Top-k Most Influential Location Selection

Jin Huang[1], Zeyi Wen[2], Jianzhong Qi[2], Rui Zhang[2], Jian Chen[1], and Zhen He[3]

[1]South China University of Technology
[2]The University of Melbourne
[3]La Trobe University

Technical Report, 2011

Abstract

Facility location selection queries are important in many marketing applications and decision support systems. In this paper, we propose and study a new type of facility location selection query, the *top-k most influential location selection query*. Given a set M of customers and a set F of existing facilities, the top-k most influential location selection query finds k locations from a set C of candidate locations with the largest influence values, where the influence of a candidate location c ($c \in C$) is defined as the number of customers in M who are the reverse nearest neighbors of c. We first present a naive algorithm for processing the query. However, the naive algorithm is computationally expensive and not scalable to large datasets. This motivates us to explore more efficient solutions. We propose two branch and bound algorithms, the *Estimation Expanding Pruning (EEP)* algorithm and the *Bounding Influence Pruning (BIP)* algorithm. These two algorithms exploit various geometric properties to prune the search space, and thus achieve much better performance than that of the naive algorithm. Specifically, the EEP algorithm estimates the distances to the nearest existing facilities for the customers and the numbers of influenced customers for the candidate locations, and then gradually refines the estimation until the answer set is found, during which distance metric based pruning techniques are used to improve the refinement efficiency. BIP only estimates the numbers of influenced customers for the candidate locations. But it uses the existing facilities to limit the space for searching the influenced customers and achieve a better estimation, which results in an even more efficient algorithm. Extensive experiments conducted on both real and synthetic datasets validate the effectiveness of the prune techniques and the efficiency of the algorithms.

1 Introduction

A common problem many businesses and organizations share is finding a suitable place for the establishment of a new facility. For example, McDonald's may want to add a new restaurant to compete with other restaurants; a wireless service provider may want

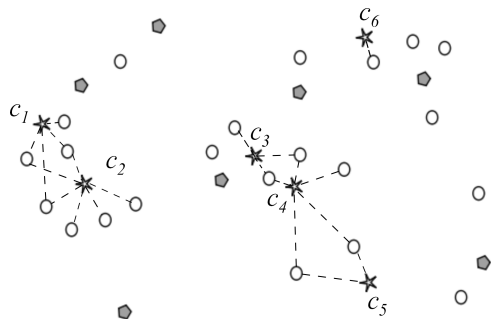


Figure 1: The top-k most influential location selection problem

to add a hotspot to a densely populated area to improve the quality of service. In most cases, there is a set of candidate locations to select from (e.g., real estate agents have databases of places for lease or sale). Then, an important business decision (for McDonald's or a wireless service provider) is to select a location that attracts as many customers as possible. Sometimes, one may want to first have a number (k) of such locations and then to make decisions with further considerations. In this paper, we investigate this problem of finding the top-k candidate locations that attract the largest numbers of customers, where k is a user input parameter. These k candidate locations can then be fed to further decision making procedures for selecting an overall best location for the new facility. Here, we assume a customer is attracted by her nearest facility and the business has the knowledge of customer distribution from surveys or past sales records.

An example of the above problem is given in Fig. 1, where circles, pentagons and stars denote customers, existing facilities and candidate locations, respectively. The candidate locations are labeled as $c_1, ..., c_6$. First, for every candidate location c_i, we compute the number of customers c_i can attract if a new facility is established at c_i. For example, if a new facility is established at c_1, then it will attract 4 customers. Similarly, we can compute the numbers for c_2, c_3, c_4, c_5 and c_6, which are 6, 3, 5, 2 and 1, respectively. Suppose now we want to have the top-3 candidate locations that attract the largest numbers of customers, i.e., $k = 3$. Then, c_2, c_4 and c_1 are returned as the answer set. We observe that adding a new facility to c_2 may reduce the use of existing facilities. This is expected as more facilities serve the same set of customers. Often, one of the main reasons for adding a new facility is that the current ones are overused or have an uneven load. In the extreme case, a company might even want to abandon an existing facility as its maintenance cost gets too high and our method can easily address this requirement.

The above facility location selection problem aims at maximizing the "influence" of the candidate locations, where the influence of a candidate location c denotes the number of customers whose respective nearest facility will be c if a new facility is established at c. Therefore, we formulate the above described problem as the top-k most influential location selection query. In this paper, we propose algorithms to process this query efficiently and make the following contributions.

- We formulate the top-k most influential location selection query.
- We analyze the properties of the query and propose pruning techniques to reduce the search space for processing the query.
- Based on the proposed pruning techniques, we propose two algorithms, namely, the Estimation Expanding Pruning algorithm (EEP) and the Bounding Influence Pruning (BIP) algorithm, to efficiently process the query.
- We perform an extensive experimental study using both synthetic datasets and real datasets. The results confirm the effectiveness of the proposed pruning techniques and the efficiency of the proposed algorithms. The improvement of the proposed algorithms over a naive algorithm is up to 10 times.

The rest of the paper is organized as follows. Section 2 reviews previous studies on related topics. Section 3 defines the top-k most influential location selection query and gives a naive algorithm. Section 4 and 5 describe our EEP algorithm and BIP algorithm, respectively. Section 6 presents the experimental results and Section 7 concludes the paper.

2 Related Work

There are many existing studies on facility location problems with various optimization functions. In this section, we review two major categories of facility location problems, namely, the *min-dist* problems and the *max-inf* problems. Most of these problems are closely related to the concept of *reverse nearest neighbor*. Therefore, we review the reverse nearest neighbor problem before reviewing the min-dist problems and the max-inf problems.

Reverse nearest neighbor problems: Korn and Muthukrishnan [10] first propose the reverse nearest neighbor (RNN) query and define the RNNs of an object o to be the objects whose respective nearest neighbor is o. In the same paper [10], Korn and Muthukrishnan propose an RNN-tree based solution to the RNN query, where the RNN-tree is an R-tree [8] 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. 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. Since RNN query was proposed, various techniques [18, 15, 1] have been proposed to process this type of query more efficiently and solve its variants under different settings. To solve the first problem, Yang and Lin [18] 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, and thus avoid the maintenance of an extra RNN-tree. To solve the second problem, Stanoi et al. [15] 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 large amount 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, but its problem solving techniques do not solve the facility location problems.

Min-dist problems: Min-dist facility location problems aim at minimizing the average distance between the customers and their respective nearest facilities. Zhang et al. [19] 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. [11] 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. 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 paticular facility. Hence, their solutions are inapplicable.

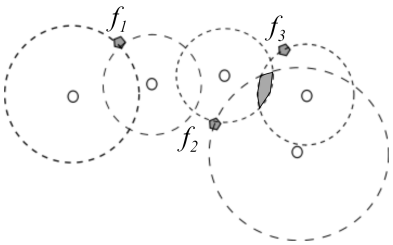


Figure 2: Example of the MAXCOV problem [2]

Max-inf problems: 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. Here, the concept of *"attract"* can have different meanings in different max-inf problems. Cabello et al. [2] propose a facility location problem based on the MAXCOV optimization criterion, which is to find regions in the data space to maximize the numbers of RNNs for the points in these regions. Figure 2 gives an example, where one of the gray region is the optimal region. Points in this region have 3 RNNs, while any point outside of this region has at most 2 RNNs. They introduce the concept of *nearest location circle (NLC)* to solve the problem, where the NLC 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 NLCs, which requires complex computations. The study gives a theoretical analysis, but no efficient algorithm is presented. Chen et al. [5] study this problem further and propose an efficient solution for finding the nearest facilities. Xia et al. [17] 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 example, in Figure 3, where the region Q is denoted by a rectangle and the existing sites are labeled with $f_1, f_2, ..., f_5$, the top-2 most influential sites query should return f_1 and f_2 since their influence values are both larger than f_3's influence value. Du et al. [6] propose to find a point from a continuous candidate region that can maximize the influence value. They use L_1 as the distance

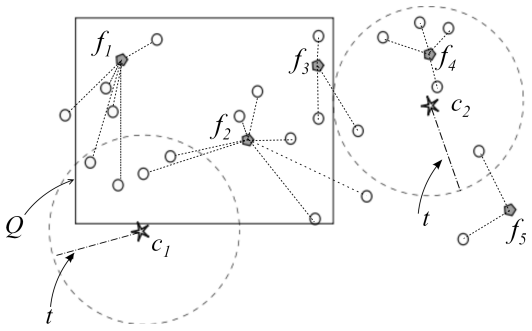


Figure 3: Examples of max-inf problems [17, 7]

metric 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. [4] 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. Similarly, Wong et al. [16] propose to find the optimal region in which all query object is of the maximum influence. Compared with this problem, our problem focuses on the numbers of RNNs of the candidate locations instead of specific locations of the RNNs. Unlike the above problems, which relate the influence values to the cardinalities of RNN sets, Gao et al. [7] propose to find the optimal location c outside a given region Q based on locations' optimality values, where the optimality of a location c is modeled by the amount of customers in Q whose distances to c is within a given threshold t. Figure 3 shows an example where the candidate locations are c_1 and c_2. c_1 is the problem answer since it attracts 4 customers in region Q whose respective distance to c_1 is within the distance threshold t while c_2 attracts no customer in Q. These problems use different settings from ours. Therefore, their solutions do not apply.

3 Preliminaries

In this section, we provide a formal definition of the top-k most influential location selection query and a naive algorithm to process the query. Table 1 summarizes the frequently used symbols.

3.1 Problem Definition

We assume three object sets, namely, a set of customers M, a set of existing facilities F and a set of candidate locations C. All the data objects (customers, existing facilities or candidate locations) are represented by points in an Euclidean space. We may refer to a data object as a data point or simply as a point. The distance between two points p_1 and p_2 are denoted as $d(p_1, p_2)$.

Table 1: Symbols

Symbol	Explanation
M, F, C	The sets of customers, existing facilities and candidate locations, respectively
m, f, c	A customer, an existing facility and a candidate location, respectively
p	A point in the data space
$d(p_1, p_2)$	The distance between two points p_1 and p_2
tr_M, tr_F, tr_C	R-trees on M, F and C, respectively
N_M, N_F, N_C	A node in tr_M, tr_F and tr_C, respectively
$N, N.mbr$	A node in an R-tree and its MBR

Given a customer $m \in M$, we denote her nearest existing facility as $nf(m)$, and call the distance between m and $nf(m)$, $d(m, nf(m))$, the *nearest facility distance (nfd)* of m. We say that customer m is attracted by a candidate location $c \in C$ if the distance between m and c is less than the nearest facility distance of m, i.e., $d(m, c) < d(m, nf(m))$. In this case, if we add a facility at c, it will become the new nearest facility of m.

A candidate location $c \in C$ may attract multiple customers. The number of these attracted customers defines the *influence* of c, denoted by $inf(c)$. Formally,

$$inf(c) = |\{m \in M | d(m, c) < d(m, nf(m))\}|.$$

The top-k most influential location selection query is to find k candidate locations from C that attract the largest numbers of customers.

Definition 1 *Top-k Most Influential Location Selection (TILS) Query: Given a constant k, a set of points M as customers, a set of points F as existing facilities and a set of points C as candidate locations, the top-k most influential location selection query finds a set $C' \subset C$ with k points, so that $\forall c_i \in C', c_j \in C \setminus C' : inf(c_i) \geq inf(c_j)$.*

We call the subset C' the *top-k most influential location set* and denote it as TopInf(k,M,F,C)[1]. The goal of TILS query is to find TopInf(k,M,F,C). Next, we present a naive algorithm to find this set.

3.2 A Naive Algorithm

A naive algorithm to process the TILS query is to first compute the nearest facility distance for every $m \in M$, which is done by scanning F to find m's nearest existing facility. Then, for every candidate location $c \in C$, we scan the customer set M. If the distance between c and a customer m, $d(c, m)$, is less than the nearest facility distance of m, i.e., c attracts m, then we increment the influence value of c. After we have scanned every pair of c and m, we obtain the influence value for every c. We then sort the candidate locations according to their influence values, and output the first k candidate locations in the sorted result as the query answer set.

[1]Note that there may be ties in the influence values. To simplify our discussion, we always return the first k candidate locations found that have the largest influence values.

Algorithm 1: A **NA**ive Algorithm

Input: k, customer set M, existing facility set F, candidate location set C
Output: TopInf(k, M, F, C)

```
1  foreach m ∈ M do
2      foreach f ∈ F do
3          if m.nfd > d(f, m) then
4              m.nfd ← d(f, m)
5  foreach c ∈ C do
6      foreach m ∈ M do
7          if m.nfd > d(c, m) then
8              c.inf++
9  Sort C by c.inf
10 TopInf(k, M, F, C) ← first k locations in C
```

We call this algorithm the *NA* algorithm and summarize it as Algorithm 1, where $m.nfd$ denotes the nearest facility distance of a customer m, and $c.inf$ denotes the influence of a candidate location c.

The NA algorithm is inefficient in that it has to scan every pair of customer and existing facility to find customers' nearest existing facilities, and every pair of candidate location and customer to compute candidate locations' influence values. This motivate us to explore effective pruning techniques to reduce the search space. We assume that the datasets are maintained in spatial indexes and propose two algorithms utilizing different geometric properties to achieve effective pruning. The first algorithm estimates the distances to the nearest existing facilities for the customers and the numbers of influenced customers for the candidate locations, and then use distance metric based pruning techniques to gradually refine the estimations until the answer set is found. The second algorithm only estimates the numbers of influenced customers for the candidate locations, but it uses the existing facilities to reduce the space for searching the influenced customers, and refine the estimation to achieve the query answer set. We assume the R-tree [8] (or its variant) as the spatial index, although our algorithms apply to any hierarchical index.

4 Estimation Expanding Pruning

In this section, we propose an algorithm that estimates the distances to the nearest existing facilities for the customers and the numbers of influenced customers for the candidate locations, and gradually refines these estimations to obtain k candidate locations with the largest influence values. We use two R-trees and an R-tree variant to index the three datasets. The **estimations** and the refinement are performed during a traversal on the trees (**expanding** tree nodes), where the *importance* value is introduced to help achieve a best first tree traversal while branch and bound techniques are introduced to **prune** the search space. We call this algorithm the *Estimation Expanding Pruning (EEP)* algorithm.

EEP indexes the set of existing facilities F and the set of candidate locations C

with two R-trees tr_F and tr_C, respectively. It indexes the set of customers M with an R-tree variant called the *aggregate R-tree (aR-tree)* [12] tr_M, where a tree node stores the number of data objects that are enclosed by its MBR in addition to what is stored in a regular R-tree node. These three trees are traversed simultaneously. The traversal is managed using three auxiliary lists L_M, L_F and L_C, which store objects (either nodes or data objects) from tr_M, tr_F and tr_C, respectively. For an object O in an auxiliary list, we define its *importance* value with its estimated distance to its nearest existing facility (if O is an object in the customer tree tr_M) or estimated number of influenced customers (if O is an object in the existing facility tree tr_F or the candidate location tree tr_C) (details are in Section 4.2). We use the importance value to determine which objects should be accessed first, and which objects should be pruned. Every object in these lists stores some *influence relation information* that indicates its position relation with the objects in other lists and will be used for importance value computation. Initially, each of these three lists contains only the root node of the corresponding tree. Then, these three lists are accessed repeatedly in the order of L_M, L_F and L_C. Algorithm 2 describes the above high level algorithm of EEP.

Algorithm 2: Estimation **E**xpanding **P**runing (EEP) Algorithm

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
2 Initialize the influence relation information for the nodes in the three lists
3 while |TopInf(k, M, F, C)| < k do
4     ExpandObject( L_M )
5     ExpandObject( L_F )
6     ExpandObject( L_C )
```

Every time a list L is accessed, EEP computes the importance value $imp(O)$ for every object O contained in L. The object in L with the largest importance value is accessed when EEP accesses L. When an object O is accessed, if O is a node, the following 5 steps are performed: (i) O's child objects (either nodes or data objects) use O's influence relation information to compute their own influence relation information according to their positions; (ii) EEP prunes a child object O' of O if the influence relation information of O' indicates that the data objects enclosed in O' will not influence or be influenced by the data objects enclosed in the objects of other lists; (iii) if O is a node in tr_C, we compute an influence value upper bound for each of its child objects O', denoted by $maxinf(O')$, which is an upper bound of the influence value for all the candidate locations enclosed by O'. We prune O' if $maxinf(O') < inf_{pr}$, where inf_{pr} is an influence threshold value used for pruning objects in tr_C from further consideration. (iv) the unpruned child objects are used to update the influence relation information of the objects in other lists and further prune some of the objects. (v) EEP removes O and inserts the unpruned child objects of O into L. If O is a candidate location $c \in C$, EEP computes the maximum and minimum possible influence values for c, denoted by $maxinf(c)$ and $mininf(c)$, and compare $mininf(c)$ with the largest

$maxinf$ value, inf_{mx}, for all the objects in L_C. If $mininf(c) \geq inf_{mx}$, then c is put in the query answer set TopInf(k,M,F,C) and the pruning influence value inf_{pr} is updated to $mininf(c)$. If O is an existing facility or a customer, there is no computation required for processing O. However, we cannot simply remove O because there are objects in other lists that are related to O. Therefore, EEP skips it and continues to access objects in other lists. The algorithm terminates when TopInf(k,M,F,C) is filled with k candidate locations and no other candidate location can have larger influence values than those of the candidate locations in TopInf(k,M,F,C). Details of the above steps are presented in Algorithms 3.

Algorithm 3: ExpandObject

Input: An auxiliary list L

```
1  Pick O with the largest imp(O) from L
2  if O is a node then
3      Expand O
4      foreach child object O' ∈ O do
5          Compute the influence relation information of O'
6          Update the influence relation information of the objects related to O'
7          Prune unpromising objects
8          if O' cannot be pruned then
9              Insert O' into L
10 if O is a data object then
11     if O ∈ C then
12         Compute maxinf(O) and mininf(O)
13         if maxinf(O) > inf_pr then
14             Put O in TopInf(k,M,F,C)
15             inf_pr ← mininf(O)
```

In what follows, we first explain the influence relation information in Section 4.1, based on which we then explain the object importance and the overall pruning process in Section 4.2 and Section 4.3, respectively.

4.1 The Influence Relation Information

The influence relation information helps EEP to compute an importance value for an object in one of the the auxiliary lists L_M, L_F or L_C, which is further used for object pruning. In L_M, an object O_M (a node in tr_M or a customer) stores a set of objects in L_F and a set of objects in L_C, which enclose O's possible nearest existing facilities and possible candidate locations, respectively. We denote these two sets by $O_M.S_F$ and $O_M.S_C$. These two sets are used to estimate O_M's distances to its nearest existing facility and its nearest candidate location. In L_F, an object O_F (a node in tr_F or an existing facility) stores a set of objects in L_M, which enclose the customers that are possibly influenced by O_F. We denote this set by $O_F.S_M$ and use it to estimate the number of customers influenced by O_F. In L_C, an object O_C (a node in tr_C or a candidate location) stores a set of objects in L_M, which enclose the customers that are possibly influenced by O_C. We denote this set by $O_C.S_M$ and use it to estimate the

number of customers influenced by O_C. (c.f. Figure 4)

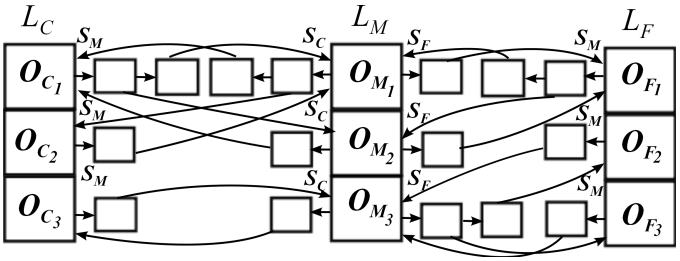


Figure 4: Influence relation sets

We call the sets described above the *influence relation sets*. To construct these sets, we use the following distance metrics to estimate position relations between objects (c.f. Figure 5). Given two objects O_1 and O_2, distance metrics **MinDist(**O_1**,** O_2**)** and **MaxDist(**O_1**,** O_2**)** are defined as the minimum and maximum distances between any points in the MBRs of O_1 and O_2, and distance metric **MinExistDist**$_{O_2}(O_1)$ is defined as the minimum upper bound of the distance from an object enclosed by the MBR of O_1 to its nearest object enclosed by the MBR of O_2, i.e., every object O_1' enclosed by the MBR of O_1 can find an object O_2' enclosed by the MBR of O_2 whose distance to O_1' is less than or equal to $MinExistDist_{O_2}(O_1)$. In other words, $MinExistDist_{O_2}(O_1)$ is the distance that guarantees any objects in O_1 to meet a neighbor in O_2. The following two theorems regarding the above distance metrics help determine which objects should be in the sets of $O_M.S_F$, $O_M.S_C$, $O_F.S_M$ and $O_C.S_M$, and which should not.

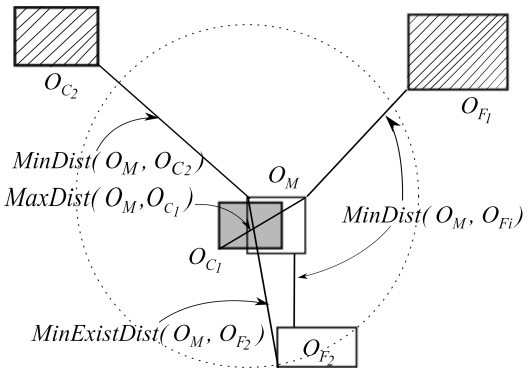


Figure 5: An example for Theorems 1 and 2

Theorem 1 *Given three objects O_M, O_F and O_C in the three lists L_M, L_F and L_C, respectively, if $MinDist(O_M, O_C) \geq MinExistDist_{O_F}(O_M)$, then $\forall m \in M$ and m enclosed by the MBR of O_M: m is not influenced by any c enclosed by the MBR of O_C.*

Intuitively this theorem says if a customer object O_M is farther away from the

closest possible candidate location in O_C than at least one existing facility in O_F then O_M is influenced by O_F, but O_M does not influence O_C.

Theorem 2 *Given an object O_M in L_M and an object O_C in L_C , if $\forall O_F \in O_M.S_F$: $MaxDist(O_M, O_C) < MinDist(O_M, O_F)$, then $\forall m \in M$ and m enclosed by the MBR of O_M, $\forall c \in C$ and c enclosed by the MBR of O_C: m is influenced by c.*

Intuitively this theorem says if a candidate location object O_C is closer to a customer object O_M than the closest possible existing facilities of O_M, then it can influence all the customers enclosed by O_M.

Proofs of the two theorems are straightforward based on the definitions of the distance metrics and thus are omitted.

Theorems 1 and 2 determine the influence relation between objects. To utilize them for influence relation set determination, we further compute and store the following two threshold distances with an object O_M in L_M to help determine the influence relation sets. The first threshold distance, denoted as $O_M.Td_{Min}$, defines an lower bound for the distance between a customer enclosed by the MBR of O_M and her nearest existing facility. Formally, $O_M.Td_{Min} = \min(\{MinDist(O_M, O_F) | \forall O_F \in O_M.S_F\})$. The second threshold distance, denoted as $O_M.Td_{Ext}$, defines an upper bound for the minimum distance between a customer enclosed by the MBR of O_M and her nearest existing facility. Formally,
$O_M.Td_{Ext} = \min(\{MinExistDist_{O_F}(O_M) \, | \forall O_F \in O_M.S_F\})$.

Using the above thresholds and Theorems 1 and 2 we arrive at three rules for pruning the influence relations sets. Given an object O_M in L_M, if there is an object O_C in L_C that satisfies $O_M.Td_{Min} > MaxDist(O_C, O_M)$, then according to Theorem 2, every candidate location enclosed by O_C can influence every customer enclosed by O_M. Therefore, we increase the minimum possible influence value of O_C, $mininf(O_C)$, by $|O_M|$, and then can remove O_M from $O_C.S_M$ and remove O_C from $O_M.S_C$ (**rule 1**). Also, if there is an object O_C in L_C satisfying $MinDist(O_C, O_M) \geq O_M.Td_{Ext}$, then according to Theorem 1, candidate locations enclosed by the MBR of O_C do not influence customers enclosed by O_M. Therefore, O_M is removed from $O_C.S_M$, O_C is removed from $O_M.S_C$, and $maxinf(O_C)$ is decreased by $|O_M|$ (**rule 2**).

Given an object O_F in L_F, if there is an object O_M in L_M that satisfies $MinDist(O_F, O_M) > O_M.Td_{Ext}$, then according to the definition of the distance metrics, O_F can be removed from $O_M.S_F$. Also, O_M should be removed from $O_F.S_M$ (**rule 3**).

Figure 5 gives an example, where O_M is an object in L_M, O_{F_1}, O_{F_2} are objects in L_F, and O_{C_1}, O_{C_2} are objects in L_C. Object O_{F_2} is closer to O_M than O_{F_1} and O_{C_2}. Thus, O_{F_2} is put in $O_M.S_F$ and O_M is put in $O_{F_2}.S_M$. Meanwhile, O_{C_1} is closer to O_M than O_{F_2}. Therefore, we also put O_{C_1} in $O_M.S_C$ and O_M in $O_{C_1}.S_M$.

4.2 The Importance of an Object

The importance values of objects determine the order of the objects are accessed in their respective auxiliary lists so that the search space can be reduced as much as possible. Since objects in different lists functions differently for search space pruning, we give

different definitions for the importance of objects in different lists respectively. To simplify the discussion, for an object O, we use $area(O)$ and $|O|$ to denote the area of the MBR of O and the number of data objects enclosed by the MBR of O. We use $|O.S|$ to denote the number of objects in O's influence relation set $O.S$.

For L_M, an object O_M that has a larger area or encloses a larger number of customers has a larger possibility to affect the influence relation information of objects in other lists, which is used for pruning those objects. Therefore, we define the importance value of O_M, $imp(O_M)$, as $|O_M| \cdot |O_M.S_C| \cdot |O_M.S_F| \cdot area(O_M)$.

For L_F, an object O_F's importance is mainly represented by its area and the number of objects in $O_F.S_M$. Therefore, we define the importance value of O_F, $imp(O_F)$, as $|O_F.S_M| \cdot area(O_F)$.

The TILS query aims at finding the candidate locations with the largest influence values. Thus, we always access first the object O_C in L_C that has the largest number of customers possibly influenced by O_C. Therefore, we define the importance value of O_C, $imp(O_C)$, as $maxinf(O_C)$, where $maxinf(O_C) = mininf(O_C) + \sum_{\forall O_M \in O_C.S_M} |O_M|$.

4.3 Overall Pruning Process

Now we summarize how all the techniques described above are integrated together to process the TILS query. As we traverse the three trees, every time an auxiliary list is accessed, we compute the importance values for the objects in the list as described in Section 4.2 and pick the object with the largest importance value to access. When an object O is accessed, if it is a tree node, then we construct the influence relation sets for its child objects and update the influence relation sets of the objects related to these child objects as described in Section 4.1. Once an object's influence relation set becomes empty, we prune this object from further consideration. If O is from L_C, we further check whether its importance value is smaller than the smallest influence value of all the currently found top-k candidate locations. If yes, we prune this object. After the pruning, we put the unpruned child objects into the corresponding auxiliary list. If O is a customer or an existing facility, we stop accessing its corresponding list and continue to access other lists. If O is a candidate location, we put it into the query answer set if we have not found k candidate locations whose smallest influence value is larger than O's influence value, or terminate the algorithm if O's influence value is not larger than that of any of the currently found top-k candidate locations.

4.4 Complexity Analysis

In the following, we analyze its complexity. Let t denote the average times of execution on while loop in Algorithm 2. The average sizes of three lists are denoted by m_1, f_1, and c_1, respectively. The average sizes of $O_C.S_M$, $O_F.S_M$, $O_M.S_C$, and $O_M.S_F$ are denoted as m_2, m_3, c_2, and f_2, respectively. Then Retrieving $imp(O_M)$, $imp(O_F)$ and $imp(O_C)$ takes $\mathcal{O}(m_1 + f_1 + c_1 \log c_1)$. Note that $imp(O_C)$ is obtained by sorting L_C. Expanding $imp(O_M)$ costs $\mathcal{O}(2 \cdot f_2 + c_2 \cdot (m_2 + c_2)) = \mathcal{O}(f_2 + c_2 \cdot m_2 + c_2^2)$, which is for updating the relation information for objects in other lists. Similarly, the cost of expanding $imp(O_F)$ is $\mathcal{O}(m_3 + m_3 \cdot f_2 + m_3 \cdot c_2 \cdot (m_2 + c_2)) = \mathcal{O}(m_3 \cdot f_2 + m_3 \cdot m_2 \cdot c_2 + m_3 \cdot c_2^2)$, and expanding $imp(O_C)$ costs $\mathcal{O}(m_2 + c_2)$. Thus the overall cost of EEP is $\mathcal{O}(t \cdot (m_1 + f_1 + c_1 \log c_1 + m_3 \cdot f_2 + m_2 \cdot m_3 \cdot c_2 + m_3 \cdot c_2^2))$.

- In the best case, most of M and F are pruned, so the sets $O_M.S_F$, $O_C.S_M$, $O_F.S_M$ and $O_M.S_C$ are very short and only one branch of each tree is traveled to achieve top-k answer. Thus, $f_2 \ll f_1$, $m_2 \ll m_1$, $m_3 \ll m_1$, $c_2 \ll c_1$, $\mathcal{O}(t) = \mathcal{O}(\log|C|)$, $\mathcal{O}(m_1) = \mathcal{O}(\log|M|)$, $\mathcal{O}(f_1) = \mathcal{O}(\log|F|)$, $\mathcal{O}(c_1) = \mathcal{O}(\log|C|)$. Therefore, the complexity of EEP is $\mathcal{O}(\log|C| \cdot (\log|M| + \log|F| + \log|C|))$.

- In the worst case, all the leaves of tr_M, tr_F and tr_C are pushed into the corresponding lists, the size of sets for relation information equal to the size of lists, and t equals to the datasets which has largest cardinality (we use $t = |M|$, since in practical $|M|$ is largest compared to $|F|$ and $|C|$). The complexity is $\mathcal{O}(|M|^2 \cdot (|F| + |M| \cdot |C| + |C|^2))$.

- In the average case, most of O_M and O_F are pruned, part of O_C needs to compute its exact influence value. Thus, the most time consuming part is retrieving $imp(O)$ which is of $\mathcal{O}(t \cdot (m_1 + f_1 + c_1 \log c_1))$. Let α denote the average number of branches need to be visited in R-Tree and we get $t = \alpha \cdot \log|C|$. Then, the complexity is $\mathcal{O}(\alpha^2 \cdot \log|C| \cdot (\log|M| + \log|F| + \log|C|))$, where α reflects the pruning efficiency of the algorithm.

5 Bounding Influence Pruning

In this section, we present a second strategy to estimate and refine the influence values for candidate locations. This strategy employs the concept of *influencing region*, which is a region computed from the existing facilities near a candidate location c to enclose all the customers who are influenced by c. The number of customers in this region forms an upper bound for c's influence value. By gradually refining the influencing region (**pruning** customers), we reduce c's **influence** value upper **bound** until we get c's exact influence value.

We present a branch and bound algorithm called the *Bounding Influence Pruning (BIP)* algorithm that utilizes the above strategy to process the TILS query. Like the EEP algorithm, the BIP algorithm also indexes the set of candidate locations C and the set of existing facilities F with two R-trees tr_C and tr_F. It indexes the set of customers M with an aR-tree tr_M. BIP also takes a best first approach. The algorithm uses a priority queue Q_C to perform a best first traversal on tr_C, where each queue element is a node N_C from tr_C, associated with a set of *relevant F nodes* $N_C.R_F$ from tr_F for influencing region computation, a set of relevant M nodes $N_C.R_M$ from tr_M for influence value estimation, and an estimated upper bound for the influence values of the candidate locations enclosed by the MBR of N_C as N_C's priority. To simplify the discussion, we call this upper bound the influence value upper bound of N_C. Initially, Q_C only contains the root node of tr_C, denoted as $root_C$, with $root_C.R_F = \{root_F\}$ and $root_C.R_M = \{root_M\}$, where $root_F$ and $root_M$ denote the root nodes of tr_F and tr_M, respectively.

Every time Q_C is accessed, the first node N_C in the queue is popped out. If N_C is a non-leaf node, the algorithm (i) retrieves N_C's child nodes, (ii) constructs their own sets of relevant nodes according to $N_C.R_F$ and $N_C.R_M$, (iii) computes their influencing regions and influence value upper bounds, (iv) removes a child node if its influence value upper bound is less than the smallest influence value of the candidate

locations in TopInf(k,M,F,C), inf_{pr}, and (v) pushes the unpruned child nodes into Q_C. If N_C is a leaf node, the algorithm (i) retrieves the candidate locations indexed in N_C, (ii) computes their own influencing regions and exact influence values using $N_C.R_F$ and $N_C.R_M$, (iii) inserts a candidate location into TopInf(k,M,F,C) if its influence value is larger than inf_{pr}, and (v) updates inf_{pr}. The algorithm terminates when Q_C is empty.

Algorithm 4: Bounding **I**nfluence **P**runing (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.R_F ← {root_F}, root_C.R_M ← {root_M}, Q_C.enqueue(root_C)
2  while Q_C ≠ φ do
3      N_C = Q_C.pop()
4      if N_C is leaf then
5          foreach c ∈ N_C do
6              Compute exact influence value c.inf
7              if c.inf > inf_pr then
8                  Update TopInf(k, M, F, C) and inf_pr
9      else
10         foreach child node N'_C of N_C do
11             Construct N_C'.R_M and N_C'.R_F;
12             Compute the influence value upper bound N'_C.maxinf
13             if N_C'.maxinf < inf_pr then
14                 Prune N'_C
15             else
16                 Q_C.push(N'_C)
```

The detailed steps of the BIP algorithm are presented in Algorithms 4. In what follows, we first explain how the sets of relevant nodes, the influence region and the influence value upper bound are computed in Section 5.1. Then we explain how we compute the exact influence values in Section 5.2.

5.1 Influencing Regions and Relevant Nodes

Given a node N_C in tr_C and a set S_F of nodes in tr_F, they define a region that encloses all the customers who are influenced by at least one candidate location enclosed by the MBR of N_C. We call this region the influencing region of N_C and say that the customers in this region are possibly influenced by N_C.

The computation of the influence region utilizes the following idea. Given two points p_1 and p_2, their *perpendicular bisector* divides the whole plane $\mathbb{P}$ into two regions. If a point q is influenced by p_1, then it must lie in the same region as p_1. Cheema et al. [3] proofed a theorem that extends this idea from points to rectangles. Given two rectangles R_1 and R_2, this theorem uses multiple lines to divide $\mathbb{P}$ into two regions. Among them, one contains all the points that are influenced by R_1.

This theorem requires the concept of *antipodal corners* to define the *normalized*

perpendicular bisectors, denoted by NB, which serve as the boundary lines for the influencing region (c.f. Figure 6).

Definition 2 *Antipodal Corners: Let a rectangle R's lower left, lower right, upper left and upper right corners be $R.ll$, $R.lr$, $R.ul$ and $R.ur$, respectively. Given two rectangles 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 3 *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*

$$p_{ac_1,ac_2}.x = \begin{cases} \frac{R_1.ur.x+R_2.ur.x}{2}, & \textit{if } ac_1.x < ac_2.x, \\ \frac{R_1.ul.x+R_2.ul.x}{2}, & \textit{if } ac_1.x \geq ac_2.x. \end{cases} \quad (1)$$

$$p_{ac_1,ac_2}.y = \begin{cases} \frac{R_1.ul.y+R_2.ul.y}{2}, & \textit{if } ac_1.y < ac_2.y, \\ \frac{R_1.ll.y+R_2.ll.y}{2}, & \textit{if } ac_1.y \geq ac_2.y. \end{cases} \quad (2)$$

The above concepts are used to divide $\mathbb{P}$ into half planes, which will be used for influencing region computation. Specifically, the perpendicular bisector of ac_1 and ac_2 divides $\mathbb{P}$ into two half planes. Let P_{ac_1} and P_{ac_2} be the two half planes that contains ac_1 and ac_2, respectively. The normalized perpendicular bisector of ac_1 and ac_2 also divides the plane into two *normalized half planes*. We denote the one corresponding to P_{ac_1} as NP_{ac_1}, and the one corresponding to P_{ac_2} as NP_{ac_2}. The four pairs of antipodal corners of R_1 and R_2 have four normalized perpendicular bisectors and thus four pairs of half planes. We denote these four pairs of half planes as (NP_{11}, NP_{21}), (NP_{12}, NP_{22}), (NP_{13}, NP_{23}) and (NP_{14}, NP_{24}). Then, assuming that R_1 is the MBR of a node N_C in tr_C, R_2 is the MBR of a node N_F in tr_F, the following theorem proofed by Cheema et al. [3] guarantees that any point in $\bigcap_{i\in[1,4]} NP_{2i}$ is not influenced by N_C.

Theorem 3 *Given two rectangles R_1 and R_2, let p be a point in $\bigcap_{i\in[1,4]} NP_{2i}$, where NP_{2i} denotes a normalized half plane corresponding to R_2. Then the minimum distance between p and a point in R_1 must be larger than the maximum distance between p and a point in R_2.*

We call the region $\bigcap_{i\in[1,4]} NP_{2i}$ the customer pruning region of N_C defined by N_F and denote it as $CPR_{N_F}(N_C)$. Given a set of nodes S_F in tr_F, N_C's customer pruning region defined by S_F, $CPR_{S_F}(N_C)$, is defined as $\bigcup_{N_{Fi}\in S_F} CPR_{N_{Fi}}(N_C)$.

Suppose S_{tr_F} contains all the nodes in tr_F, then $\mathbb{P} \setminus CPR_{S_{tr_F}}(N_C)$ is the smallest region defined by tr_F that bounds all the customers who are possibly influenced by N_C, which is an influencing region of N_C. However, to compute this region requires a complete traversal on tr_F. If we do this for every node N_C in tr_C, then the computational cost will be too high. Therefore, we use the following approach to compute

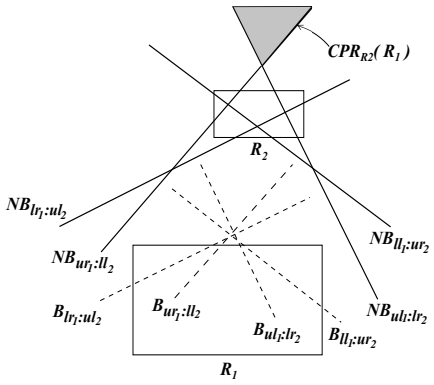


Figure 6: Normalized half planes

approximate influencing regions for the nodes in tr_C recursively from the root node to the leaf nodes during we perform the best first traversal on tr_C to compute the query answer set. We call the nodes used to compute the approximate influencing region of a node N_C the relevant F nodes of N_C, and the nodes of tr_M the relevant M nodes of N_C.

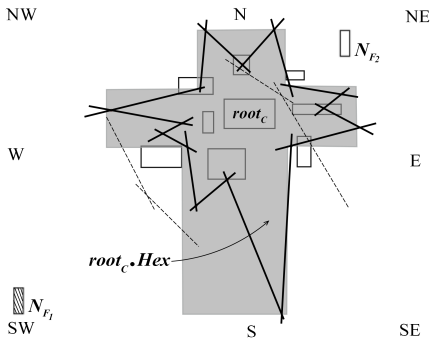


Figure 7: $root_C$'s initial influencing region

The traversal on tr_C starts with the root node $root_C$ of tr_C. For $root_C$, we compute a node in tr_F that is nearest to $root_C$ while does not intersects $root_C$ in each of the eight directions, i.e., western (W), southwestern (SW), southern (S), southeastern (SE), eastern (E), northeastern (NE), northern (N), and northwestern (NW)[2]. These eight nodes of tr_F form an initial set of relevant F nodes of $root_C$, $root_C.R_F$. They generate an initial influence region of $root_C$, which is a hexadecagon (a 16-sided polygon) that encloses $root_C$, as guaranteed by the axiom system of Euclidean geometry (c.f. Figure 7). We denote this polygon as $root_C.Hex$. After $root_C.Hex$ is computed, we traverse tr_F. For a node N_F in tr_F, if this node intersects $root_C$, then we add it

[2]To simplify the discussion, we assume these eight nodes can always be found. In fact, even if only some of them exist, we can still compute the approximate influencing region.

to $root_C.R_F$ for child nodes' approximate influence region computation since it can still be disjoint to $root_C$'s child nodes, and stops traversing its child nodes. Otherwise, we (i) compute the influence region defined by N_F, $\mathbb{P} \setminus CPR_{N_F}(root_C)$, (ii) update $root_C.Hex$ and add N_F to $root_C.R_F$ if this region reduces $root_C.Hex$, and (iii) stop traversing N_F's child nodes. After the traversal on tr_F has stop, $root_C.Hex$ is an approximate influence region of $root_C$, $root_C.Ar$.

After the traversal is initiated, once a node N_C's approximate influencing region is computed and $N_C.R_F$ is found, we compute the approximate influencing regions for N_C's child nodes using a similar approach to that of computing the approximate influencing region of $root_C$. The only difference is that, now we can use only the nodes in $N_C.R_F$ to compute approximate influencing regions for the child nodes, instead of traversing tr_F repeatedly. This process terminates when we reach the leaf nodes of tr_C.

During the approximate influencing region computation, we simultaneously compute the relevant M nodes for the nodes in tr_C as follows. Initially, the relevant M nodes of $root_C$ is set to $\{root_M\}$, where $root_M$ denotes the root node of tr_M. Then, Every time we have computed the approximate influencing region of a child node N'_C of N_C, for every node N_M in $N_C.R_M$, we check which child nodes of N_M intersects $N'_C.Ar$, add those to $N'_C.R_F$, and compute an influence upper bound for N'_C as $\sum_{N_M \in N_C.R_F} \sum_{N'_M \in N_M, N'_M \text{ intersects } N'_C.Ar} |N'_M|$. This value determines the accessing order of the nodes in tr_M.

5.2 Exact Influence Value Computation

When the BIP algorithm reaches a leaf node N_C of tr_C, for every candidate location c indexed in N_C, BIP derives the relevant F nodes and compute an influence region, uses this influence region to prune the relevant M nodes in $N_C.R_M$, and then computes the exact influence value of c.

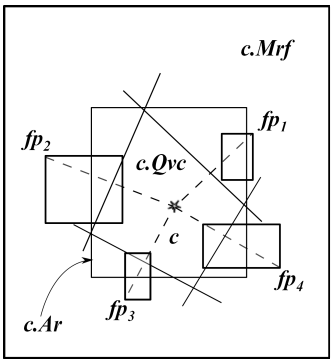


Figure 8: $c.Qvc$, $c.Ar$ and $c.Mrf$

The influence region computation for a candidate location c uses a similar idea as that of a node N_C in tr_C, but it employs a different technique called *quasi-Voronoi cell* (c.f. Figure 8) as follows. In the coordinate system with the origin at c and the two axes parallel with the original axes, we find the nearest F node of c in each

of the four quadrants. Here, we say a node N_{F1} is nearer to c than node N_{F2} if $MaxDist(c, N_{F1}) < MaxDist(c, N_{F2})$. In each of the MBRs of the four nodes, we find the point whose distance to c equals the MaxDist between c and the corresponding MBR. We denote the resultant points as fp_1, fp_2, fp_3 and fp_4. Draw the bisector between each fp_i $(i = 1, 2, 3, 4)$ and c, and the four bisectors form a polygon. This polygon is the quasi-Voronoi cell of c, denoted as $c.Qvc$. Stanoi et al. [15] proofed that $c.Qvc$ encloses the influenced customers of c. We use the MBR of $c.Qvc$ as c's approximate influencing region, and denote it as $c.Ar$. All the nodes in $N_C.R_M$ that intersect $c.Ar$ form $c.R_M$. From Stanoi et al.'s proof [15], we can also derive that, by doubling the size of $c.Ar$, we will have an MBR that bounds all the relevant F nodes of c. We denotes this MBR as $c.Mrf$ and put every node in $N_C.R_F$ that intersect $c.Mrf$ into $c.R_F$.

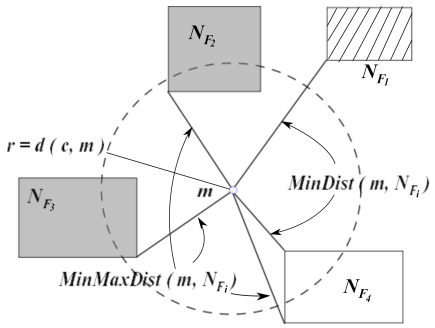


Figure 9: Exact influence value computation

Once we have $c.R_M$ and $c.R_F$, which are fairly small sets after being filtered by all the pruning techniques described above, we compute the influence value of c. For each $m \in c.R_M$, (i) if there is no node N_F in $c.R_F$ with $MinDist(m, N_F) < d(c, m)$, then we increase the influence value of c, (ii) if there is one node N_F in $c.R_F$ with $MinMaxDist(m, N_F) < d(c, m)$, then m is ignored by c, (iii) if there is a node N_F in $c.R_F$ with $MinMaxDist(m, N_F) > d(c, m) \geq MinDist(m, N_F)$, then we check every existing facility in N_F to see if there is one $f \in N_F$ such that $d(c, m) > d(f.m)$ and if yes, we discard m for c. After we have checked all the nodes that satisfy case (iii), if we still cannot find an existing facility with $d(c, m) > d(f.m)$, then we increase the influence value of c. (c.f. Figure 9)

5.3 Complexity Analysis

We analyze the complexity of BIP in this subsection. When computing exact influence, let f_1, f_2, c_1, m_1 denote average amount of checked N_F, computed f, computed c, and computed m, respectively, then the procedure costs $\mathcal{O}(c_1 \cdot m_1 \cdot (f_1 + f_2))$. In the pruning procedure, let c_2, f_3, m_2, h denote the average amount of N_C needed to be computed on influence bound, corresponding $N_C.R_F$, corresponding $N_C.R_M$, and the average height of max-heap. Then finding 8 nearest N_F is of $\mathcal{O}(\log(f_3))$, constructing $N_C.Hex$ and obtaining $N_C.R_M$ is of $\mathcal{O}(1)$, pruning F to obtain $N_C.R_F$ is of $\mathcal{O}(f_3)$, re-insertion into the heap is of $\mathcal{O}(h)$. Thus, the overall cost for pruning is

$\mathcal{O}(c_2 \cdot (\log f_3 + f_3 + h))$. Since $\mathcal{O}(\log f_3) \ll \mathcal{O}(f_3)$ and in most situation, $\mathcal{O}(h) = \mathcal{O}(\log |C|) \ll \mathcal{O}(f_3)$, the cost is $\mathcal{O}(c_2 \cdot f_3)$. To summarize, the complexity of BIP is $\mathcal{O}(c_1 \cdot m_1 \cdot (f_1 + f_2) + c_2 \cdot f_3)$.

- In the best case, where the pruning procedure is smooth, namely $c_1 \ll c_2, m_1 \ll |M|$, the cost of BIP is $\mathcal{O}(c_2 \cdot f_3)$. Since in best situation, $\mathcal{O}(c_2) = \mathcal{O}(\log |C|)$, and $\mathcal{O}(f_3) = \mathcal{O}(\log |F|)$, the complexity is $\mathcal{O}(\log |C| \cdot \log |F|)$.
- In the worst scenario, the pruning procedure prunes little M, F, or C, thus $\mathcal{O}(c_2) = \mathcal{O}(|C|) \gg \mathcal{O}(c_1)$, the overall cost of BIP is $\mathcal{O}(|C| \cdot m_1 \cdot (f_1 + f_2)) = \mathcal{O}(|M| \cdot |F| \cdot |C|)$.
- In the average cases, the pruning procedure prunes part of C. Let α and β denote the average pruning ratio on f and m needed in computing exact influence for each c, then $\mathcal{O}(m_1) = \mathcal{O}((1-\beta) \cdot |M|)$ and $\mathcal{O}(f_1 + f_2) = \mathcal{O}((1-\alpha) \cdot |F|)$. Thus the overall cost should be $\mathcal{O}(\log |C| \cdot \log |F| + (1-\alpha) \cdot (1-\beta) \cdot |M| \cdot |F| \cdot |C|)$. Generally α and β are closed to 1.

Though the theoretic performance of BIP drifts between $\mathcal{O}(\log |F| \cdot \log |C|)$ and $\mathcal{O}(|M| \cdot |F| \cdot |C|)$, we will use experiments to confirm that in most situations, BIP constantly outperforms naive algorithm, proving the accuracy of our analysis on the average cases of BIP. Moreover, in the best and average cases, the performance of BIP relates little to the size of dataset M, which is highly desired in real applications since the number of customers tend to be massive while that of facilities is relatively small.

6 Performance Study

In this section, we report the results of our performance study. To evaluate the performance of the proposed algorithms under different environments, we conduct experiments on both synthetic and real datasets. The experimental settings are detailed in Section 6.1. From Section 6.2, we present the experimental results. Specifically, Section 6.2 presents experiments where we vary the tree node size to find a best value for our algorithm implementation. Section 6.3 presents experiments where we vary the cardinalities of the datasets. Section 6.4 presents experiments where we vary the value of k in the top-k query.

6.1 Experimental Setup

All experiments were conducted on a personal computer with a 2GHz CPU and 2GB RAM. We implement the algorithms with C++. R-trees and aR-trees are used to index the datasets. With practical cardinalities of the three datasets M, F and C, the total data size is less than 100MB. Given the current computer memory size, it is reasonable to assume that all the datasets reside in the memory and our performance measurement focuses on the total response time of the algorithms. To help understand the computation of the algorithms, we also measure the number of distance metric computations performed by the algorithms, which is a good indicator of the pruning power of the algorithms.

We conduct experiments on both synthetic and real datasets. Synthetic datasets are generated in a space domain of 1000×1000. The dataset cardinalities range from 1,000 to 2,000,000. To simulate real-life scenarios, where residential distribution and facility

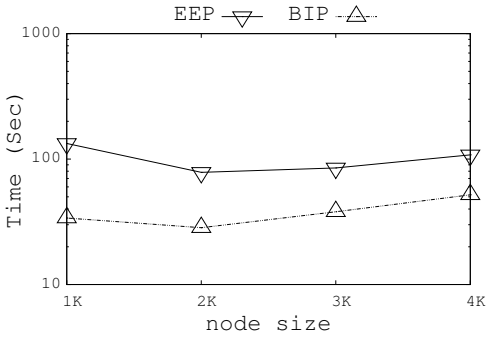


Figure 10: Effect of node size

distribution are skewed while candidate location distribution are relatively uniform, we generate the set of customers M and the set of existing facilities F with Zipfian distribution with $\alpha = 1.2$, and the set of candidate locations C with uniform distribution. We call the resultant datasets the **"ZIPF"** datasets. To verify the effect of the value of k, we use values ranging from 10 to 5,000. As previous studies [14, 9, 20] on main memory databases show, the tree node size of main memory index has a significant impact on the index performance. Therefore, we vary the node size ranging from 1K to 4K to study the effect of node size.

Table 2: Experiment Parameters

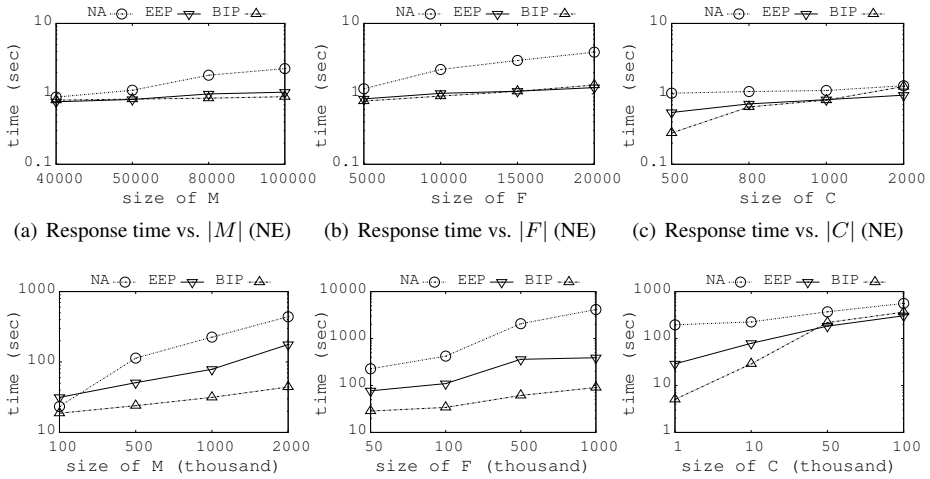
Parameters	Synthetic Data	Real Data
$\|M\|$	100K, 500K, **1M**, 2M	40K, **50K**, 80K, 100K
$\|F\|$	**50K**, 100K, 500K, 1M	**5K**, 10K, 15K, 20K
$\|C\|$	1K, **10K**, 50K, 100K	500, 800, **1K**, 2K
k	**10**, 50, 500, 5K	**10**
Node Size	1K, **2K**, 3K, 4K	**2K**
α	**1.2**	N/A

The real dataset we use is the North East dataset from R-tree Portal [13]. The North East dataset contains 123,593 postal addresses, which represent three metropolitan areas of the USA (New York, Philadelphia and Boston). We uniformly sample from this dataset to generate M, F and C, with dataset cardinalities ranging from 500 to 100,000. The resultant datasets are called the **"NE"** datasets. We set k to 10 and the node size to 2K in these experiments.

The parameters used in the experiments summarized in Table 2, where values in bold denote default values used.

6.2 Effect of Node Size

This subsection studies the effect of tree node size on the performance of the proposed techniques. Results in Figure 10 show that the two algorithms' performance varies with different node sizes. Specifically, when increasing the node size, the response time of



(a) Response time vs. $|M|$ (NE) (b) Response time vs. $|F|$ (NE) (c) Response time vs. $|C|$ (NE)

(d) Response time vs. $|M|$ (ZIPF) (e) Response time vs. $|F|$ (ZIPF) (f) Response time vs. $|C|$ (ZIPF)

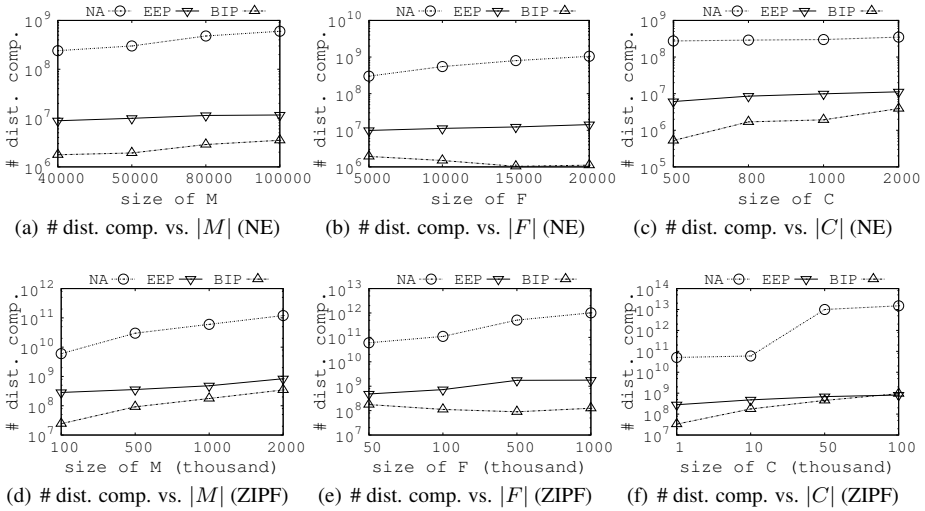
Figure 11: Effect of dataset cardinality on the total response time

both algorithms first decreases and then increases. Note that for both algorithms, their pruning techniques relies on the geometric relationship among the MBRs of the nodes in tr_M, tr_F and tr_C. The node size affects the size of these MBRs and further affects the pruning power. Larger node size leads to larger MBRs and therefore more data objects can be pruned at the same time. However, larger node size also results in smaller tree heights, and thus weaker pruning power. Therefore, when the node size becomes too large, both algorithms' performance get worse, and as the experimental result shows, both algorithms prefer the node size of 2K. Hence, in the following experiments, we use node size of 2K to implement the algorithms.

6.3 Effect of Dataset Cardinality

In this subsection, we vary the cardinalities of the set of customers, the set of existing facilities and the set of candidate locations to verify the efficiency of the proposed algorithms. When varying the cardinality of one dataset, the cardinalities of other datasets are set to default values. Figures 11 and 12 give the experimental results of the total response time and the number of distance metric computations, respectively.

Effect of varying $|M|$. First, we vary the cardinality of the set of customers. Figure 11(a) and Figure 12(a) show the total response time and the number of distance metric computations respectively for the experiments conducted on the "NE" datasets, and Figure 11(d) and Figure 12(d) show those for the "ZIPF" datasets. We can see from these figures that both the EEP algorithm and the BIP algorithm significantly outperform the NA algorithm in terms of both the total response time and the number of distance metric computations. Also, with the increase in the customer set cardinality, EEP and BIP keep relatively stable performance, while NA's performance deteriorates drastically (please note the logarithmic scale in Figure 12(a) and 12(d)). The reason is that NA sequentially scans all the datasets, which results in large number of distance



(a) # dist. comp. vs. $|M|$ (NE) (b) # dist. comp. vs. $|F|$ (NE) (c) # dist. comp. vs. $|C|$ (NE)

(d) # dist. comp. vs. $|M|$ (ZIPF) (e) # dist. comp. vs. $|F|$ (ZIPF) (f) # dist. comp. vs. $|C|$ (ZIPF)

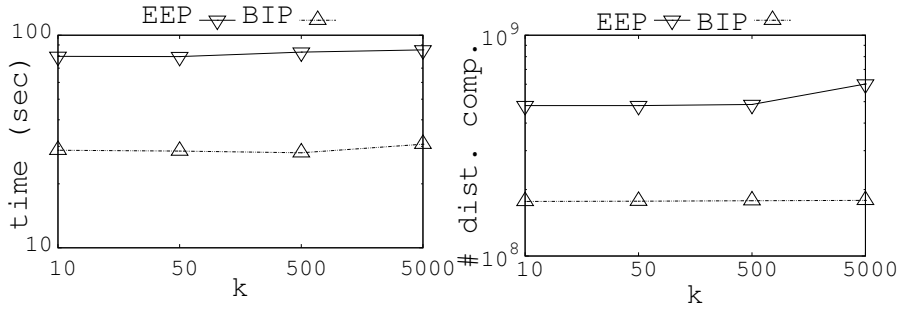
Figure 12: Effect of dataset cardinality on the number of distance metric computations

computations. The dataset cardinality directly affect the number of distance computations. In contrast, EEP and BIP use pruning techniques to keep relatively small search spaces and thus achieve much better performance.

Effect of varying $|F|$. Next, we vary the cardinality of the set of existing facilities. The experimental results on the total response time and the number of distance metric computations for the "NE" datasets and the "ZIPF" datasets are shown in Figure 11(b), 12(b), 11(e) and 12(e). Again, these figures show that EEP and BIP have much better performance than that of NA, the reason being EEP and BIP's much smaller search spaces achieved by the proposed pruning techniques. An interesting phenomenon that can be observed from these figures is that with the increase in the number of existing facilities, the number of distance metric computations of BIP becomes smaller. This is because BIP uses the set of F to prune some of the nodes in tr_F as well as tr_M. Specifically, more existing facilities means more nodes in tr_F and thus smaller distances between a node N_C in tr_C and its nearest nodes in tr_F. Hence, $N_C.Hex$ becomes smaller. Therefore, larger search space can be pruned and BIP achieves better performance.

Effect of varying $|C|$. Finally, we vary the cardinality of the set of candidate locations and the results are shown in Figures 11(c), 11(f), 12(c) and 12(f). All these figures confirm the superiority of EEP and BIP over NA. We observe that compared with varying the cardinality of M or F, varying the cardinality of C affects the performance of BIP more significantly. This is because BIP's pruning power on C relies on the relative size of C over F. For BIP's pruning techniques to work better, it requires a smaller number of $|C| : |F|$. Therefore, given a fixed F, when the cardinality of C increases, BIP's performance decreases faster than that of EEP. However, BIP's performance is still better than that of NA.

From the experiments above, we can see that both EEP and BIP outperform NA in



(a) Response time vs. k (b) # dist. comp. vs. k

Figure 13: Effect of k

terms of both the total response time and the number of distance metric computations because of the pruning techniques used to reduce the search space.

6.4 Effect of the Value of k

Due to NA's inefficiency, in the following experiments, we only show the results of EEP and BIP. In this subsection, we examine the performance of the proposed algorithms under different values of k in the top-k most influential location selection query. As the results in Figure 13(a) and 13(b) show, the total response time and the number of distance metric computations stay stable for both algorithms. This is expected because the proposed pruning techniques focus on reducing the search space when computing the influence values. The algorithms still need to compute the influence values of more than k candidate locations until they can early terminate and report the top-k query answer set. Therefore, given a group of datasets, the algorithms keep a relatively stable performance when we vary the value of k.

7 Conclusions

We formulated the top-k most influential location selection query and conducted a comprehensive study on processing this query. We first analyzed the basic properties of this query type and proposed a naive algorithm (NA) to process the query. However, the NA algorithm is inefficient because it needs to scan all the datasets repeatedly. Motivated by this, we explored geometric properties of spatial data objects, and proposed techniques to prune the search space. This resulted in two algorithms, the EEP algorithm and the BIP algorithm. The EEP algorithm estimates the distances to the nearest existing facilities for the customers and the numbers of influenced customers for the candidate locations, and then use distance metric based pruning techniques to gradually refine the estimation until the answer set is found. The BIP algorithm only estimates the numbers of influenced customers for the candidate locations, but it uses the existing facilities to reduce the search space, and thus achieves an even more efficient algorithm. Experimental results show that the proposed pruning techniques are effective and the proposed algorithms outperform the naive algorithm significantly (up to 10 times). In most cases, BIP performs better than EEP.

References

[1] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 886–897, New York, NY, USA, 2009. ACM.

[2] S. Cabello, J. M. D, S. Langerman, and C. Seara. Reverse Facility Location Problems. In *Proc. of CCCG*, 2006.

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

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

[5] Y. Chen and J. M. Patel. Efficient evaluation of all-nearest-neighbor queries. In *ICDE*, pages 1056–1065. IEEE, 2007.

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

[7] Y. Gao, B. Zheng, G. Chen, and Q. Li. Optimal-Location-Selection Query Processing in Spatial Databases. *IEEE TKDE*, 21(8):1162–1177, Aug. 2009.

[8] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proc. of SIGMOD*, pages 47–57, 1984.

[9] R. A. Hankins and J. M. Patel. Effect of node size on the performance of cache-conscious b+-trees. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pages 283–294, New York, NY, USA, 2003. ACM.

[10] F. Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. *ACM SIGMOD Record*, 29(2):201–212, June 2000.

[11] K. Mouratidis, D. Papadias, and S. Papadimitriou. Medoid Queries in Large Spatial Databases. In *Proc. of SSTD*, pages 55–72, 2005.

[12] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. *Advances in Spatial and Temporal Databases*, 2121:443–459, 2001.

[13] R.-T. Portal. http://www.rtreeportal.org/.

[14] J. Rao and K. A. Ross. Cache conscious indexing for decision-support in main memory. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 78–89, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[15] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of Influence Sets in Frequently Updated Database. In *Proc. of VLDB*, 2001.

[16] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proc. VLDB Endow.*, 2:1126–1137, August 2009.

[17] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On Computing Top-t Most Influential Spatial Sites. In *Proc. of VLDB*, 2005.

[18] C. Yang and K.-i. Lin. An index structure for efficient reverse nearest neighbor queries. In *Proc. of ICDE*, pages 485–492. IEEE Comput. Soc, 2001.

[19] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, 2006.

[20] R. Zhang and M. Stradling. The hv-tree: a memory hierarchy aware version index. *Proc. VLDB Endow.*, 3:397–408, September 2010.