Top-k Most Incremental Location Selection with Capacity Constraint

Yu Sun[1], Jin Huang[2], Yueguo Chen[3], Xiaoyong Du[1,3], and Rui Zhang[2]

[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,soone.enoos,chenyueguo}@gmail.com,
duyong@ruc.edu.cn, rui@csse.unimelb.edu.au

Abstract. Bichromatic reverse nearest neighbor (BRNN) based query uses the number of reverse nearest customers to model the influence of a facility location. The query has great potential for real life applications and receives considerable attentions from spatial database studies. In real world, facilities are inevitably constrained by designed capacities. When the needs of service increase, facilities in those booming areas may suffer from overloading. In this paper, we study a new kind of BRNN related query. It aims at finding most promising candidate locations to increase the overall service quality. To efficiently answer the query, we propose an $O(n \log n)$ algorithm using pruning techniques and spatial indices. To evaluate the efficiency of proposed algorithm, we conduct extensive experiments on both real and synthetic datasets. The results show our algorithm has superior performance over the basic solution.

Key words: capacity constraint, location selection, BRNN, spatial database

1 Introduction

A common problem many companies are facing is to establish new facilities in most suitable places. Take online shopping as an example, to improve the shipping service and minimize storage cost, online sales giants such as Amazon and 360Buy have built numerous warehouses throughout the country. Since each warehouse is constrained by its designed capacity (e.g. storage ability or the number of employees), it is desired to choose good locations for new warehouses. The mobile service is another example showing the important application of this problem.

The problem is based on bichromatic reverse nearest neighbor (BRNN) query [3]. Let W and R denote two sets of locations in the same space. Given a location $w \in W$, a BRNN query returns all locations $r \in R$ whose nearest neighbor is w. Those locations form the BRNN set of w, denoted by $B(w, W)$. In our problem, we denote the facility set as W and the customer set as R. In addition to the common BRNN query, the capacity constraint of facilities is considered. We use L_2 (Euclidean) distance metric and assume that: (1) a customer is served by the nearest facility and (2) service providers have the knowledge of customer distribution and can offer a serving order that

determines the sequence of locations being served. Let $c(w)$ denote the corresponding capacity of a facility and $wt(r)$ denote the weight of a customer location r. A w is responsible to serve its reverse nearest neighbors. And a r that has w as its nearest facility is *within* w's capacity only if w provides service to some customers reside in r.

We use a running example for illustration throughout the paper. For presenting convenience, we set all customer locations the same weight 1, although the proposed techniques can be applied to any weights. We also adopt the serving order that makes w serve its customers r in an increasing order of $dist(r, w)$ for the same reason. Here we use $dist(r, w)$ to denote the distance between r and w. In Fig.1 and 2, circles, small and

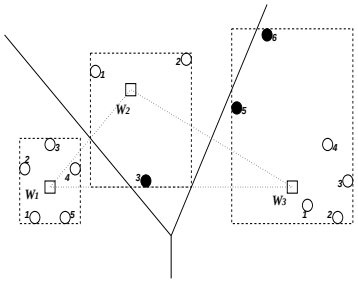


Fig. 1. $c(w_1, w_2, w_3) = (5, 2, 4)$

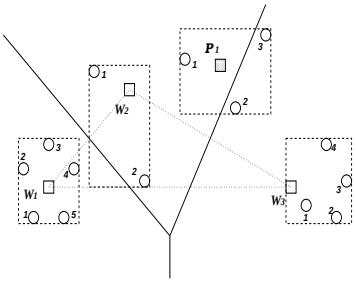


Fig. 2. $c(p_1) = 4$

big rectangles denote customer locations, facilities and BRNN sets respectively, and the number besides each r denotes the serving sequence. In Fig.1, w_2 serves 3 locations. Location 1 and 2 use up $c(w_2)$, thus 3 is out of its capacity. Here black circles denote poorly served ones. After adding p_1, as shown in Fig.2, three locations are well served by p_1. Since p_1 shares the workload of w_2, w_2 has the ability to satisfy previous black locations. The total weight of newly served parts is 3, so p_1 has an increment of 3. Our aim is finding top-k locations that have maximal increments from a candidate set P. We make the following contributions:

- We formulate the problem of top-k most incremental location selection query.
- We propose pruning techniques and an $O(n \log n)$ algorithm to reduce the complexity of query processing.
- We perform extensive experiments and the results confirm the effectiveness of the pruning techniques and the efficiency of the algorithm.

The rest of the paper is organized as follows. Section 2 reviews previous studies on related topics. Section 3 defines the problem and gives a baseline solution. Section 4 describes pruning techniques and the proposed algorithm. The empirical study is given in Section 5, followed by the conclusion in Section 6.

2 Related Work

The concept of *reverse nearest neighbor (RNN)* query [3] has been raised more than ten years. Korn et al.[3] first propose methods to solve RNN query. Paper [5] studies how to discover influence sets based on RNN in frequently updated databases and proposes an efficient algorithm to solve BRNN query. Since we focus on location selection, without proper modification, the proposed methods cannot be applied. Wong et al.[6] study the problem called MaxBRNN. It aims at finding an optimal region that maximizes the size

of BRNN set. Zhou et al.[10] extend the problem to MaxBRkNN. Their problems are different from ours. First, their studies attempt to retrieve the optimal region with highest influence, while ours focuses on selecting top-k ones from a candidate set. Second, capacity constraint is not considered in their studies.

Paper [9] and [2] propose and solve the min-dist optimal location query, which minimizes the average distance from each customer location to its closest facility after adding a new one. Some other papers [8, 1] describe another BRNN related query, called maximal influence query. The influence of a facility is defined as the total weight of its BRNN members. Though these problems are similar to ours, they fail to consider the capacity of each facility. The authors of [7] and [4] study the capacity constrained assignment problem. Paper [7] proposes an algorithm that assigns each $r_i \in R$ to a $w_j \in W$ without exceeding w_j's capacity. Additionally, paper [4] tries to minimize the assignment cost of $\sum_{(r_i,w_j)\in R\times W} dist(r_i, w_j)$. However, they are more suitable for profile-matching applications which provide service using existing facilities. Our problem considers both existing and to-be built ones.

3 Preliminary

3.1 Definitions

First, we give the definition of *ε-served location* to indicate the percentage of customers who are under service in a customer location.

Definition 1. *An ε-served location is a location r that has $\varepsilon \cdot wt(r)$ customers under service, noted as $\varepsilon(r), 0 \leq \varepsilon \leq 1$.*

Given the serving order, w serves its BRNN set members one by one, until some r uses up $c(w)$ or all r are fully served. Hence the ε factor of r can be calculated. Most r are 1 or 0-served. Only those use up $c(w)$, while still have unserved customers reside in are partly served.

Definition 2. *Given W and R, service quality noted as $sq(W)$ equals to $\sum_{r\in R} \varepsilon(r)wt(r)$.*

Service quality is the number of all customers who are under service. To increase $sq(W)$, new facilities should be built and locations are chosen from a candidate set. The concept of *candidate increment* indicates the increment of service quality when a new facility p is added. Let W' denote $W \cup \{p\}$ in the following of this paper.

Definition 3. *Given a p, its increment denoted by $inc(p)$ equals to $sq(W') - sq(W)$.*

The larger increment, the better a candidate location is. Thus we formulate the proposed query as follows: given a constant $k, 0 \leq k \leq |P|$, it finds a set $P' \subseteq P$, so that $|P'| = k$ and $\forall p_i \in P', p_j \in P \setminus P', inc(p_i) \geq inc(p_j)$. Answering the query means finding k most promising candidates that maximize the increase of service quality.

3.2 Basic Solution

The query definition gives a basic solution. It takes R, W, P as input and returns P', which also applies to other algorithms. It scans the candidate set, adds p to W, calculates the new service quality, gets $inc(p)$ then removes p and tries next candidate. Finally it picks out those have top k increments as results.

4 Algorithm

4.1 Notation and Properties

The challenge of the query are two folds. One is that the influence set is not only related to customer locations, as addressed in [8, 1], but also related to facilities. When a new facility is added, it becomes the new nearest facility of some customers and their old facilities are also affected. Here we give the exact definition the *influence set* of a candidate location.

Definition 4. *The influence set of a p denoted by $I(p)$ is $B(p, W') \cup \{r \in B(w, W) | w$ is old nearest facility of $r \in B(p, W')\}$.*

The other challenge is to efficiently deal with large amounts of input data. Hence we introduce the concept of *nearest facility circle* [3] (NFC) of r, which is a circle denoted by $nfc(r, w)$ that centers at r and has a radius of $dist(r, w)$. If p falls into $nfc(r, w)$, then p will become the new nearest facility of r, which is useful to efficiently get $I(p)$.

4.2 Pruning Techniques

To calculate $inc(p)$, it is unnecessary to assign every r to a facility again. As Fig.1 and 2 show, after adding p_1, same customers are assigned to w_1. We know that $w_1 \notin I(p_1)$. So we have the following theorem (proof can be found in the full paper).

Theorem 1. *Given a p, $\forall w \in W \setminus I(p)$, we have $B(w, W) = B(w, W')$.*

Hence, it is possible that we set aside those unaffected locations and focus on a candidate's influence set. This idea is supported by the following theorem.

Theorem 2. *Given a p, $inc(p) = \sum_{r \in I(p)} (\varepsilon_{new}(r) - \varepsilon(r)) wt(r)$, $\varepsilon_{new}(r)$ denotes the new ε factor of r after p is added.*

Maintaining the nn and rnn lists, we can easily calculate $\varepsilon_{new}(r)$ for every $r \in I(p)$, then get $inc(p)$ according to theorem 2, which effectively bounds the search space.

4.3 Spatial Index

We apply several spatial indices to further lower the time complexity. Two time-consuming parts are: (1) to find nearest facility for each $r \in R$ and (2) to get the influence set for each $p \in P$. For the first part, we introduce a spatial structure s_1 that supports NN query to index W. Whereas there is no direct tool to handle the second part. We use NFC and a spatial index s_2 that supports point enclosure query to achieve our goal. The first part has assigned each r to a w, so we can build $nfc(r, w)$ and insert it into s_2. When considering p, we form a p point enclosure query, search s_2 and process all NFCs returned by s_2 to get $I(p)$.

Theorem 3. *Given a structure s that indexes all $nfc(r, w), r \in R$, if we search s with a p-enclosure query, then $\{nfc(r, w) | \forall r \in B(p, W')\} = \{$all NFCs returned by $s\}$*

According to theorem 3 and definition 4, we can get the influence set of p efficiently. The detailed steps of the proposed algorithm are given as follows. And a formal analysis shows the basic solution has a $O(n^3)$ time complexity. After applying pruning techniques, it reduces to $O(n^2)$. When we introduce the spatial indices, the proposed algorithm has a $O(nlogn)$ complexity.

Algorithm 1 Index

1. index each $w \in W$ with s_1
2. **for** each $r \in R$ **do**
3. $w \leftarrow$ NN query result of r on s_1, and assign r to w
4. calculate ε factor for all $r \in R$
5. build $nfc(r, w)$ for each $r \in R$ and insert it into s_2
6. **for** each $p \in P$ **do**
7. search s_2 with a p point enclosure query
8. **for** each $nfc(r, w) \in \{$all NFCs returned by $s_2\}$ **do**
9. add r and $r' \in B(w, W')$ to $I(p)$
10. **for** each $r \in I(p)$ **do**
11. calculate $\varepsilon_{new}(r)$
12. $inc(p) \leftarrow inc(p) + (\varepsilon_{new}(r) - \varepsilon(r))wt(r)$
13. sort $p \in P$ according to $inc(p)$, and get first k candidates as P'

5 Empirical Study

5.1 Experimental Setup

We use C++ to implement all algorithms and conduct all experiments on a PC with an Intel(R) Core 2 Duo 1.7 GHz processor, 2 GB memory and running Window XP platform. We use both real-world and synthetic datasets. Two real-world datasets NE and NA are downloaded from http://www.rtreeportal.org/spatial.html. NE contains 123,593 points in north east of USA. NA contains 24,493 locations in North America. When using NE or NA, we uniformly sample from it to get R,W and P. To simulate real-life scenarios, we generate R, W with Zipfian distribution and P with uniform distribution. The serving sequence we adopt is an increasing order of the distances between customer locations and facilities. The weight of each r in both real and synthetic datasets is set to 1, whose results are similar to those set to any positive integer. The capacity of w and p is generated with uniform distribution ranging [1,40]. Cardinalities are set to $|R| : |W| : |P| = 20 : 2 : 1$ in all experiments. We adopt kd-tree as the spatial index for the nearest neighbor query and R*-tree for the point enclosure query.

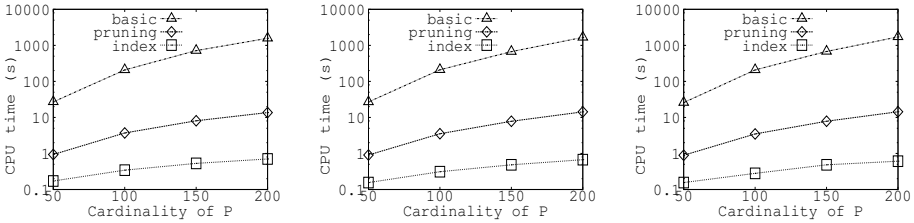
5.2 Comparisons

Since the basic solution is not scalable for large data, we conduct this set of experiments with relative small datasets. Fig.3 shows the proposed algorithm performs nearly 4 orders of magnitude faster than the basic solution when $|P|$ is 200. And the gap will be bigger with the growth of $|P|$. Only adopting the pruning techniques, the basic solution runs 10^2 times faster. Both real and synthetic datasets show similar results.

5.3 Scalability

Then we study the scalability of the proposed algorithm. Apart from two real-world datasets, we synthesize R, W and P with cardinality of 200K, 20K and 10K respectively. Each dataset is used from a quarter to the whole of it. To get the influence of capacity, we sample 3 subsets from them and vary the range of distribution from [1,20]

to [1,60]. Both the execution time and index size are evaluate. Fig.4 and Fig.5 show both the cpu time and index size linearly increase with the growth of input data. Fig.6 shows the influence of different capacities is minor. All datasets support the above observations. So we conclude that the proposed algorithm has good scalability.



(a) NE (b) NA (c) SYN

Fig. 3. Execution time comparison on NE, NA and synthetic datasets.

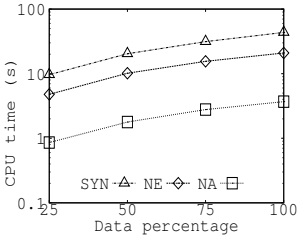


Fig. 4. Execution time

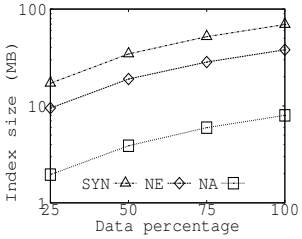


Fig. 5. Index size

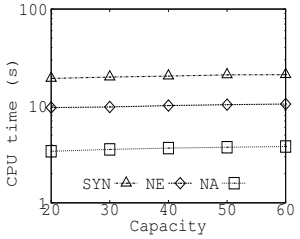


Fig. 6. Capacity influence

6 Conclusion

In this paper, we formulate the top-k most incremental location selection query. Through analyzing the properties of the query, we propose pruning techniques and an $O(n \log n)$ algorithm to answer the query. The results of experiments confirm the effectiveness of the pruning techniques and the efficiency of the proposed algorithm.

Acknowledgements. This work is supported by NSFC under the grant No. 61003085 and HGJ PROJECT 2010ZX01042-002-002-03.

References

1. J. Huang and Z. Wen. Top-k most influential locations selection. In *CIKM '11*, 2011.
2. Q. Jianzhong and Z. Rui. The min-dist location selection query. In *ICDE '12*, 2012.
3. F. Korn. Influence sets based on reverse nearest neighbor queries. *SIGMOD Rec.*, 2000.
4. U. Leong Hou. Capacity constrained assignment in spatial databases. In *SIGMOD '08*, 2008.
5. I. Stanoi. Discovery of influence sets in frequently updated databases. In *VLDB '01*, 2001.
6. R. C.-W. Wong and M. T. Özsu. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proc. VLDB Endow.*, 2009.
7. R. C.-W. Wong and Y. Tao. On efficient spatial matching. In *VLDB '07*, 2007.
8. T. Xia and D. Zhang. On computing top-t most influential spatial sites. In *VLDB '05*, 2005.
9. Zhang. Progressive computation of the min-dist optimal-location query. In *VLDB '06*, 2006.
10. Z. Zhou and W. Wu. Maxfirst for maxbrknn. In *ICDE*, 2011.