S_F^-$ to compute the influence region. For those customers locations laying inside r_C, we assume they are influenced by candidates in n_C as we will ignore $n_C.S_F^-$ when computing the influence region. Figure 4.3 shows an example of expanding n_F. In the figure, gray rectangles represent n_F with r_F intersecting with r_C, and black rectangles represent $n_F \in n_C.S_F^-$.

We use the following idea to compute the influence region for a given node n_C. On plane $\mathbb{P}$, given two points, their *perpendicular bisector* divides $\mathbb{P}$ into two regions, each of which contains one point. If another point q is influenced by point p_1 rather than p_2, it must lie inside the half plane containing p_1 rather than that containing p_2. [6] generalizes this idea from points to rectangles. Given two rectangles r_1 and r_2, the generalized theorem uses multiple *normalized perpendicular bisectors* to divide the plane $\mathbb{P}$ into two regions, one of which contains all the points that might be influenced by points in r_1.

Formally, we introduce this idea using concepts *antipodal corners*, *normalized perpendicular bisectors* and Theorem 3.

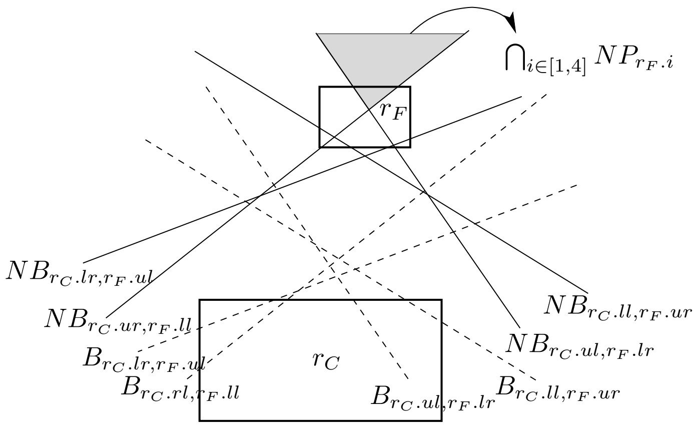


Figure 4.4: Normalized half planes are divided by $NB_{r_C.1,r_F.3}, NB_{r_C.2,r_F.4}, NB_{r_C.3,r_F.1}, NB_{r_C.4,r_F.2}$; the gray region $\bigcap_{i\in[1,4]} NP_{r_F.i}$ contains customers who won't be influenced by candidates enclosed by r_C according to Theorem 3

Definition 5 (Antipodal Corners). *Let a rectangle r's lower left, lower right, upper left, and upper right corners be $r.ll, r.lr, r.ul, r.ur$, respectively. Given two rectangle r_1 and r_2, the antipodal corners of r_1 and r_2 are four pairs of corner points $(r_1.ll, r_2.ur)$, $(r_1.lr, r_2.ul)$, $(r_1.ul, r_2.lr)$, and $(r_1.ur, r_2.ll)$.*

Definition 6 (Normalized Perpendicular Bisectors). *Given two rectangles r_1 and r_2 and a pair of antipodal corners of r_1 and r_2, (ac_1, ac_2), the normalized perpendicular bisector of ac_1, ac_2, denoted as NB_{ac_1,ac_2}, is obtained through moving their perpendicular bisector, denoted as B_{ac_1,ac_2}, to intersect a point p_{ac_1,ac_2}, where*

$$p_{ac_1,ac_2}.x = \begin{cases} \frac{r_1.ur.x + r_2.ur.x}{2}, & \text{if } ac_1.x < ac_2.x, \\ \frac{r_1.ul.x + r_2.ul.x}{2}, & \text{if } ac_1.x \geq ac_2.x. \end{cases}$$

$$p_{ac_1,ac_2}.y = \begin{cases} \frac{r_1.ul.y + r_2.ul.y}{2}, & \text{if } ac_1.y < ac_2.y \\ \frac{r_1.ll.y + r_2.ll.y}{2}, & \text{if } ac_1.y \geq ac_2.y. \end{cases}$$

Figure 4.4 shows an example of the normalized half planes described above.

The concepts defined above help us to find influence regions for a given node n_C. Specifically, perpendicular bisectors ac_1 and ac_2 divide the plane $\mathbb{P}$ into two half planes, denoted as P_{ac_1} and P_{ac_2}, respectively. The normalized perpendicular bisectors of ac_1 and ac_2 also divide the plane into two planes. Let NP_{ac_1} be the normalized perpendicular bisector corresponding to the half plane P_{ac_1} and NP_{ac_2} be the one corresponding to the half plane P_{ac_2}. Remember that two rectangles have four pairs of antipodal corners, each pair of nodes contribute to a pair of normalized half planes. For brevity, let $r_i.lr = r_i.1$, $r_i.ur = r_i.2$,

$r_i.ul = r_i.3$, $r_i.ll = r_i.4$, where $i = \{1,2\}$. Hence, the four pairs of half planes can be denoted as $(NP_{r_1.1}, NP_{r_2.3})$, $(NP_{r_1.2}, NP_{r_2.4})$, $(NP_{r_1.3}, NP_{r_2.1})$, and $(NP_{r_1.4}, NP_{r_2.2})$. Given a node n_C and a node n_F, and the theorem below proved by [6], any customer m lying in $\bigcap_{i \in [1,4]} NP_{r_2.i}$ is not influenced by any candidate enclosed by r_C.

Theorem 3. *Given two rectangles r_1 and r_2, let p be a point in $\bigcap_{i \in [1,4]} NP_{r_2.i}$, where $NP_{r_2.i}$ denotes a normalized half plane corresponding to r_2. Then the minimum distance between p and any point in r_1 must be larger than the maximum distance between p and any point in r_2.*

The gray region in Figure 4.4 is the described region of Theorem 3 of the given example. Since this region contains customers, it cannot be influenced by n_C, so we call it a pruning region on n_C defined by n_F, denoted as $n_C.PR_{n_F}$. The union of pruning regions defined by each n_F in $F \setminus n_C.S_F^-$ is a region containing customers not influenced. In other words, $\mathbb{P} \setminus \bigcap_{n_F \in F \setminus n_C.S_F^-} (n_C.PR_{n_F})$ is a region containing customers potentially influenced by candidates enclosed by r_C for a given n_C. We define this region as the *influence region* for n_C, i.e., $n_C.R$. However, according to its definition, computing this region would require extensive accesses on $F \setminus n_C.S_F^-$. To avoid this tremendous cost, we select the nearest n_Fs in 8 directions, denoted as *Western(W)*, *Southwestern(SW)*, *Southern(S)*, *Southeastern(SE)*, *Eastern(E)*, *Northeastern(NE)*, *Northern(N)*, *Northwestern(NW)*, of node n_C. Again, the distance indicator used is the maximum distance between two rectangles r_C and r_F. Let $n_C.NN_{n_F}$ denote the set for these eight nearest n_Fs, $n_C.R_{app}$ denote $\mathbb{P} \setminus \bigcap_{n_C.NN(n_F)} n_C.PR_{n_F}$, then since $n_C.NN_{n_F} \subset F \setminus n_C.S_F^-$, $n_C.R \subset n_C.R_{app}$. Region $n_C.R_{app}$ is only an *approximate influence region* of n_C, but as it contains the desired $n_C.R$, it can be used as a substitute for computing the influence upper bound and relevant sets. In in the remainder of this report, we use $n_C.R$ to denote the approximate influence region if there is no ambiguity.

Figure 4.5 demonstrates an example of computing $n_C.R$ for a given n_C, where its nearest 8 n_Fs are in different directions. In the figure, the nearest n_Fs are tagged by their corresponding direction to n_C. The inner polygon is $n_C.R$ is computed from r_C and r_Fs using Theorem 3.

Due to the fact customers are indexed by an aggregate R-Tree $t_M H$, we further bound the polygon $n_C.R$ with several rectangles to enable efficient point enclosure queries on t_M. The gray region in Figure 4.5 corresponds to these regions. In the implementation of the algorithm, it is these bounding rectangles rather than the exact $n_C.R$ that is used to compute I_C^u. Also, according to the definition of relevant M set, only n_M with r_m lying inside $n_C.R$ can have customers being influenced by n_C. For those intersecting r_M, we expand the corresponding n_M just like expanding n_F in the aforementioned description, until there is no intersection. Thus, $n_C.S_M = \{n_M \subset n_C.R\}$.

Besides obtaining the relevant M set, we also need to compute the relevant HF set with help from $n_C.R$. Recall that the operations in Theorem 3 divide the plane into 2 regions, one of which contains points not influenced by any

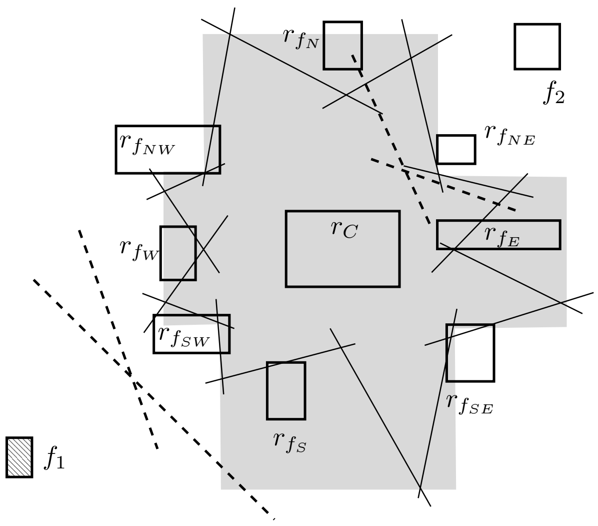


Figure 4.5: Example of Influence Region

point in r_1. We would like to use this to evaluate whether a node n_F $F \setminus n_C.S_F^-$ is relevant to n_C. Specifically, we apply Theorem 3 treating n_F as the rectangle r_1, leading to a region $n_F.PR_{n_C} = \bigcap_{i\in[1,4]} NP_{r_C.i}$ This $n_F.PR_{n_C}$ contains customers that cannot be influenced by any existing facility locations f enclosed by r_F. Obviously, if $n_C.R \subset n_F.PR_{n_C}$, then for any customer m potentially influenced by c in n_C, any f enclosed by r_F won't influence it. Namely, if a given node n_F meets $n_C.R \subset n_F.PR_{n_C}$, it can be eliminated from $n_C.S_F$ since it is not relevant to the influence value of n_C. Of all nodes n_F those are not eliminated by this evaluation form the outer relevant F set of n_C, denoted as $n_F.S_F^+$. Then, we have $n_C.S_F = n_C.S_F^+ + n_C.S_F^-$. In Figure 4.5, f_1 meets $n_{f_1}.PR_{n_C} \subset n_C.R$ thus it will be eliminated from $n_C.S_F$, while f_2 has $n_{f_2}.PR_{n_C} \not\subset n_C.R$, thus $f_2 \in n_C.S_F$.

As described above, in order to compute $n_C.S_F^+$, we need to evaluate all n_F $F \setminus n_C.S_F^-$. This can be rather time-consuming when F is massive. To overcome this, the following property is introduced.

Property 2. For child node $n_{C'}$ of node n_C,

$$n_{C'}.R \subset n_C.R, n_{C'}.S_F \subset n_C.S_F, n_{C'}.S_M \subset n_C.S_M.$$

Property 2 enables us to tighten the search space for evaluating $n_C.S_F$. Let $n_{C'}$ be a child node of n_C, when computing $n_{C'}.S_F^+$, instead of checking every $n_F \in F \setminus n_{C'}.S_F^-$, it is adequate to check only $n_F \in n_C.S_F \setminus n_{C'}.S_F^-$. This trick reduces unnecessary computation dramatically.

4.3.2 Computing exact influence for candidates

When the top node on max-heap H_C is a leaf node, we need to compute the exact influence values for all candidate locations under it. Let's denote the picked leaf node as n_C. According to Property 2, we can compute influence regions and relevant sets for candidate location from n_C's relevant sets rather than from scratch. In other words, from the definition of relevant sets, $n_C.S_F$ and $n_C.S_M$ are the only customers and existing facilities that need to be considered in computing the influence value of $c \in n_C$, i.e. $c.S_F \subset n_C.S_F$, $c.S_M \subset n_C.S_M$.

Naively, we could perform the sequential scan styled computation on relevant sets to obtain the exact influence value for c. However, thanks to the MBRs we stored in relevant sets, it is possible to further tighten the scan space with simple geometric checks. Again, the primary idea is that very distant customers are not likely to be influenced due to the existence of existing facilities near them. Also, very distant existing facilities cannot affect the influence value of a candidate since the customers can only be influenced by either nearby existing facilities or the candidate facility.

Formally, given candidate location c under leaf node n_C, we first find 4 nearest F nodes in the relevant F set of $n_C.S_F$. Here, the distance criterion used is the maximum distance between any point in r_F and c, denoted as $MaxDist(c, r_F)$. We use the furthest node to c in each nearest r_F to draw a perpendicular bisector so together we have a Voronoi cell for c. Similar to the technique in [27], the Voronoi cell is bounded by a minimum rectangle denoted as $c.R$. According to the property of Voronoi cells, only customers located in this $c.R$ can be influenced by c. To prune relevant F sets, we extend $c.R$ by doubling the distances between rectangle vertices and c, obtaining a new rectangle denoted as $c.R'$. According to the conclusion in [27], all existing facilities outside this $c.R'$ can be pruned when computing the influence of c. Using this properties, we have $c.S_F = \{n_F \in n_C.S_F, r_F \bigcap c.R' \neq \emptyset\}$ and $c.S_M = \{n_M \in n_C.S_M, r_M \bigcap c.R \neq \emptyset\}$.

Figure 4.6 gives an example of the technique described above, where $n_{F_{1-5}}$ are 5 F nodes in $n_C.S_F$, n_M is a M node in $n_C.S_M$. According to the property of bounding rectangles on Voronoi cells and the conclusion in [27], n_{F_5} and n_M can be pruned out from $c.S_F$ and $c.S_M$, respectively.

After $c.S_F$ and $c.S_M$ are refined, we can then employ sequential scan on them to obtain the exact influence value for c. However, since we still have R-Tree nodes in the relevant sets, it is possible to take further advantage of the property of MBRs before diving into distance computations between locations. Our basic idea is to avoid executing redundant distance computations for existing facility locations which simply give no helpful information. Specifically, given c and a customer location m, we want to prune n_F from $c.S_F$. To achieve this, we use a distance metric named **MinMaxDist**(m, r_F), which was originally introduced in [24]. The $MinMaxDist$ between a point m and an MBR r_F is defined as the minimum distance from m, within which at least one point enclosed in r_F can be found. Note that according to the R-tree definition, there is at least one point on each edge of a given MBR. Therefore, we follow [35] to define

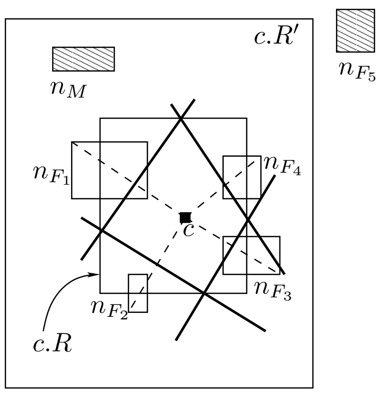


Figure 4.6: $n_{F_{1-4}}$ are 4 nearest $n_F \in n_C.S_F$, n_M can be pruned from $c.S_M$ since it lies outside $c.R$, n_{F_5} can be pruned from $c.S_F$ since it lies outside $c.R'$

the $MinMaxDist$ as the 2^{nd} smallest distance among $d(m, f_{1-4})$ where f_{1-4} represent the four vertexes of r_F.

Together with the distance metric $MinDist(m, r_F)$, we can obtain the following pruning rules. Given m and c,

1. If for $n_F \in c.S_F$, $MinDist(m, r_F) \geq d(c, m)$, then n_F can be pruned from $c.S_F$; This is because if $MinDist(m, r_F) \geq d(c, m)$, $\forall f \in r_F$, $d(f, m) \geq d(c, m)$, i.e., whether c influences m is irrelevant to r_F;

2. If for $n_F \in c.S_F$, $MinMaxDist(m, r_F) < d(c, m)$, c cannot influence m; This is because according to definition of $MinMaxDist$, $\exists f \in r_F$ that $d(f, m) < d(c, m)$;

3. If for $n_F \in c.S_F$, $MinDist(m, r_F) < d(c, m)$ and $MinMaxDist(m, r_F) \geq d(c, m)$, n_F should be preserved in $c.S_F$ for further computation; Since we are not sure whether there is a $f \in r_F$ that $MinDist(m, r_F) < d(f, m) < MinMaxDist(m, r_F)$, we are not sure whether c can influence m given r_f.

Figure 4.7 gives an example of the above pruning rules. On determining whether a given c influences m, n_{F_1} should be discarded since it provides no information for the computation; n_{F_4} should be preserved in $c.S_F$ for further computations; and if either n_{F_2} or n_{F_3} is present, then we can be sure c cannot influence m, so we can skip checking this m.

4.3.3 Complexity of BIP

The computational cost of BIP can be divided into index construction, pruning and exact influence computation. The R-Tree indexes needed can be constructed in $O(n \log n)$.

In the best case, only $O(k)$ most influential candidates are accessed. Thus only $kO(\log n)$ nodes n_C in t_C is accessed. This is because if the pruning

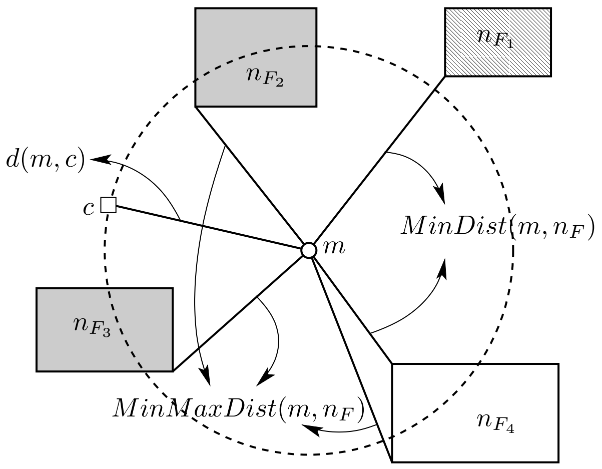


Figure 4.7: n_{F_1} can be discarded; n_{F_4} suggests c won't influence m; n_{F_2} and n_{F_3} should be further expanded to help determining whether c influences m

technique works well, most of the irrelevant F and M locations are pruned after the construction of the influence region and relevant sets in the beginning with $O(n)$ cost. Thanks to Property 2, subsequent computations on influence upper bounds and relevant sets consume $O(1)$. Also, the exact influence computation procedure only need to consume $O(k) \cdot O(c.S_M) \cdot O(c.S_F) = O(k)$. Thus, the overall run-time complexity of BIP in the best case is $O(n \log n) + O(n) + O(k) = O(n \log n)$.

In the worst case, all candidates are accessed, therefore all nodes in t_C are accessed. For each accessed node n_C in t_C, an $O(n)$ cost is needed to compute the influence upper bound and relevant sets. Also, an $O(n)$ cost is again needed for further pruning of $c.S_M$ and $c.S_F$ before computing the exact influence. Due to ineffectiveness of pruning, a cost of $O(n) \cdot O(n) \cdot O(n) = O(n^3)$ is needed when computing the exact influence values. To summarize, the overall run-time complexity of BIP in the worst case is $O(n \log n) + O(n^2) + O(n^3) = O(n^3)$.

That is to say, the bottleneck of BIP could be the exact influence value computation since it simply follows a three-layer loop on all three data sets. Yet as demonstrated in Chapter 5, BIP beats SS, which is of a $O(n^2)$ complexity, in terms of execution time. This suggests that in both synthetic and real world data sets, BIP can achieve a reasonable pruning performance, substantially eliminating unnecessary exact influence computations.

Similar to the analysis for EEP, when we perceive $|M|$ as the dominating data set in the input, the complexity of BIP can be represented by $O(|M| \log |M|)$ in both the best case and the worst case. This is because BIP centers on candidate tree t_C and tries to avoids repeated access on M. Even in the worst case, the pruning phase costs $O(|M|)$ and exact influence computation also only costs $O(|M|)$, leaving the overall cost to be $O(|M| \log |M|)$.

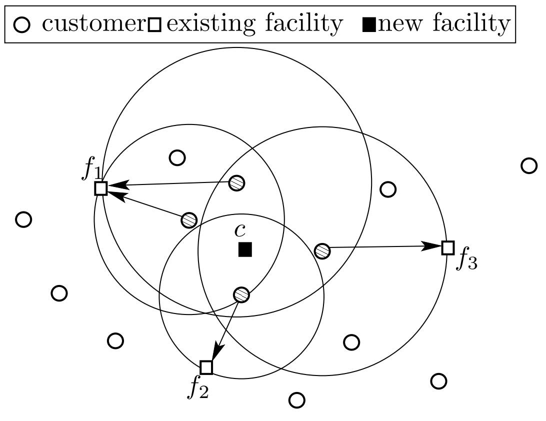


Figure 4.8: c lies in 4 nearest facility circles of customers(shaded ones), therefore $I_c = 4$

4.4 Nearest Facility Circle

Though both EEP and BIP introduced in the last sub sections prune unnecessary influence computations for unpopular candidate locations, they still need to extensively traverse t_F or t_M to obtain hints on pruning C. Remember that Definition 4 gives the direct way to obtain the answer, based on which we get the sequential scan algorithm. The implementation of that idea can be enhanced by dividing the process into to two problems. In the first problem we can apply classic geometric algorithms and use R-Tree to index the information computed by the first algorithm instead of the original data sets to boost query answering efficiency.

First, let us define *nearest facility circle*(nfc) for a customer location. The nearest facility circle of a customer m is a circle centered at m, with a radius of $min_{\forall f \in F}(d(f, m))$. Let $m.nfc$ denote this circle, and $m.nfd$ denote its radius. A facility would be the nearest facility of a given customer m if and only if the facility lies in the nearest facility circle $m.nfc$. Hence, the potential influence can be computed as the number of the nearest facility circles the facility lies in if it is built on that candidate location. For example, in Figure 4.8, the new facility c is located in four nearest facility circles of the shaded customers, therefore its potential influence is 4. By computing nearest facility circles for all customers first, we could reduce the influence computing procedure to point enclosure queries on an R-Tree indexing the circles.

Specifically, we still need to find the nfd for each m just like in the sequential scan algorithm. However, since we need to find all of m's nearest neighbors in another set F, it can actually be solved as an all nearest neighbors problem, where the task is to find the nearest neighbors for all data points. After obtaining nfds, instead of scanning M for every c, we can construct an R-Tree index for these $m.nfc$s. With this R-Tree, we can obtain influence of c by performing a point enclosure query to count the circles in which it is located. This simplifies the previous repeated scan process on M tremendously. Since this algorithm is based on R-Tree indexing the nearest facility circle, we name it the *Nearest Facility Circle* algorithm. Its pseudo-code is shown in Algorithm 5.

Algorithm 5: Nearest Facility Circle (NFC)

```
  Input: $k$, customer set $M$, existing facility set $F$, candidate location set $C$
  Output: TopInf(k, M, F, C)
1 compute $m.nfd$ for all $m \in M$ on $F$ using the all nearest neighbor algorithm
2 construct an R-Tree $tree$ indexing $m.nfc$
3 foreach $c \in C$ do
4     use $c$ to perform point enclosure query on $tree$ to obtain $I_c$
5 Sort $C$ by $I_c$
6 TopInf(k, M, F, C) $\leftarrow$ First $k$ locations in $C$
```

It is known algorithms such as the k-d tree based nearest neighbor search algorithm and the Voronoi diagram algorithm can solve all nearest neighbor problem on the plane with a cost of $O(n \log n)$ [31] [40]. Thus, the nearest facility circles can be computed with a cost of $O(n \log n)$. The construction of an R-Tree index tree takes $O(n \log n)$ as well, while the point enclosure queries take at best $O(n \log n)$ and at worst $O(n^2)$ cost. To summarize, the run-time complexity of NFC algorithm is $O(n \log n)$ in the best case and $O(n^2)$ in the worst case. In practical situations, where the customer data set is likely to be much larger than both facility and candidate sets, the most expensive steps in NFC are computing the nearest facilities for every customer and constructing an R-tree on customers' nearest facility circles, both of which incur $O(n \log n)$ cost. In other words, when $|M|$ dominates the problem scale, the overall cost of NFC is $O(|M| \log |M|) + O(\log |M|) = O(|M| \log |M|)$. This explains the advantages NFC maintains over EEP and BIP since NFC manages to achieve an $O(|M| \log |M|)$ complexity without using any complicated pruning techniques that are employed by EEP and BIP.

4.5 Nearest Facility Circle Join

It is easy to observe that in NFC algorithm, the candidates are considered individually, which may incur high cost in the online query procedure if the number of candidates climbs. In this section, we propose to construct another R-Tree for the candidate set such that the online influence querying procedure can be done via joining this candidate R-Tree with the NFC R-tree. The expected improvement brought by the additional R-Tree will be the competitive CPU time and much less I/O operations as the query procedure is done for a group of candidates altogether. We denote this algorithm as *Nearest Facility Circle Join* (NFCJ). Algorithm 6 shows the steps of NFCJ, which simply replace the point enclosure query with a classic intersection join procedure over the two R-trees [4] (line 4).

Specifically, the join procedure begins from the roots of two trees and recursively call a method (shown in Algorithm 7) to join the child nodes of two given nodes from the candidate R-tree and the NFC R-tree. Here, two nodes should be joined together if their MBRs intersect with each other. If both two to-be joined nodes are internal nodes (line 1-2), a plan sweep procedure is used to determine which child nodes should be joined. This procedure first sorts the child nodes of each node respectively, and then checks the intersection relationship between nodes in the two sorted queues. If either node is a leaf and the other is an internal, then it checks whether the child nodes of the internal one intersect with the leaf node and if so joins them together (line 3 - 10). When reaching to the leaf nodes, the counter for each candidate is added by the number of NFC joined with that candidate (line 11-13). After all leaf nodes in the candidate R-tree are joined with their intersected NFC, the counter of each candidate gives the influence value of that candidate.

Algorithm 6: Nearest Facility Circle Join (NFCJ)

```
Input: $k$, customer set $M$, existing facility set $F$, candidate location set
       $C$
Output: TopInf(k, M, F, C)
1 compute $m.nfd$ for all $m \in M$ on $F$ using the all nearest neighbor
  algorithm
2 construct an R-Tree $tree$ indexing $m.nfc$
3 construct an R-Tree $t_C$ indexing candidates $C$
4 $Join(tree, t_C)$ by counting the cardinality of joined nfcs for each $c \in C$
5 Sort $C$ by $I_c$
6 TopInf(k, M, F, C) $\leftarrow$ First $k$ locations in $C$
```

Because NFCJ still needs to compute the nearest facilities for each customer, and the R-tree two way join is of time complexity $O(n \log n)$ in average and $O(n^2)$ in worst, the theoretic time complexity of NFCJ is the same to NFC, i.e., $O(|M| \log |M|)$.

Algorithm 7: *Join*(n_F, n_C) in NFCJ

Input: A node n_F from NFC tree and a node n_C from t_C
Output: Joined nearest facility circles for each candidate

```
1  if $n_F$ is not leaf && $n_C$ is not leaf then
2      plane sweep child nodes of $n_F$ and $n_C$
3  if $n_F$ is leaf && $n_C$ is not leaf then
4      foreach child node $n'_C$ of $n_C$ do
5          if $n_F$ intersects $n'_C$ then
6              Join($n_F$, $n'_C$)
7  if $n_C$ is leaf && $n_F$ is not leaf then
8      foreach child node $n'_F$ of $n_F$ do
9          if $n'_F$ intersects $n_C$ then
10             Join($n'_F$, $n_C$)
11 if $n_F$ is leaf && $n_C$ is leaf then
12     if $n_F$ intersects $n_C$ then
13         $I_C++$
```

Chapter 5

Performance Study

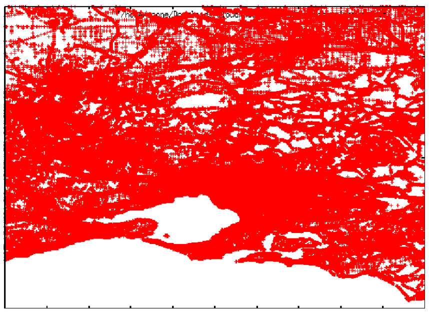
In this chapter, we present a performance study conducted on the algorithms described in Chapter 4. Notice that we are dealing with two dimensional space data sets, and we use two *double* variables to store a location. Hence the biggest data set used in the experiment, namely the $4M$ data set of M, only consumes 64 MB main memory. This suggests the problem is computation-intensive instead of I/O-intensive. In the remainder of this chapter, we still show results on both CPU time and I/O operation number for the sake of completeness. Additionally, to follow the tradition in the literature and to conduct a comparison of the true performance, we do not use any page buffering or cache scheduling techniques in our experiments. We incorporate the I/O operation cost into the CPU time cost to roughly compute a running time cost (by timing a 12ms random read latency of hard disk to the number of I/O operations) for EEP, BIP, NFC, and NFCJ.

5.1 Experimental Setup

We conduct all experiments on a workstation running CentOS 6.4 with a 3.2GHz six-core CPU and 6GB RAM memory. All algorithms are implemented in C++. Specifically, the implementation of the NFC and NFCJ algorithm used the methods of [2] to perform all nearest neighbor queries.

Both synthetic and real world data sets are used in our experiments. The synthetic data sets are generated to contain 20 clusters; each cluster in the Gaussian data sets and the Zipfian data sets follows Gaussian and Zipfian distribution, respectively. The real world data sets contain 1,758,928 and 2,330,014 real place locations in Melbourne, VIC, Australia and Los Angeles, CA, United States, respectively. This data are part of OpenStreet Map Project [21] and are pre-processed by CloudMade [10]. The distribution of the real world data sets are shown in Figure 5.1. For all data sets, we uniformly sample specific number of data points to form separate sets C, F, and M.

The parameters and data set cardinalities are listed in Table 5.1, where the default values are highlighted in bold style.


(a) Melbourne data sets

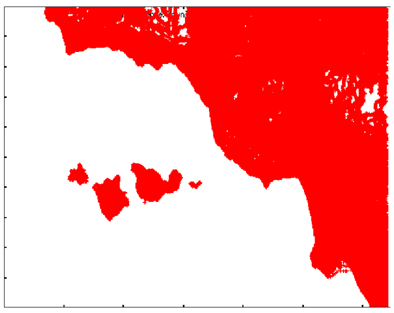

(b) Los Angeles data sets

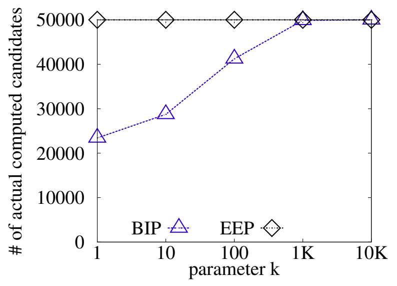
Figure 5.1: Real world data sets distribution

Table 5.1: Experiment configurations

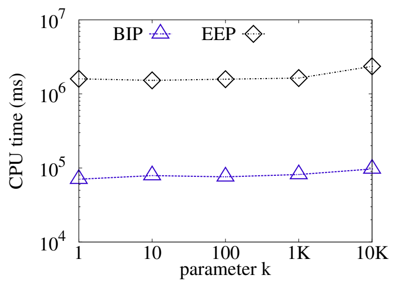
Parameter	**Synthetic Data**	**Melbourne**	**Loa Angeles**
$\|M\|$	500K, 1M, **2M**, 4M	200K, 400K, **800K**, 1.6M	250K, 500K, **1M**, 2M
$\|F\|$	5K, **10K**, 20K, 40K	2.5K, **5K**, 10K, 20K	5K, **10K**, 20K, 40K
$\|C\|$	25K, **50K**, 100K, 200K	10K, 20K **40K**, 80K	25K, **50K**, 100K, 200K
k	1, **10**, 100, 1000, 10000	**10**	1, **10**, 100, 1000, 10000
Node Size	1K, **2K**, 4K	**2K**	1K, **2K**, 4K
Distribution	Gaussian, Zipfian	Real	Real

5.2 Effect of Parameter k

As different values of parameter k are expected to affect the pruning power of the proposed branch and bound algorithms BIP and EEP, we conduct experiments using default configuration. Figure 5.2 gives the results of the performance of BIP and EEP when varying k from 1 to 10000 on the Zipfian data sets. Results on other data sets are similar thus omitted here. Though the two algorithms are designed to leverage the parameter k to early terminate as described in Section 4.2 and Section 4.3, it turns out that in most cases their pruning power is limited on the candidate sets. While BIP can prune around half of the candidates when k is small, EEP can rarely prune any in our experiments. This can be explained by two reasons. First, during the procedure of pruning in both algorithms, all three R-trees are involved; as each R-tree bounds the branch by aggregating the spatial information, involving all of them renders the obtained bounds significantly less effective. Second, both branch and bound algorithms aim at pruning the existing facility and the customer set in addition to the candidate set; the less effectiveness on the candidate sets may not reflect the

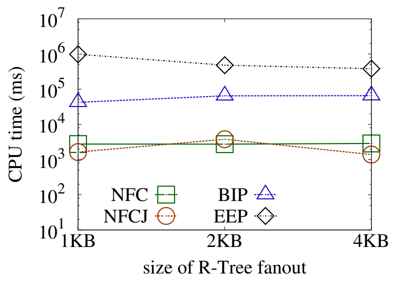


(a) # of computed candidates, Zipfian

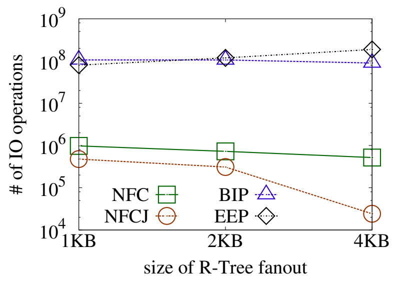


(b) CPU time, Zipfian

Figure 5.2: Effect of parmeter k



(a) CPU time, Los Angeles



(b) # of I/O, Los Angeles

Figure 5.3: Effect of fanout

pruning power of the algorithms on the other two sets. As the result, the performance of the algorithms in terms of CPU time is relatively insensitive towards the value of k, which is demonstrated in Figure 5.2b.

5.3 Effect of Node Size (fanout)

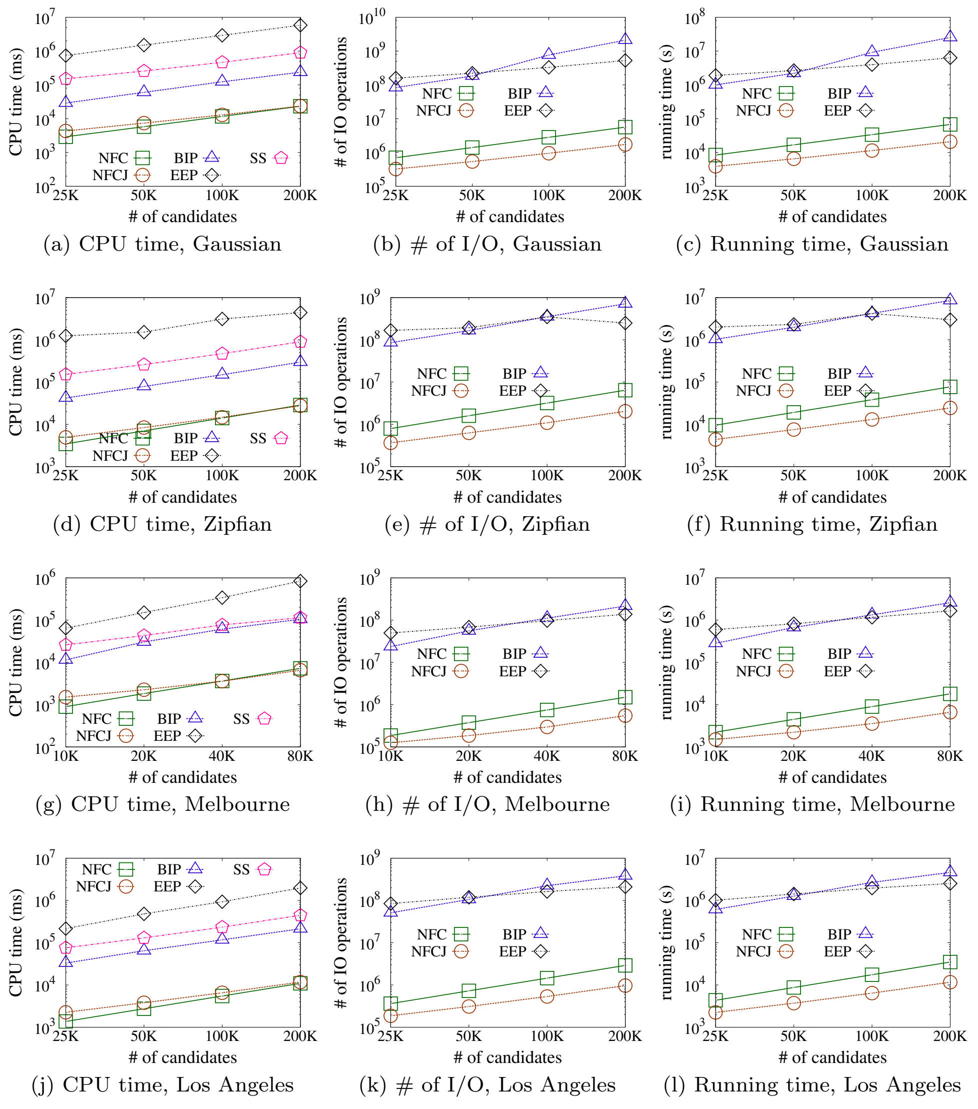
We vary the R-tree node size to study its effect on the performance of the proposed algorithms. The corresponding results on the Los Angeles data sets are shown in Figure 5.3. As demonstrated in these figures, in all cases, both NFC and NFCJ significantly outperforms EEP and BIP in two orders of magnitude. While the increased size of node does decrease the I/O performance of both NFC and NFCJ gradually, it has little effect on the CPU time cost of NFC and NFCJ. This is because for the same amount of data, bigger nodes lead shorter trees but do not bring significant benefits on distance computation. Both EEP and BIP are less sensitive towards the variation of node size as the variation here is not significant, it may take a much larger variation on the tree height to affect the performance of branch and bound algorithms. It can also be observed that when the node size is $2K$, the CPU time of NFCJ turns out to be outperformed by NFC; this can be explained by that the performance of R-tree join is highly dependent upon the (spatial) grouping arrangement of R-tree, when the node size is set to $2K$, the overlapping regions between different branches in the candidate R-tree could be relatively large that the efficiency earned by grouping query is overweighted by traversing multiple branches of one tree for every node in the other tree.

5.4 Effect of Data Set Cardinality

In this section, we study the effect of the cardinality of each data set has on all algorithms. For each case, the CPU time, the number of I/O operations and the running time derived by summing the I/O cost (timed by 12ms latency per operation) and the CPU time cost are shown respectively for all algorithms.

Effect of candidate cardinality $|C|$. It is expected that more candidates result to higher query cost. We vary the cardinality of candidate set and list the performance of all algorithms on four data sets in Figure 5.4. Overall, the performance of all algorithms degrades gradually as the cardinality of C grows. NFC and NFCJ outperform all other algorithm to up two, four, and three orders of magnitudes in terms of the CPU time, the number of I/O operations, and the running time, respectively. EEP turns out to be less efficient than SS. The reason for this is that EEP traverses the tree guided by the importance of nodes; it may follow a traversal order more like breadth first search such that it degrades to SS at the bottom of the trees but has additional overhead on attempt of pruning. This suggests that the performance of EEP tends to have a time complexity close to that in the worst case. BIP outperforms SS by almost one order of magnitude in terms of CPU time. The reason behind this is that BIP follows a best first search more like a depth-first search, i.e., computing the exact influence values for part of the candidates first and use them as the guide during traversal. The overall performance result fit to our theoretic analysis that NFC and NFCJ have the lower time complexity comparing to both BIP and EEP. Additionally, in terms of number of operations, NFCJ constantly beats NFC by several times, justifying the intuition of grouping point enclosure queries by using the candidate R-tree. This advantage makes NFCJ the best performing algorithm in terms of running time cost.

Effect of facility cardinality $|F|$. The cardinality of the facility set only has a linear effect on the performance when the sequential scan method is used. However, it may have significant effect on algorithms that utilize the facility set to prune the computations. We vary this cardinality and show the results of the performance of all algorithms in Figure 5.5. As illustrated in the figures, the performance of BIP and NFC is improved while that of NFCJ stays the same when the number of existing facilities grows. For BIP, the effect on CPU time is two-edge: the more existing facilities, the smaller the influence region for each candidate node, yet also the more facilities to be evaluated to see whether they are relevant. These two factors explain the first decreasing then increasing CPU time cost in Figure 5.5a and Figure 5.5d, and also the slightly increasing CPU time cost in Figure 5.5g and Figure 5.5j. However, the benefit of more existing facilities, i.e., the greater pruning power, does gain advantage in terms of the number of I/O operations, as depicted in Figure 5.5b, Figure 5.5e, Figure 5.5h, and Figure 5.5k. For NFC, the more existing facilities indicate the



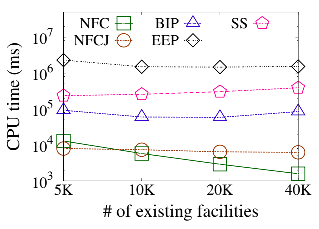
(a) CPU time, Gaussian (b) # of I/O, Gaussian (c) Running time, Gaussian

(d) CPU time, Zipfian (e) # of I/O, Zipfian (f) Running time, Zipfian

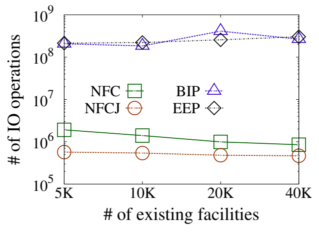
(g) CPU time, Melbourne (h) # of I/O, Melbourne (i) Running time, Melbourne

(j) CPU time, Los Angeles (k) # of I/O, Los Angeles (l) Running time, Los Angeles

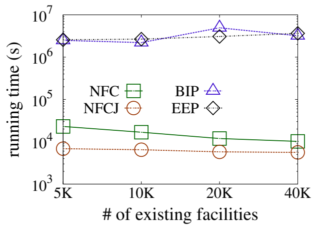
Figure 5.4: Effect of $|C|$ on algorithm performance



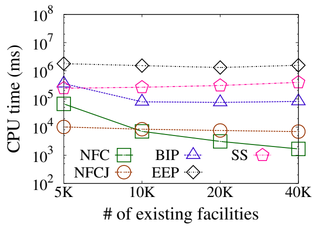
(a) CPU time, Gaussian



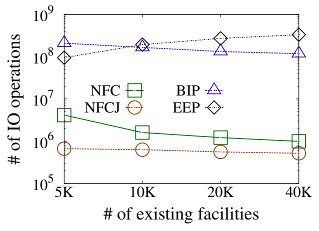
(b) # of I/O, Gaussian



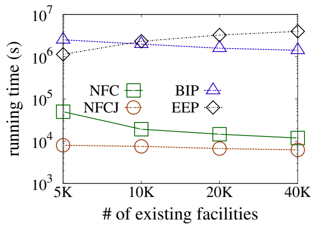
(c) Running time, Gaussian



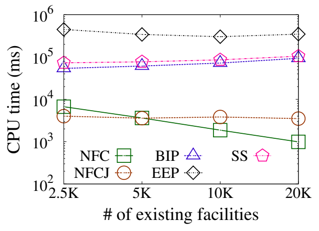
(d) CPU time, Zipfian



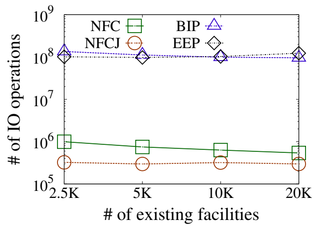
(e) # of I/O, Zipfian



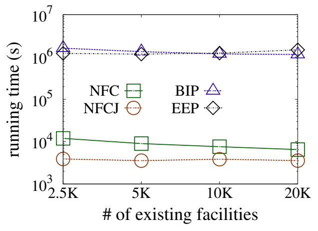
(f) Running time, Zipfian



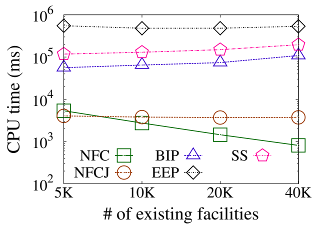
(g) CPU time, Melbourne



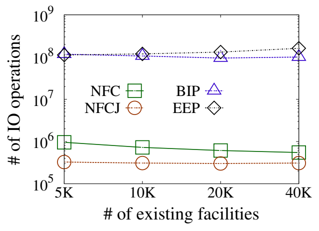
(h) # of I/O, Melbourne



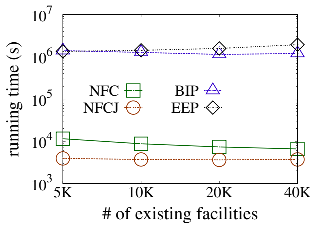
(i) Running time, Melbourne



(j) CPU time, Los Angeles



(k) # of I/O, Los Angeles



(l) Running time, Los Angeles

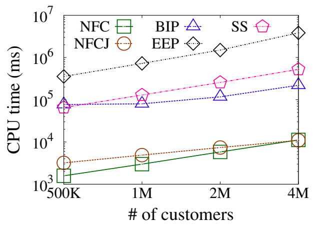
Figure 5.5: Effect of $|F|$

smaller each nearest facility circle will be, which result to the better organized R-tree as less nearest facility circles are likely to overlap with each other. This is why NFC enjoys the greater existing facility set F in all cases shown in the figures. However, NFCJ is not sensitive towards the varying F as it groups candidates and nearest facility circles together and traverse both trees level by level; it does not matter much if many nodes are overlapped with each other as grouping effect may diminish the disadvantage brought by overlapping nodes, e.g., extensive traversals will be shared. Overall, NFCJ and NFC beat SS, EEP, and BIP in orders of magnitude. NFC outperforms NFCJ in terms of CPU time when facility set grows bigger, NFCJ is still the best among all algorithm which achieve the lowest number of I/O operations and therefore the smallest running time cost in all cases.

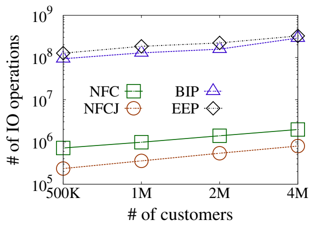
Effect of customer cardinality $|M|$. More customers may increase the influence values of facilities and it is expected to consume more time as well as I/O operations. The results of varying the cardinality of M are given in Figure 5.6. In all cases, the performance of all algorithms degrades as the cardinality of M grows. The ranking of algorithms in terms of their efficiency still stays the same, i.e., NFCJ and NFC beat other algorithms by up to two orders of magnitude while BIP outperforms SS and SS outperforms EEP. It can also be observed that BIP is more steady when compared to SS when the cardinality of the customer set grows. The reason is that BIP tends to avoid querying the customer R-tree and does not extensively traverse this customer R-tree when pruning. Hence, the growth of customer does not directly affect the performance of BIP. To summarize, the results here are similar to what has been shown in previous experiments and fit to our theoretic analysis in previous sections.

5.5 Summary of Experimental Results

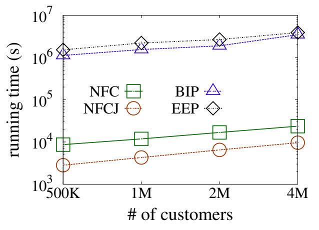
According to the results and analysis, in most cases NFCJ is the best choice for answering the top-k most influential location selection query, while NFC remains a reasonable alternatives when the existing facility set is rather large and the CPU time is the dominant concern.



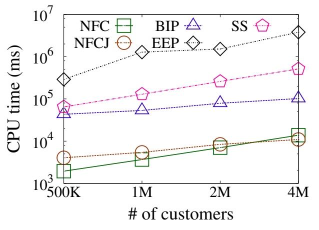
(a) CPU time, Gaussian



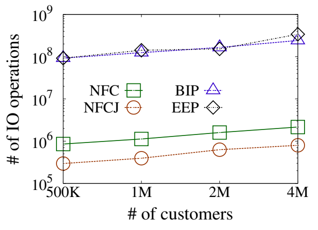
(b) # of I/O, Gaussian



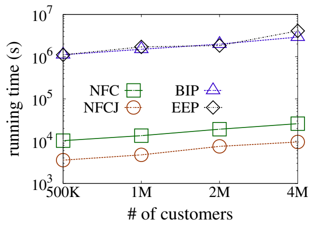
(c) Running time, Gaussian



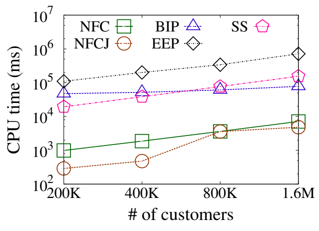
(d) CPU time, Zipfian



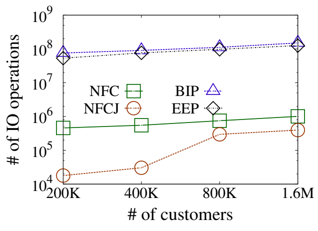
(e) # of I/O, Zipfian



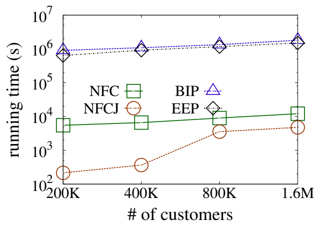
(f) Running time, Zipfian



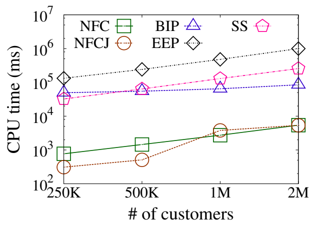
(g) CPU time, Melbourne



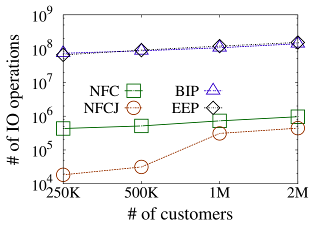
(h) # of I/O, Melbourne



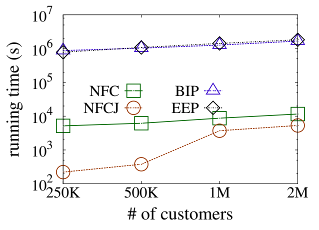
(i) Running time, Melbourne



(j) CPU time, Los Angeles



(k) # of I/O, Los Angeles



(l) Running time, Los Angeles

Figure 5.6: Effect of $|M|$

Chapter 6

Conclusion

We formulated a practical location selection problem using reverse nearest neighbor semantics, proposing a novel top-k most influential location selection query. We presented several algorithms for processing the query, namely Sequential Scan (SS), Estimation Expanding Pruning (EEP), Bounding Influence Pruning (BIP), Nearest Facility Circle (NFC), and NFC join (NFCJ), together with thorough analysis. We further provided an extensive experimental study on them and the results agree with the analysis. The results confirm that among all algorithms, NFCJ is the most efficient algorithm in terms of both time and I/O operations in most cases.

Bibliography

[1] Achtert, E., Kriegel, H.P., Krger, P., Renz, M., Zfle, A.: Reverse k-nearest neighbor search in dynamic and general metric databases. In: Proc. of EDBT (2009)

[2] ANNLibrary: http://www.cs.umd.edu/∼mount/ann/ (2011)

[3] Aronovich, L., Spiegler, I.: Bulk Construction of Dynamic Clustered Metric Trees. Knowledge and Information Systems **22**(2), 211–244 (2009)

[4] Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: Proc. of SIGMOD (1993)

[5] Cabello, S., D, J.M., Langerman, S., Seara, C.: Reverse Facility Location Problems. In: Proc. of CCCG (2006)

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

[7] Cheema, M.A., Lin, X., Zhang, W., Zhang, Y.: Influence Zone : Efficiently Processing Reverse k Nearest Neighbors Queries. In: Proc. of ICDE (2011)

[8] Cheema, M.A., Zhang, W., Lin, X., Zhang, Y.: Efficiently Processing Snapshot and Continuous Reverse k Nearest Neighbors Queries. The VLDB Journal (2012)

[9] Chen, H., Liu, J., Furuse, K., Yu, J.X., Ohbo, N.: Indexing Expensive Functions for Efficient Multi-Dimensional Similarity Search. Knowledge and Information Systems **27**(2), 165–192 (2010)

[10] CloudMade: http://downloads.cloudmade.com/ (2013)

[11] Du, Y., Zhang, D., Xia, T.: The Optimal-Location Query. Advances in Spatial and Temporal Databases **3633**, 163–180 (2005)

[12] Gao, Y., Zheng, B., Chen, G., Li, Q.: Optimal-Location-Selection Query Processing in Spatial Databases. IEEE TKDE **68**(8), 1162–1177 (2009)

[13] Ghaemi, P., Shahabi, K., Wilson, J.P., Banaei-Kashani, F.: Optimal network location queries. In: Proc. of GIS (2010)

[14] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: Proc. of SIGMOD, pp. 47–57 (1984)

[15] Huang, J., Wen, Z., Qi, J., Zhang, R., Chen, J., He, Z.: Top-k Most Influential Location Selection. In: Proc. of CIKM (2011)

[16] Korn, F., Muthukrishnan, S.: Influence Sets Based on Reverse Nearest Neighbor Queries. In: Proc. of SIGMOD (2000)

[17] Liu, X., Wu, X., Wang, H., Zhang, R., Bailey, J., Ramamohanarao, K.: Mining distribution change in stock order streams. In: ICDE, pp. 105–108 (2010)

[18] Mouratidis, K., Papadias, D., Papadimitriou, S.: Medoid Queries in Large Spatial Databases. In: Proc. of SSTD, pp. 55–72 (2005)

[19] Nutanong, S., Tanin, E., Zhang, R.: Incremental evaluation of visible nearest neighbor queries. TKDE **22**(5), 665–681 (2010)

[20] 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)

[21] OpenStreetMap: http://www.openstreetmap.org/ (2013)

[22] Qi, J., Zhang, R., Kulik, L., Lin, D., Xue, Y.: The Min-dist Location Selection Query. In: Proc. of ICDE (2012)

[23] Qi, J., Zhang, R., Wang, Y., Xue, A., Yu, G., Kulik, L.: The min-dist location selection and facility replacement queries. World Wide Web pp. 1–33 (2013)

[24] Roussopoulos, N., Kelley, S., Vincent, F.: Nearest Neighbor Queries. In: Proc. of SIGMOD, pp. 71–79 (1995)

[25] Shang, S., Yuan, B., Deng, K., Xie, K., Zhou, X.: Finding the Most Accessible Locations - Reverse Path Nearest Neighbor Query in Road Networks Categories and Subject Descriptors. In: Proc. of GIS (2011)

[26] SouFang: http://www.soufun.com (2013)

[27] Stanoi, I., Riedewald, M., Agrawal, D., Abbadi, A.E.: Discovery of Influence Sets in Frequently Updated Database. In: Proc. of VLDB (2001)

[28] Sun, Y., Huang, J., Chen, Y., Zhang, R., Du, X.: Location selection for utility maximization with capacity constraints. In: Proc. of CIKM (2012)

[29] Tao, Y., Lian, X.: Reverse kNN Search in Arbitrary Dimensionality. In: Proc. of VLDB (2004)

[30] trulia: http://trulia.com (2013)

[31] Vaidya, P.M.: AnO(n logn) Algorithm for the All-Nearest-Neighbors Problem. Discrete and Computational Geometry **4**(1) (1989)

[32] Wong, R.C.W., Özsu, M.T., Fu, A.W.C., Yu, P.S., Liu, L., Liu, Y.: Maximizing bichromatic reverse nearest neighbor for L p -norm in two- and three-dimensional spaces. The VLDB Journal **20**(6), 893–919 (2011)

[33] Wong, R.C.w., Ozsu, M.T., Yu, P.S., Fu, A.W.c., Liu, L.: Efficient Method for Maximizing Bichromatic Reverse Nearest Neighbor. In: Proc. of VLDB (2009)

[34] Wu, W., Yang, F., Chan, C.Y., Tan, K.L.: FINCH : Evaluating Reverse k-Nearest-Neighbor Queries on Location Data. In: Proc. of VLDB (2008)

[35] Xia, T., Zhang, D., Kanoulas, E., Du, Y.: On Computing Top-t Most Influential Spatial Sites. In: Proc. of VLDB (2005)

[36] Yan, D., Wong, R.C.w., Ng, W.: Efficient Methods for Finding Influential Locations with Adaptive Grids. In: Proc. of CIKM, pp. 1475–1484 (2011)

[37] Yang, C., Lin, K.i.: An index structure for efficient reverse nearest neighbor queries. In: Proc. of ICDE, pp. 485–492 (2001)

[38] Yu, C., Zhang, R., Huang, Y., Xiong, H.: High-dimensional knn joins with incremental updates. GeoInformatica **14**(1), 55–82 (2010)

[39] Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive Computation of the Min-Dist Optimal Location Query. In: Proc. of VLDB (2006)

[40] Zhang, J., Mamoulis, N., Papadias, D., Tao, Y.: All-Nearest-Neighbors Queries in Spatial Databases. In: Proc. of SSDM, pp. 297–306 (2004)

[41] 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)

[42] Zheng, K., Huang, Z., Zhou, A., Zhou, X.: Discovering the Most Influential Sites over Uncertain Data : A Rank Based Approach. IEEE TKDE (2011)