Location Selection for Utility Maximization with Capacity Constraints

Yu Sun[1], Jin Huang[2], Yueguo Chen[3], Rui Zhang[2], Xiaoyong Du[1,3]
[1]School of Information, Renmin University of China, Beijing, China
[2]University of Melbourne, Melbourne, Australia
[3]Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), MOE, China
yusun.aldrich@gmail.com, jin.h@iojin.com, chenyueguo@ruc.edu.cn
rui@csse.unimelb.edu.au, duyong@ruc.edu.cn

ABSTRACT

Given a set of client locations, a set of facility locations where each facility has a service capacity, and the assumptions that: (i) a client seeks service from its nearest facility; (ii) a facility provides service to clients in the order of their proximity, we study the problem of selecting all possible locations such that setting up a new facility with a given capacity at these locations will maximize the number of served clients. This problem has wide applications in practice, such as setting up new distribution centers for online sales business and building additional base stations for mobile subscribers. We formulate the problem as ***location selection query for utility maximization***. After applying three pruning rules to a baseline solution,we obtain an efficient algorithm to answer the query. Extensive experiments confirm the efficiency of our proposed algorithm.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications–*Spatial databases and GIS*

General Terms

Algorithms, Performance, Theory

Keywords

Location Selection, Capacity Constraints

1. INTRODUCTION

A common problem in operation management is to select optimal locations to build facilities. The choice is based on factors such as the gain on profits and improvement on service quality. Let us consider an example in online sales business. To ensure quick shipment, customers are served by their nearest distribution centers. In Fig.1 rectangles represent the distribution centers. Every distribution center has a capacity that indicates the number of customers it can serve. When the number of customers exceeds its capacity, the distribution center only serves the customers in the order of their proximity. Hence, there may be unserved customers. In the figure, the capacities of distribution centers f_1, f_2, and f_3 are 3, 4, and 5, respectively. Circles and black dots indicate the served and unserved customers, respectively. The customer c_4 is unserved since its nearest distribution center f_1 only has a capacity of 3, which has been used up by c_1, c_2, c_3 since they are nearer to f_1 than c_4. Assume that we want to build a new distribution center with a capacity of 4. Setting it up in any region in the set $\{R_1, R_2, R_3, R_4, R_5\}$ will serve 4 more customers, which reaches its full capacity. This is the highest utility that can be achieved. Therefore, $\{R_1, R_2, R_3, R_4, R_5\}$ is the answer to this query.

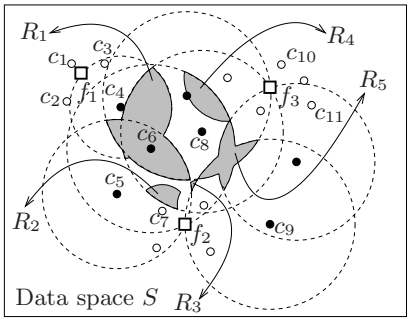


Figure 1: Query example

We assume that a facility only serves clients who perceive it as their nearest facility and it serves clients to its capacity in an ascending order of the distances to them.

DEFINITION 1. *In a 2-dimensional data space S, let F denote the facility set containing locations of existing facilities f, C denote the client set containing locations of clients c, $v(f)$ denote the capacity of a facility f, and $SE(C, F)$ denote the number of served clients. Given the capacity $v(f_n)$ of a new facility f_n, the* ***location selection query for utility maximization*** *returns a set M of regions which contains all regions such that if f_n is set up in any of them, the number of newly served clients (utility of the new facility), denoted as $u(R) = SE(C, F \cup \{f_n\}) - SE(C, F)$, is maximized.*

Practically, a client may require multiple units of service, which can be considered as the weight of the client. In this general setting, we still follow the serving order rule. The f_n may contribute to increasing $u(R)$ in two ways: (i) to

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

directly serve unserved clients; (ii) to share the balance of overloaded facilities by serving already served clients, so that they have the capacity to serve unserved clients. In Fig.1, if f_n is set up in R_1 or R_5, it directly serves 4 unserved clients. If f_n is located in R_2, c_7 is taken over by f_n, then f_2 is able to serve unserved c_9. Meanwhile, f_n directly serves unserved c_5, c_6 and c_8. The number of newly served clients is also 4. Similar cases happen when f_n is located in R_3 or R_4. The optimization goal is global, considering only unserved clients is not enough in solving our problem.

In this paper, we make following contributions: (i) A new type of query, named ***location selection query for utility maximization***, is proposed; (ii) *Arc* based methods are introduced to describe and validate regions; (iii) An efficient algorithm to answer the query is obtained via applying three pruning rules to a straightforward solution; (iv) Extensive experiments are conducted. The results confirm the efficiency of the proposed algorithm.

2. RELATED WORK

Our problem is based on **b**ichromatic **r**everse **n**earest **n**eighbor (BRNN) query introduced by Korn et al. [3]. For the situation where database is updated frequently, a two-stage approximate Voronoi diagram algorithm is proposed to efficiently answer the query [5]. Recent progress on improving the efficiency of computing RNN can be found in many studies [12, 1].

Some distance based location selection queries aim at optimizing distance functions. Minimizing the average distance between clients and facilities by locating a new facility is studied [14, 4]. Finding the optimal matching between clients and facilities under capacity constraints is also tackled [11, 8, 7]. All above studies concentrate on optimizing distance functions or matching applications, thus the solutions are inapplicable to our problem. Some influence based queries define the number of RNN of a facility as its influence. Xia et al.[13] find locations with maximal influence in a given query region. Wong et al.[10] extend similar work to L_p-norm space of two and three dimensionality [9]. Huang et al.[2] find those locations among a candidate set. However, no capacity constraints are considered.

3. MAXIMAL REGIONS AND BRNN SETS

Assume that every client has only one nearest facility [6, 15]. Given a client c, the nearest facility circle (NFC) [3] of c, denoted as $n(c)$, is a circle that centers at c and has a radius of $dist(c, f)$, where f is the nearest facility of c and $dist(c, f)$ is the Euclidean distance between c and f. We denote the set of all the points within $n(c)$ as $\bar{c}$ and the set of all the points outside as $\hat{c}$. The points right on $n(c)$ cannot be answers to the query on the above assumption.

DEFINITION 2 (CONSISTENT REGION). *A consistent region R is a set of points satisfying that: 1) $\forall c \in C, \nexists p_i, p_j \in R$ such that $p_i \in \bar{N}(c)$ and $p_j \in \hat{N}(c)$; 2) $\forall c \in C, n(c) \cap R = \emptyset$.*

DEFINITION 3 (MAXIMAL REGION). *A consistent region R is a maximal region if $\forall R' \cap R \neq \emptyset$, then $R' \subseteq R$.*

For example, in Fig.1, the set of all points in R_4 is a consistent region, while set $\{c_{10}, c_{11}\}$ is not. According to the definition, two different maximal regions cannot *overlap*. Because once they overlap, they share at least one point, thus they are subsets of each other. They are therefore the same region. Fig.2(b) shows an example of four maximal regions $\bar{c_1} \cap \bar{c_2}$, $\bar{c_1} \cap \hat{c_2}$, $\hat{c_1} \cap \bar{c_2}$ and $\hat{c_1} \cap \hat{c_2}$ determined by two overlapped NFCs. Note that a maximal region does not have to be a connected region. Fig.2(c) shows an example.

Let $B(f)$ denote the BRNN set of a facility f, which contains the clients whose nearest facility is f. Similarly, $B(p)$ denotes the BRNN set of a point p. The following two lemmas show the one-to-one relationship between a maximal region and a BRNN set.

LEMMA 1. *$\forall p_i, p_j$ in a region R, we have $B(p_i) = B(p_j)$.*

LEMMA 2. *$\forall p_i, p_j$, if $B(p_i) = B(p_j)$, then p_i, p_j must be in the same maximal region.*

Thus a maximal region can be represented by a BRNN set. Points in the corresponding maximal region of a BRNN set must satisfy: (1) lying within the NFCs of all clients in the BRNN set; and (2) locating outside of the NFC of any other client not in the set. Actually, the data space is partitioned into many maximal regions by the NFCs of all clients. Hence, we are able to describe a maximal region by using $\bar{c}$ or $\hat{c}$. Formally, given a BRNN set $B(p)$, the corresponding maximal region is $R = (\bigcap_{c_i \in B(p)} \bar{c_i}) \bigcap (\bigcap_{c_j \in \bar{B}(p)} \hat{c_j})$. So it is possible to get *all* maximal regions through all combinations of clients. For $|C|$ clients in C, there are totally $2^{|C|}$ possible BRNN sets can be enumerated. Some enumerations may have no corresponding maximal regions (or in other words the corresponding regions are empty sets). For example, in Fig.2(c), the region $\hat{c_1}\bar{c_2}\bar{c_3}$ for the enumerated BRNN set $\{c_2, c_3\}$ actually does not exist. Techniques on checking the validation of an enumerated BRNN set will be discussed in the next section.

When adding a new facility f_n with a given capacity in a region, we can get the total weight of newly served clients according to the corresponding $B(f_n)$ [6]. This total weight is the facility utility. So to get the highest utility, we can use maximal regions as basic units to search the whole space.

4. VALIDATING AN ENUMERATION

A maximal region is *enclosed* by arcs generated from some NFCs intersecting with each other. Given an enumerated BRNN set, we need to validate it and find its corresponding maximal region. In our problem the containment relationship between NFCs is impossible [10], only the intersection and separation relationship need to be considered.

We adopt an incremental way of checking. During the validating process, a temporary maximal region R is incrementally maintained. When a new client c is involved for checking, based on the geographic relationship between $n(c)$ and R, the temporary maximal region is updated. Checking one client after another, if R becomes an empty set, then the enumeration is invalid. However, if R is not empty after the checking process, we are sure that it is valid, and the current temporary maximal region R is its corresponding maximal region.

When checking a new client c, we need to update the arcs enclosing R. Given an NFC, a point has three states in terms of their positional relationship: *in*, *on*, and *out*. According to the states of the two end points of an arc, to a given NFC, there are 12 cases of the location relationship between them. We show all the 12 cases in Fig.2(a). For all

the cases, we update those enclosing arcs accordingly. If it is $\bar{c}$, then we keep the inside part of the arc, else the outside part is kept. In addition, new enclosing arcs are generated when $n(c)$ intersects with R. The new NFC is partitioned into a number of arcs by the intersect points between $n(c)$ and the existing arcs. For each arc, we only need to check its middle point. If its middle point is in R, it is a new enclosing arc. Otherwise it is not. When finishing the updating and insertion of arcs, the temporary maximal region becomes a new one.

We use an example in Fig.2(b)(c) to illustrate the validating process. In this example, we need to validate $\bar{c_1}\hat{c_2}\hat{c_3}$. Let the temporary maximal region be $R = \bar{c_1}\hat{c_2}$, whose enclosing arcs are a_1 and a_2. When checking c_3, both a_1 and a_2 are updated according to case 12, leading to 4 smaller arcs remained. Finally, two new arcs a_3 and a_4 from the new NFC are added, which causes the two shadowed regions to be the maximal region of $\bar{c_1}\hat{c_2}\hat{c_3}$.

5. SEARCH MAXIMAL UTILITY REGIONS

5.1 Baseline Algorithm

A straightforward solution is to enumerate all combinations of clients, validate each enumeration, compute the increment of its corresponding BRNN set [6] and get the utility. Since a meaningful result serves at least one client, we can therefore get all possible cases by enumerating maximal regions within one NFC, then another. To enumerate within one NFC, we only need to consider those NFCs that intersect with it. To avoid repetitions, we assign each NFC an unique ID so that they can be scanned in order.

An example is shown in Fig.2(d). There are six NFCs and we assign each an unique ID. We start from the NFC with the smallest ID 1. Since 1 only intersects with 2 and 3. Then, all BRNN sets we will enumerate for the NFC 1 are $12\hat{3}\hat{4}\hat{5}\hat{6}, 12\hat{3}\hat{4}\hat{5}\hat{6}, 1\hat{2}3\hat{4}\hat{5}\hat{6}$ and $1\hat{2}\hat{3}\hat{4}\hat{5}\hat{6}$. Then, we turn to check 2, it intersects with $1, 3, 4$, since 1 is smaller than 2, thus we will only enumerate $\hat{1}234\hat{5}\hat{6}, \hat{1}2\hat{3}\hat{4}\hat{5}\hat{6}, \hat{1}2\hat{3}4\hat{5}\hat{6}$ and $\hat{1}2\hat{3}\hat{4}5\hat{6}$. In this way, the maximal region that lies in both 1 and 2, i.e. $12\hat{3}\hat{4}\hat{5}\hat{6}$, will not be repeated.

5.2 Advanced Pruning Algorithm

Though the baseline algorithm can avoid a large percentage of enumerations, it is still not efficient. We propose three pruning rules to speedup the search process.

(I) In each step of the baseline algorithm, the maximal regions within an NFC $n(c)$ are checked. We can treat $n(c)$ as the root of this search step. Given an NFC $n(c)$, we denote the set of clients whose NFC intersects with $n(c)$ and having a larger ID than c as $C_e(c)$. Baseline algorithm enumerates all combinations of clients in $C_e(c)$. However, before enumerating, we can estimate the upper bound utility of all maximal regions within $n(c)$, denoted as $u(c)$, it helps us determine whether the $n(c)$ is worth exploring or not.

For presenting convenience, we give some relevant definitions used here. Given a facility f, its *work load* denoted by $wl(f)$, is the total weight of all clients in its BRNN set. The *max increment* of a facility f, denoted by $inc(f)$ equals to $wl(f) - v(f)$ if $wl(f) > v(f)$; otherwise it is zero. Let $wt(c)$ denote the weight of c. Given a client c and its nearest facility f, the *increment* of c, denoted by $inc(c)$ equals to (1) $wt(c)$, if c is unserved; (2) $min\{wt(c), inc(f)\}$, otherwise.

With all clients in $C_e(c)$, we have $u(c) = min\{v(f_n), inc(c) + \sum_{c_i \in C_e(c)} inc(c_i)\}$, $inc(c) + \sum_{c_i \in C_e(c)} inc(c_i)$ means total weight of newly served clients. This is only an *ideal* case, for it does not consider the serving order and dynamic updates of the increments of existing facilities. With the estimated upper bound, we can effectively prune the search space.

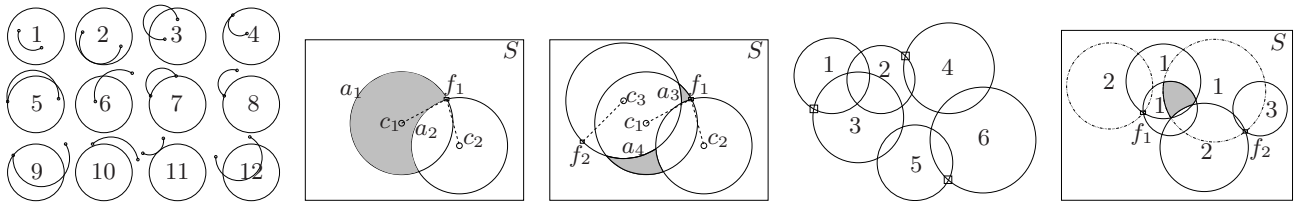
(II) Besides pruning at the branch root, unnecessary search can also be avoided at lower levels. For every f, the best case is the f_n shares exactly $inc(f)$ of its work load. If the input capacity $v(f_n)$ equals to the total weight of unserved clients and there exists a region that can make them all served, then f_n must share $inc(f)$ workload of every existing facility. For example, in Fig.2(e), the number inside each $n(c)$ denotes the weight of c, the capacity of f_1 and f_2 is respectively 2 and 5, and the dashed NFCs denote unserved clients. The shadowed region can make all unserved clients fully served if we add f_n with a capacity $v(f_n) = 10 - 7 = 3$ in it. The shared workload of f_1 and f_2 is exactly 2 and 1, respectively.

In this case, any other region that shares more than the max increment of any facility will not be a result. In more general situations, such a region may not exist, but we can record the current highest utility u_h, which is not more than $v(f_n)$. Given the input capacity $v(f_n)$, for any existing facility f, a promising maximal region shares not more than $inc(f) + v(f_n) - u_h$ workload of f. Otherwise, with less than $v(f_n) - (inc(f) + v(f_n) - u_h) = u_h - inc(f)$ capacity remained, even this part can be fully utilized, the final utility is less than $inc(f) + (u_h - inc(f)) = u_h$. To sum up, when enumerating all combinations of clients in $C_e(c)$, we can use the maximal workload that a facility can be shared as an upper bound to prune unpromising branches.

(III) The baseline algorithm checks whether a maximal region exists only when it gets a possible combination of all clients in $C_e(c)$. However, we can check the exitance of the temporary maximal region at a higher level. If the temporary maximal region corresponding to a subset of an enumeration does not exist, then there is no need to enumerate remained clients. Based on the above pruning rules, we propose the our solution in Algorithm 1.

Algorithm 1: Pruning Algorithm

```
   Input: F, C, v(f_n)
   Output: M
1  I to record intersected NFCs, P to record in clients or c̄
2  foreach c ∈ C do
3      c.id ← id, id ← id + 1
4      assign c to its nearest facility f
5      insert n(c) to index structure s
6  foreach c ∈ C do
7      if u(c) >= u_h then
8          P ← ∅ and P.push(c)
9          I ← nfc(c) range query in s
10         foreach c_i ∈ I do
11             if c_i.id < c.id then
12                 set c_i as ĉ_i
13         foreach c_i ∈ C_e(c) do
14             P.push(c_i) and R ← validating(P)
15             if R ≠ ∅ then
16                 if ∀f ∈ F, Σ_{c∈P∩B(f)} wt(c) <
                   inc(f) + v(f_n) − u_h then
17                     recursively enumerate c_{i+1} ∈ C_e(c)
18                     compute u(R) when reach leaf nodes
19                     add R to M
20             P.remove(c_i)
21 M ← M removes R without highest utility
```



(a) Arc NFC relationship (b) $\bar{c_1}\hat{c_2}$ (c) $\bar{c_1}\hat{c_2}\hat{c_3}$ (d) baseline example (e) $v(f_1) = 2, v(f_2) = 5$

Figure 2: Arc NFC relationship and different examples

6. EXPERIMENTAL STUDIES

We implemented all algorithms using C++. All experiments are conducted on a 64-bit Windows machine with an Inter(R) Core(TM)2 1.86GHz CPU and 7.00GB memory. Because it is possible to read all locations into the main memory, we only include the running time as the indicator of performance. Both real world datasets and synthetic datasets are used. Two real world datasets NE and NA are downloaded from http://www.rtreeportal.org/spatial.html. In NE there are 123,593 addresses of north east of USA and in NA there are 24,493 locations of North America. When using NE or NA, we uniformly sample from it to generate C and F. We also generate C and F with Zipfian distribution. The proposed algorithms *take different weights of client points into account*. Yet through the experiments, we find the weight of clients has little impact on the comparison on performance. We therefore set the weights of all clients as one (although in real applications, it may be a large value) in our study. We use kd-tree to answer the nearest neighbor query and R*-tree to answer the range query.

Before conducting experiments, we analyze three datasets specifically on the clients density because the number of intersections determines the search space of our problem. The results are shown in Table 1. For example, in NE, when $|C|/|F|$ is set to 20, in average an NFC will intersect approximately 54 other NFCs, which is the highest among all used datasets. So the number of actual regions produced by NFCs is dramatically smaller than that in the worst case, where each NFC intersects all other NFCs.

Table 1: Average Intersected NFCs

$\|C\|/\|F\|$	1	2	5	10	20
NE	2.78	6.21	12.66	27.49	53.95
NA	1.92	4.57	13.08	22.60	44.16
SYN	2.11	5.85	13.28	24.56	47.57

In the following subsections, we study the performance of the baseline (**basic**), baseline with first (**1-p**), second (**2-p**), third (**3-p**) pruning rule and the pruning algorithm (**all-p**) which features all three pruning rules.

6.1 Effect of $|C|/|F|$

We study the clients density effect by varying $|C|/|F|$ from 1 to 20. Fig.3(a)(b)(c) show the performance of the baseline on datasets with $|C|/|F|$ bigger than 2 is unacceptable and thus omitted. Both 1-p and 2-p have marginally advantages over the baseline, yet they fail to provide competitive performance when $|C|/|F|$ is more than 5. 3-p alone gives acceptable performance as its cost of time grows smoothly towards the growth of $|C|/|F|$, outperforming others by orders of magnitude when the varying ratio becomes larger. The pruning algorithm is always the best one among all algorithms, with marginal advantages over 3-p. Overall, this set of experiments verifies the effectiveness of pruning rules.

6.2 Effect of $|C|$

Then, we study the impact of the cardinality of datasets on the performance of algorithms. Considering that the baseline algorithm does not scale to large $|C|/|F|$, we set $|C|/|F|$ to 2 in following experiments. The input capacity is also set to $|C|/|F|$. The result is shown in Fig.3(d)(e)(f).

From the figures, we observe that on all datasets the results are similar. Baseline algorithm provides the worst performance while 1-p and 2-p perform marginally better. Both 3-p and the pruning algorithm give magnitude better performance especially when datasets grow, with the pruning algorithm outperforming 3-p in almost one order of magnitude. In terms of growth trend, it is also 3-p and the pruning algorithm that give the slowest increase rate. We also find that when the datasets are really small, such as $|C| = 20$, all algorithms with pruning rules might be beaten by the baseline algorithm. This is simply because in this condition, the cost of pruning itself overweights the gain on performance it earns from tightening the search space. Yet in practical applications, such tiny datasets are of less interests.

Note that there are some inflection points in the results, e.g., $|C| = 500$ for Fig.3(f). This is because the distribution of facilities in different samples of a dataset may vary significantly, which may cause the maximal number of NFCs intersecting with an NFC varies in different samples.

6.3 Studies on The Pruning Algorithm

We study the impacts of existing capacities, input capacity and dataset size on the pruning algorithm, respectively. For existing capacities, we uniformly sample 5412 (2194) and 52992 (22299) points from NE (NA) as facilities and clients. We also generate 10K and 100K points as facilities and clients. We adjust the expectation of the existing capacities from 10 to 30, i.e., 1 to 3 times of $|C|/|F|$. As shown in Fig.3(g), the larger the existing capacities, the better performance of the algorithm. This is because larger capacities result in less clients being unserved, which makes the first and second pruning techniques work more effectively.

To investigate the impact of the input capacity $v(f_n)$, We fix the number of facilities and clients at 0.4K and 4K, and vary $v(f_n)$ from 30 to 170. The result in Fig.3(h) shows that when the input capacity is relative small, it costs less time to answer the query. However, when it becomes large enough, the performance almost does not change. The reason is the smaller $v(f_n)$ is, the more effective the second pruning techniques will be, for it uses the input capacity as an indicator to reduce the search space.

Finally, we study the scalability of the pruning algorithm. We use the real-world dataset NE by testing it from a 25% sampling to the original size. We also generate 20K facili-

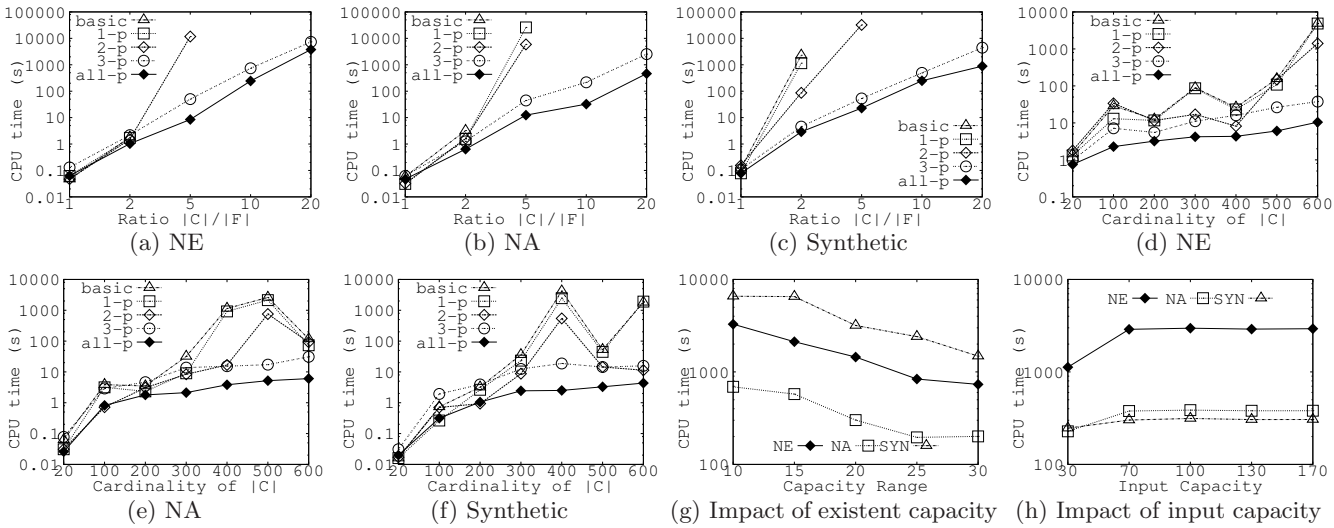


Figure 3: Impact of different factors on the search algorithm(s)

ties and 200K clients as synthetic dataset. Fig.4 shows when the cardinality of datasets climbs rapidly, the cost of computation increases gradually, which suggests the pruning algorithm can be a valid option for solving the proposed location selection query in real-world applications.

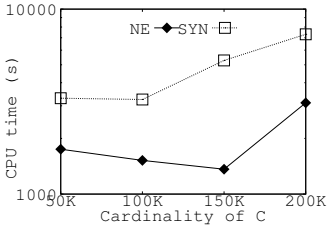


Figure 4: Scalability for large data

7. CONCLUSIONS

Location selection queries have attracted much attention due to their wide applications in real world. Since service-providing facilities are constrained by their capabilities, there may be unserved clients in a booming market. In this paper, we study on selecting all the regions in a data space such that if a new facility of a given capacity is added in any location in these regions, its capacity will be best utilized. Through introducing the concept of maximal regions and the methods of validating them, we proposed a baseline algorithm to solve the problem. To improve its efficiency, we proposed three pruning rules to tighten the search space. With all these three rules, we proposed an efficient pruning algorithm. Extensive experimental results show that the pruning algorithm outperforms the baseline in terms of running time by up to three orders of magnitude and it can be a valid option to answer the proposed query.

Further work may involve proposing worst case bounded approximate algorithms with much less cost of computation. Another interesting track would be considering moving clients instead of assuming that all clients are static.

8. ACKNOWLEDGMENTS

This work is partially supported by NSFC under the grant No. 61003085 and 61170010

9. REFERENCES

[1] M. A. Cheema, W. Zhang, X. Lin, and Y. Zhang. Efficiently processing snapshot and continuous reverse k nearest neighbors queries. *VLDBJ*, 2012.

[2] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top-k most influential locations selection. In *CIKM*, 2011.

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

[4] J. Qi, R. Zhang, L. Kulik, D. Lin, and Y. Xue. The min-dist location selection query. In *ICDE*, 2012.

[5] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, 2001.

[6] Y. Sun, J. Huang, Y. Chen, X. Du, and R. Zhang. Top-k most incremental location selection with capacity constraint. In *WAIM*, 2012.

[7] L. H. U, K. Mouratidis, M. L. Yiu, and N. Mamoulis. Optimal matching between spatial datasets under capacity constraints. *ACM Transactions on Database Systems*, 35(2):1–44, 2010.

[8] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis. Capacity constrained assignment in spatial databases. In *SIGMOD*, 2008.

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

[10] 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. *PVLDB*, 2(1):1126–1137, 2009.

[11] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In *VLDB*, 2007.

[12] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan. Finch: evaluating reverse k-nearest-neighbor queries on location data. *PVLDB*, 1(1):1056–1067, 2008.

[13] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *VLDB*, 2005.

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

[15] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. Maxfirst for maxbrknn. In *ICDE*, 2011.