MaxFirst for MaxBRkNN

Zenan Zhou[1], Wei Wu[2], Xiaohui Li[1], Mong Li Lee[1], Wynne Hsu[1]

[1]*School of Computing*
National University of Singapore
{zenan, lixiaohui, leeml, whsu}@comp.nus.edu.sg

[2]*Institute for Infocomm Research*
wwu@i2r.a-star.edu.sg

***Abstract*—The MaxBRNN problem finds a region such that setting up a new service site within this region would guarantee the maximum number of customers by proximity. This problem assumes that each customer only uses the service provided by his/her nearest service site. However, in reality, a customer tends to go to his/her k nearest service sites. To handle this, MaxBRNN can be extended to the MaxBRkNN problem which finds an optimal region such that setting up a service site in this region guarantees the maximum number of customers who would consider the site as one of their k nearest service locations. We further generalize the MaxBRkNN problem to reflect the real world scenario where customers may have different preferences for different service sites, and at the same time, service sites may have preferred targeted customers. In this paper, we present an efficient solution called MaxFirst to solve this generalized MaxBRkNN problem. The algorithm works by partitioning the space into quadrants and searches only in those quadrants that potentially contain an optimal region. During the space partitioning, we compute the upper and lower bounds of the size of a quadrant's BRkNN, and use these bounds to prune the unpromising quadrants. Experiment results show that MaxFirst can be two to three orders of magnitude faster than the state-of-the-art algorithm.**

I. Introduction

Reverse Nearest Neighbor (RNN) and Bichromatic RNN (BRNN) queries (and their variants RkNN and BRkNN) have attracted much research attention [10], [11], [12], [1], [3], [15], [8], [14]. The optimal location problem [6] and the MaxBRNN problem [5], [13] are defined using BRNN queries.

Given a set of objects $\mathcal{O}$ and a set of objects $\mathcal{P}$ in space S, a BRNN query [9] issued by an object $p \in \mathcal{P}$ finds the set of objects in $\mathcal{O}$ for which p is their nearest neighbor in $\mathcal{P}$. The optimal location problem [6] aims to find a location point that maximizes the number of objects returned by BRNN. The MaxBRNN problem [5], [13], which is also called the optimal region problem, is to find the region Q such that any point in Q is an optimal location point.

The MaxBRNN problem has many interesting applications. For example, given a set of customers and a set of service sites (e.g. base stations), MaxBRNN finds a region such that setting up a new service site within this region would guarantee the maximal number of customers by proximity.

The state-of-the-art algorithm MaxOverlap [13] uses a technique called region-to-point transformation to solve the MaxBRNN problem. Each service site has a sphere of influence denoted by the nearest location circle (NLC). The MaxOverlap algorithm assumes that each NLC intersects with at least one other NLC, and iterates through the set of intersection points to identify the point that is covered by the largest number of NLCs. The optimal region is given by the overlap of these NLCs.

The need to compute the intersection points of every pair of NLCs is a major drawback in MaxOverlap. It is unable to scale with the number of objects since the number of NLCs will increase, and this will result in a lot more intersection points. MaxOverlap is also affected by the distribution of the objects. A normal distribution means that there will regions with many objects, hence many intersections points in these dense areas. In such situations, MaxOverlap may take hours to return the optimal region.

The MaxBRNN problem assumes that that each customer only uses the service provided by his/her nearest service site. However, in reality, a customer may choose to use the services provided by any of his/her k nearest service sites . To handle this, MaxBRNN can be generalized to the MaxBRkNN problem which finds an optimal region such that setting up a service site in this region guarantees the maximal number of customers who would consider the site as one of their k nearest service locations.

Wong et al. [13] extend MaxOverlap to solve the MaxBRkNN problem by constructing the NLCs according to the k^{th} nearest neighbour instead of the nearest neighbour. This increases the size of the NLCs, leading to more intersection points. Again, the performance of MaxOverlap deteriorates especially when k is large.

In this paper, we design an efficient algorithm called MaxFirst to solve the MaxBRkNN problem. MaxFirst iteratively partitions the space into small quadrants and concentrates on the quadrants that may contain a part of the optimal region. The partitioning process is guided by the upper and lower bounds of the size of BRkNN of the points in the quadrant that are computed with the NLCs that intersect and contain the quadrant. These bounds can be used to prune off unpromising quadrants. Once we find a quadrant that is a part of an optimal region, we can construct the optimal region by the intersection of the set of NLCs that contain this quadrant. With this approach, MaxFirst is able to solve the MaxBRkNN problem at the scale of seconds instead of hours (or even days) required by MaxOverlap for large datasets.

Further, we observe that a customer may have different probabilities of using the services provided by the various

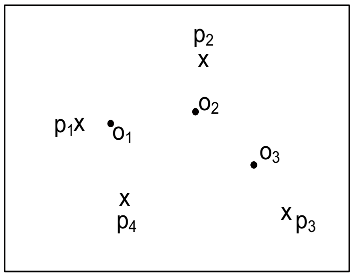


Fig. 1. Customer objects and service sites in our example.

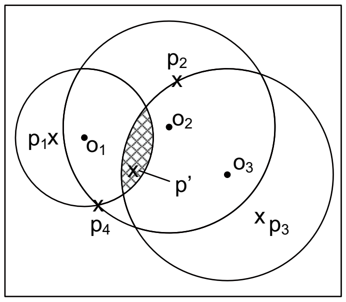


Fig. 2. Region returned by MaxOverlap for our example in Figure 1.

service sites. For example, a customer may use the service provided by his/her nearest service site 80% of the time, while uses the service provided by the second nearest service site 20% of the time. The ability to accommodate different probabilities for different sites cannot be easily handled by MaxOverlap where equal probability is assumed. Here, we define a *probability model* to capture the likelihood of a customer using the service provided by various service sites. The probability model is represented by $\{prob_1, prob_2, ..., prob_k\}$ where k is the value specified in the MaxBRkNN problem and $prob_i$ $(1 \leq i \leq k)$ is the probability that a customer uses the service of his/her ith nearest service site.

In addition, service sites tend to have targeted sets of customers. For example, a service site may prefer to be located in areas where there are more young families. This can be achieved by assigning weights to the customer objects.

Figure 1 shows a simple example where $\mathcal{O} = \{o_1, o_2, o_3\}$ and $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$, $k = 2$, and the weights of objects in $\mathcal{O}$ are all 1. Figure 2 shows the optimal region (shaded) computed by MaxOverlap. If the customers are equally likely to go to any of their k nearest service sites, then this region is optimal. However, if the customers have a probability model $\{0.8, 0.2\}$ and a new service site p' is opened in that region, then o_1, o_2, and o_3 will go to it 20% of the time, and the overall level of interest that p' gets is 60%.

Figure 3 shows the optimal region (shaded) our algorithm MaxFirst will compute when given a probability model $\{0.8, 0.2\}$. If a new service site is set up at any location in that region, o_2, and o_3 will go to it 80% of the time. As a result, the total level of interest that the new service site attracts is 160%. Note that MaxFirst will return the same optimal region as MaxOverlap if the probability model is $\{0.5, 0.5\}$.

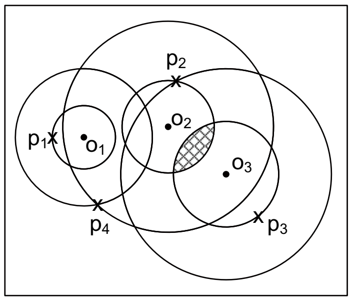


Fig. 3. Region returned by MaxFirst, with probability model $\{0.8, 0.2\}$, for our example in Figure 1.

Our major contributions can be summarized as follows:

- We propose an efficient algorithm called MaxFirst for the MaxBRkNN problem based on space partitioning. We compute with NLCs the upper and lower bounds of the sizes of BRkNN of the locations in a quadrant, and use these bounds to direct the partitioning of space and perform pruning. We also handle the situation when the NLCs intersect at a point.
- We define and incorporate the probability model and customers' weights into MaxFirst to solve the generalized MaxBRkNN problem.
- We conduct extensive experiments to evaluate the performance of MaxFirst. Experiment results show that MaxFirst can be two to three orders of magnitude faster than the state-of-the-art algorithm.

The rest of the paper is organized as follows. Section II gives a survey of the related work. Section III presents our definitions and theorems. Section IV describes the MaxFirst algorithm. We prove the correctness, completeness of MaxFirst and analyze its time complexity in Section V. Experimental results are shown in Section VI and we conclude this paper in Section VII.

II. Related Work

The MaxBRNN problem was first introduced by Cabello et al. [4], [5]. They called it the MAXCOV problem and present a solution for Euclidean space. Their solution computes the NLCs for all objects in $\mathcal{O}$, and determines the arrangement of the NLCs [2]. For each Voronoi cell in the arrangement, the number of NLCs that cover the cell is counted and associated with the cell. The cell with the largest number is the optimal region. The drawback of this solution is that computing the arrangement of a large number of NLCs is expensive. This makes the algorithm not scalable to the dataset size.

Du et al. [6] define and investigate the optimal location problem. They focus on solving the problem in the L1-norm (also known as the Manhattan distance) space. Wong et al. [13] extend the algorithm to the Euclidean space. They define two concepts called consistent region and maximal consistent region. The MaxBRNN problem is to find a maximal consistent region that contains the optimal location. An algorithm called MaxOverlap was proposed to solve it. MaxOverlap utilizes a technique called region-to-point transformation. The basic idea is to find the intersection point that is covered by the most number of NLCs whose overlap corresponds to the optimal region. The main steps in MaxOverlap are: (a) use separate R-trees to index consumer objects and service sites; (b) compute the NLCs of each object and index them using an R-tree; (c) compute the intersection points of each pair of NLCs; (d) for each intersection point, determine the NLCs that cover it; (e) identify the intersection point that is covered by the largest number of NLCs and compute the overlap of these NLCs.

Although MaxOverlap is shown to be much more efficient than the algorithms presented in [4], [5], scalability remains an issue. This is because the computation of intersection points of NLCs is very expensive and these intersection points increase rapidly with the number of objects.

Xia et al. [16] examine a related problem of finding the top-t most influential sites among a given set of service sites. Note that in the top-t influential sites problem, the search space is limited. This makes the top-t influential sites problem very different from the optimal location problem and the MaxBRNN problem.

III. Definitions and Theorems

We use $\mathcal{O}$ to denote the set of consumer objects, and $\mathcal{P}$ to denote the set of service sites. Each object $o \in O$ has a weight, denoted as $w(o)$, to indicate its importance. Different objects can have different weights. For example, if o is a shopping mall, then $w(o)$ could be the average number of shoppers in the mall.

For an object $o \in O$, $kNN(o,k,P)$ represents the set of k objects in $\mathcal{P}$ that are nearest to o. For an object $p \in P$, $BRkNN(p,k,O,P)$ represents the objects in $\mathcal{O}$ that take p as one of their k nearest neighbors in $\mathcal{P}$. For an object $s \notin P$, $BRkNN(s,k,O,P \cup \{s\})$ represents the objects in $\mathcal{O}$ that take s as one of their k nearest neighbors in $\mathcal{P}$ if s is added into $\mathcal{P}$. The size of the BRkNN of a location $s \notin P$ is the sum of the weights of the objects in $\mathcal{O}$ that take s as one of their k nearest neighbors in $P \cup \{s\}$. We assume that no two neighbors have exactly the same distance.

Definition 1. Given an object $o \in \mathcal{O}$, its i^{th} nearest location circle (NLC) c_i, $1 \leq i \leq k$, is the circle centered at the location of o with $dist(o, kNN(o,i,\mathcal{P}))$ as the radius where $dist(o, kNN(o,i,\mathcal{P}))$ is the distance from o to its i^{th} nearest neighbor in $\mathcal{P}$.

We define a probability model to capture the likelihood of an object o going to its i^{th} $(1 \leq i \leq k)$ nearest neighbor in $\mathcal{P}$, denoted as $prob_i$, $\sum_i prob_i = 1$. It is clear that if a new service site is set up in the NLC c_1, it will be o's nearest service site, and the probability that o goes to it is $prob_1$. If a new service site is set up in the annulus formed by c_{i-1} and c_i, it will be the o's ith nearest service site, and the probability that o goes to it is $prob_i$.

Figure 4 shows an example where $k = 3$ and p_1, p_2, and p_3 are o's 3 nearest service sites. The different shades indicate the different probabilities that o goes to a new service site in it. Different objects can have different probability models. These probability models can be derived by conducting surveys on existing service providers and customers.

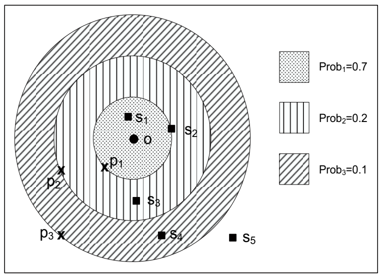


Fig. 4. An object has k NLCs in MaxBRkNN.

If a new service site is set up in the annulus formed by two NLCs c_i and c_{i+1}, and the probability that o goes to it is $prob_{i+1}$, then this service site should have a score $w(o) * prob_{i+1}$. Since c_i is always covered by $c_{i+1}, c_{i+2}, ..., c_k$, we define the score of an NLC c_i as $score(c_i) = w(o) * (prob_i - prob_{i+1})$. This allows a location to accumulate scores from all the NLCs that contain it to obtain its final score as $w(o) * prob_i$. For example, if $k = 2$ and the probability model is $\{0.8, 0.2\}$, and the weight of the object is 1, then $score(c_1)$ is 0.6, and $score(c_2)$ is 0.2. Then a location within c_1 will have a total score $score(c_1)$ + $score(c_2)$ = 0.8. We formally define the scores of an object's NLCs as follows:

Definition 2. Given an object $o \in \mathcal{O}$, a probability model $\{prob_1, prob_2, ..., prob_k\}$, the scores of its NLCs c_i $(1 \leq i \leq k)$ are given by

$score(c_k) = w(o) * prob_k$

$score(c_i) = w(o) * (prob_i - prob_{i+1})$, for $1 \leq i < k$.

With this definition, we have the property that

$\sum_{j=i}^{k} score(c_j) = w(o) * prob_i$.

Definition 3. Let c_i be the i^{th} NLC of an object o. We define the score of a location s w.r.t. c_i as follows:

$$loc\_score(s, c_i) = \begin{cases} score(c_i) & \text{if } s \text{ is inside } c_i \\ 0 & \text{otherwise} \end{cases}$$

Definition 4. Given a set of NLCs C, the total score of a location s w.r.t. C is:

$$total\_score(s, C) = \sum_{c \in C} loc\_score(s, c)$$

Definition 5. Given a region Q and a set of NLCs C, the max_score and min_score of Q are defined as:

$$max\_score(Q) = max_{s \in Q} \quad total\_score(s, C) \qquad (1)$$

$$min\_score(Q) = min_{s \in Q} \quad total\_score(s, C) \qquad (2)$$

For example, in Figure 5, the region Q contains two location points s_1 and s_2. The location s_1 is contained in the second

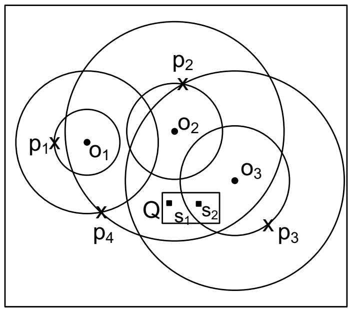


Fig. 5. Example of a region's min_score and max_score.

NLCs of both o_2 and o_3. With a probability model {0.8, 0.2}, the score of s_1 w.r.t these NLCs is 0.2 + 0.2 = 0.4. On the other hand, s_2 is contained in o_3's c_1, c_2 and o_2's c_2. Hence, the score of s_2 w.r.t these NLCs is 0.6 + 0.2 + 0.2 = 1.0. Then max_score(Q) is 1.0 and min_score(Q) is 0.4.

We can compute the upper and lower bounds of the max_score and min_score of a region Q as follows. Let $Q.C$ be the set of NLCs that contain Q and $Q.I$ be the set of NLCs that intersect Q. Since a NLC that contains Q must intersect Q, we have $Q.C \subseteq Q.I$. We use the sum of the scores of the NLCs in $Q.C$ as the lower bound of Q's min_score, and the sum of the scores of NLCs in $Q.I$ as the upper bound of Q's max_score. We establish the correctness of these bounds in Theorem 1.

Theorem 1. Given a region Q, let $Q.C$ be the set of NLCs that contain Q and $Q.I$ be the set of NLCs that intersect Q. Let $Q.min$ and $Q.max$ denote Q's min_score and max_score. Then the lower bound $Q.\widehat{min}$ and upper bound $Q.\widehat{max}$ are given by

$$Q.\widehat{min} = \sum_{c \in Q.C} score(c) \leq Q.min$$

and

$$Q.\widehat{max} = \sum_{c \in Q.I} score(c) \geq Q.max$$

where $score(c)$ is the score of a NLC c.

Proof: Let s_1 be a location in Q with the minimal score among all the locations in Q. Since the NLCs in $Q.C$ contain Q, they all contain s_1, so the score of s_1 is at least $\sum_{c \in Q.C} score(c)$. This proves $\sum_{c \in Q.C} score(c) \leq Q.min$.

Let s_2 be a location in Q with the maximal score among all the locations in Q. The score of s_2 is the sum of the scores it gets from the following two sets of NLCs: the NLCs that contains s_2 and the NLCs where s_2 is on their perimeters. All the NLCs in these two sets intersect s_2 and therefore intersect Q. Hence $Q.I$ is a superset of the set of NLCs where s_2 gets score. This means the score of s_2 is at most $\sum_{c \in Q.I} score(c)$. This proves $Q.max \leq \sum_{c \in Q.I} score(c)$.

We compute Q.$\widehat{min}$ and Q.$\widehat{max}$ by issuing a range query to retrieve the set of NLCs I that intersect the region. Since the set of NLCs C that cover the region is a subset of I, we find C by examining the NLCs in I whether they cover the region. For example, with a probability model $\{0.8, 0.2\}$, $\widehat{max}$ and $\widehat{min}$ of the quadrant q_1 in Figure 6(a) is 1.2 and 0 respectively. o_1's c_2, o_2's c_1 and c_2, and o_3's c_2 intersect q_1, so its $\widehat{max}$ is 0.2+0.6+0.2+0.2=1.2. No NLC contains q_1, so its $\widehat{min}$ is 0. As another example, $\widehat{max}$ and $\widehat{min}$ of the quadrant q_3 in Figure 6(a) is 1.8 and 0. o_1's c_1 and c_2, o_2's c_1 and c_2, and o_3's c_2 intersect q_3, so its $\widehat{max}$ is 0.2+0.6+0.2+0.6+0.2=1.8. No NLC contains q_3, so its $\widehat{min}$ is 0.

Our algorithm uses the bounds Q.$\widehat{min}$ and Q.$\widehat{max}$ to prune regions that cannot contain an optimal location as stated in Theorem 2.

Theorem 2. Given two regions Q_1 and Q_2, if $Q_1.\widehat{min} > Q_2.\widehat{max}$, then Q_2 does not contain an optimal sub-region.
Proof: We prove Theorem 2 by showing that Q_2 does not contain an optimal location. Let p be a point in Q_1, we have $score(p) \geq Q_1.\widehat{min}$. Since $Q_1.\widehat{min} > Q_2.\widehat{max}$, all the points in Q_2 have a score that is smaller than the score of p, hence Q_2 does not contain a point whose score is the maximal in the whole data space.

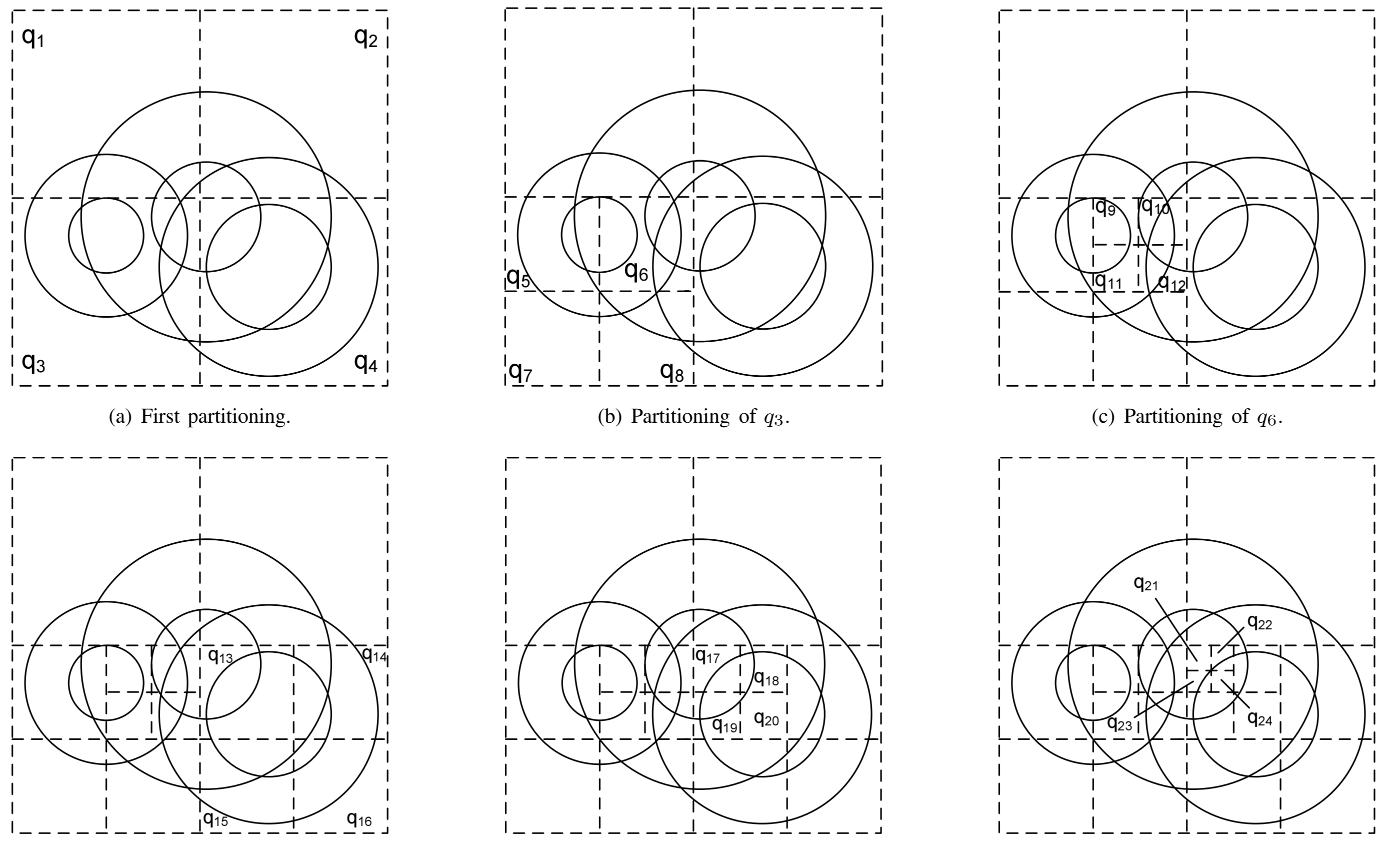
For example, in Figure 6, quadrant q_1 does not contain any optimal location since the $\widehat{min}$ of quadrant q_{24} is larger than q_1's $\widehat{max}$. We can also use the set of NLCs that cover a region and the set of NLCs that intersect a region to perform further pruning. It is formalized in Theorem 3.

Theorem 3. Given two regions Q_1 and Q_2, if $Q_2.I \subseteq Q_1.C$, then Q_2 cannot contain an optimal sub-region such that Q_1 does not intersect the corresponding complete optimal region.
Proof: If Q_2 contains an optimal sub-region, then the complete optimal region must be within the overlap of the NLCs in $Q_2.I$. Since Q_1 is contained by all the NLCs in $Q1.C$, and $Q_2.I \subseteq Q_1.C$, Q_1 is contained by all the NLCs in $Q_2.I$. This means that Q_1 is also an optimal sub-region.

Theorem 3 uses $Q.I$ and $Q'.C$ to identify the quadrants that may contain an optimal sub-region, where part of the optimal region has already been found. For instance, q_{19} in Figure 6 belongs to this category. They all contain an optimal sub-region, but the complete optimal region is the same as the one that contains q_{24} which we have already discovered.

IV. MaxFirst

In this section, we present our solution for the MaxBRkNN problem. Our algorithm, called MaxFirst, solves the problem in two phases. The first phase partitions the space into quadrants and selects the promising quadrants for further partitioning. Partitioning of the space is guided by computing the max_score and min_score of a quadrant and terminates by outputting a set of quadrants with the maximum score. The second phase computes the optimal region from this set of quadrants.



(a) First partitioning. (b) Partitioning of q_3. (c) Partitioning of q_6.

(d) Partitioning of q_4. (e) Partitioning of q_{13}. (f) Partitioning of q_{17}.

Fig. 6. Example to illustrate MaxFirst Phase I.

A. Phase I: Find Quadrants with Maximum Score

Before providing the details of our algorithm MaxFirst, we will first illustrate with an example (see Figure 6) how MaxFirst finds quadrants with the maximum score.

TABLE I
QUADRANTS GENERATED IN PHASE I.

Quadrant	$\widehat{max}$	$\widehat{min}$	Quadrant	$\widehat{max}$	$\widehat{min}$
q_1	1.2	0	q_2	1	0
q_3	1.8	0	q_4	1.6	0
q_5	1	0	q_6	1.8	0
q_7	0.2	0	q_8	0.6	0
q_9	1	0.2	q_{10}	1.2	0.2
q_{11}	1	0.4	q_{12}	1.2	0.2
q_{13}	1.6	0.4	q_{14}	1	0
q_{15}	1	0	q_{16}	1	0
q_{17}	1.6	1	q_{18}	1.6	0.4
q_{19}	1.6	0.4	q_{20}	1.6	1
q_{21}	1	1	q_{22}	1.6	1
q_{23}	1.6	1	q_{24}	1.6	1.6

Given a probability model $\{0.8, 0.2\}$, and the weights of the objects are all 1, the score of an object's first NLC c_1 is 0.6, and the score of its second NLC c_2 is 0.2. Initially the data space is partitioned into four quadrants q_1, q_2, q_3, q_4 as shown in Figure 6(a). Each quadrant has two scores: max_score and min_score. We compute the lower and upper bound of these scores, denoted as $\widehat{max}$ and $\widehat{min}$ respectively, and partition only the quadrant with the maximum $\widehat{max}$. Table I lists the quadrants generated and their corresponding $\widehat{max}$ and $\widehat{min}$ values.

Since the quadrant q_3 has the maximum $\widehat{max}$ value, it is selected for further partitioning (see Figure 6(b)). We split q_3 into four smaller quadrants q_5, q_6, q_7, and q_8. After this, q_6 has the maximum $\widehat{max}$ value, and it is further partitioned into four quadrants, as shown in Figure 6(c). Then q_4 is chosen for partitioning because it has the largest $\widehat{max}$ value. The resultant quadrants are q_{13}, q_{14}, q_{15}, and q_{16}, which are shown in Figure 6(d). Now q_{13} has the largest $\widehat{max}$ value, so it is partitioned into q_{17}, q_{18}, q_{19}, and q_{20} (see Figure 6(e)). Finally, as shown in Figure 6(f), q_{17} is partitioned and we get q_{21}, q_{22}, q_{23}, and q_{24}. We observe that $\widehat{min}$ value of q_{24} is equal to its $\widehat{max}$ value, this implies that we have obtained a consistent quadrant. Since q_{24} has the largest $\widehat{max}$ value, it is also a maximal consistent quadrant and will be returned.

All the remaining quadrants are then pruned. q_{18}, q_{19}, q_{20}, q_{22}, and q_{23} can be pruned because they intersect the same set of NLCs that cover q_{24} (see Theorem 3). The quadrants q_1, q_2, q_5, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{14}, q_{15}, q_{16} and q_{21} are pruned because their $\widehat{max}$ is smaller than q_{24}'s $\widehat{min}$ value (see Theorem 2).

Now we provide the details of MaxFirst Phase I in Algo-

Algorithm 1: MaxFirst - Phase I

```
input  : Set of NLCs of all objects in $\mathcal{O}$
output: Set of quadrants with maximum score.
1  $H$ := $\emptyset$                    /* heap of quadrants */
2  $MaxMin$ := 0; $count$ := 0
3  $R$ := an empty set of quadrants   /* result set */
4  $Q$ := the whole data space
5  $Q.\widehat{min}$ := 0; $Q.\widehat{max}$ := infinite; $Q_{split}$ = $Q$
6  insert $Q$ into $H$
7  while $H$ is not empty do
8      $Q$ := remove top entry from $H$
9      $split$ := false
10     if $Q.\widehat{max} > MaxMin$ then
11         $split$ := true
12     else if $Q.\widehat{max}$ = $MaxMin$ then
13         if $Q.\widehat{min}$ = $Q.\widehat{max}$ then
14             add $Q$ to $R$       /* $Q$ is a result */
15         else
16             if not $\exists\ Q' \in R$ such that $Q'.C$=$Q.I$ then
17                 $split$ := true
18     if $split$ then
19         if $Q.I$=$Q_{split}.I$ AND $Q.\widehat{min}$=$Q_{split}.\widehat{min}$ then
20             count := count +1
21         else
22             count := 0
23         if $count < m$ then
24             $Qs$: = partition $Q$ at its center
25         else
26             if all NLCs in $Q.I - Q.C$ intersect at a point
               $p$ in $Q$ then
27                 $Qs$: = partition $Q$ at $p$
28             else
29                 $Qs$: = partition $Q$ at its center
30             count := 0
31         $Q_{split}$ := $Q$
32         foreach quadrant $qd$ in $Qs$ do
33             compute $\widehat{min}$ and $\widehat{max}$ using NLCs
34             if $\widehat{min} > MaxMin$ then
35                 $MaxMin$ := $\widehat{min}$
36             insert $qd$ into $H$
37 return $R$
```

rithm 1. It takes a set of NLCs as input and returns a set of quadrants with the maximum score. The NLCs are indexed in an R-tree. We begin by partitioning the whole data space into four quadrants. A heap ordered by $\widehat{max}$ is used to prioritize the quadrants. A variable called $MaxMin$ is used to keep track of the maximum $\widehat{min}$ value seen so far. Initially, $MaxMin$ is set to 0. It is used to prune quadrants whose $\widehat{max}$ values are smaller than a quadrant's $\widehat{min}$ value. A flag $split$ is used to indicate whether the current quadrant should be partitioned. If a quadrant is not partitioned, it is either pruned or put into the result set R.

Lines 11-19 uses $MaxMin$ and $\widehat{max}$ to avoid examining the quadrants that do not contain an optimal location. It also utilizes $Q.I$ and $Q'.C$ to identify quadrants that may contain a previously discovered quadrant with maximum score. Finding $Q.I$, the NLCs that intersect Q, can be done quickly by issuing a range query on the R-tree that indexes the NLCs. Similarly, we issue another range query to obtain $Q'.I$. With $Q'.I$, we can compute $Q'.C$, the NLCs that contain Q'.

Lines 20-38 of Algorithm 1 deals with the partitioning of a quadrant. A region under examination is typically partitioned into four equal-size quadrants at its center. However, sometimes we have to split a quadrant at a specific point. This occurs when we need to partition a quadrant Q, and all the NLCs in $Q.I - Q.C$ meet at a point p inside Q. In this case, we have to split Q at p, otherwise we will get a quadrant Q_p (after splitting Q) that *contains* the point p, and Q_p will have the same $\widehat{max}$ value as $Q.\widehat{max}$. Further, since the NLCs in $Q.I - Q.C$ meet at a point, we will never get a region that is covered by all these NLCs. This means that the maximum $\widehat{max}$ value will always be larger than the maximum $\widehat{min}$ value, and the partitioning will not terminate. We call such problem the *intersection point problem*. We tackle the intersection point problem by splitting Q at the point p.

Figure 7(a) shows an example where three NLCs meet at p. In this case, we should partition the quadrant at p as shown in Figure 7(b).

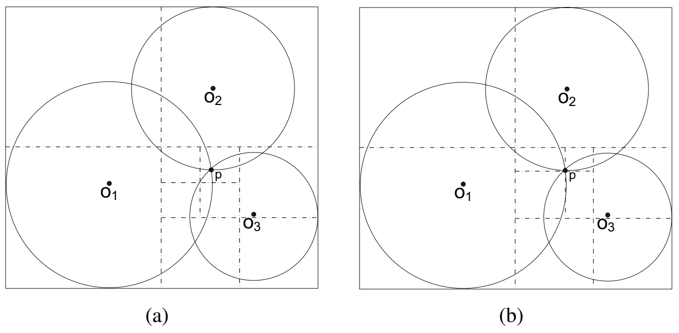


Fig. 7. Example to illustrate the intersection point problem.

The intersection point problem can be detected when the partitioned quadrants intersect the same set of NLCs, and the quadrants have the same $\widehat{min}$ value. The former implies that the quadrants have the same $\widehat{max}$ value, and the probability that we are recursively splitting the same region is high. The latter indicates that the NLCs intersecting the quadrants probably meet at a point.

In order to determine if the NLCs meet at a point, we use a threshold m to control the number of times a quadrant is allowed to be partitioned with the same set of intersecting NLCs and the same $\widehat{min}$ value. When the threshold is exceeded, the algorithm will check whether the NLCs meet at a point. If so, we split the quadrant at that point. The value of m does not affect the correctness of our algorithm, but determines how often the algorithm checks for the intersection point problem.

We carry out an initial experiment to study the effect of

m on MaxFirst's performance. Figure 8 shows the result on a synthetic datasets with 50K uniformly distributed customer objects and 500 service sites. We observe that m has little effect on the performance of MaxFirst. In our experiments, we set m to 4.

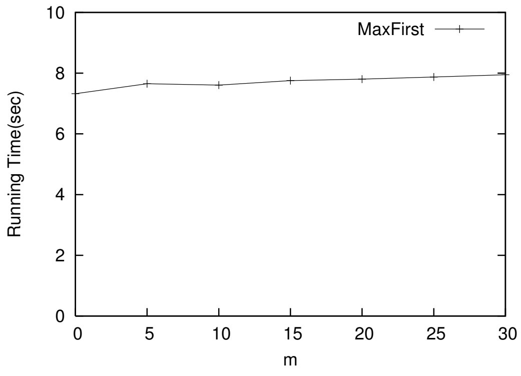


Fig. 8. Effect of m, uniform distribution.

B. Phase II: Compute Optimal Region

The first phase of MaxFirst returns a set of quadrants with the maximum score. The second phase of MaxFirst constructs the optimal region using this set of quadrants.

Given a quadrant Q, the optimal region is simply the overlap of the NLCs that cover Q. Since the set of NLCs that cover Q is $Q.C$, we only need to compute the overlap of the NLCs in $Q.C$. We propose an algorithm that uses a subset of the NLCs to compute the optimal region.

We observe that the circumference of many NLCs do not intersect the circumference of the optimal region. These NLCs cannot be part of the boundary of the optimal region. Hence, we do not need to include them in the computation of the optimal region. Based on this observation, our idea is to iteratively select NLCs that are closest to Q to determine their overlap area, and stop when adding more NLCs will not refine the overlap area.

Figure 9 shows how MaxFirst computes the optimal region given a quadrant Q. The four circles in the figure are the NLCs that cover Q. Given an NLC with center o and radius r, the shortest distance between s and the circumference of the NLC is given by r - $dist(o, s)$ where $dist(o, s)$ is the distance between points o and s. Figure 9(a) shows the shortest distances from the center point s of Q to the circumference of the NLCs.

We can order the NLCs by their shortest distances as follows: o_4's NLC, o_1's NLC, o_2's NLC, and o_3's NLC. We first compute the overlap area of o_4's NLC and o_1's NLC. At the same time, we obtain the maximum distance from s to the overlap area. This is given by $max\{dist(s, v_1), dist(s(v_2)\}$ where v_1 and v_2 are the intersection points of the two NLCs (see Figure 9(b)).

After this, we select the next closest NLC, i.e., o_2's NLC to refine the overlap area, as shown in Figure 9(c). The maximum distance from s to the boundary of the overlap area, given by $max\{dist(s, v_2), dist(s(v_3), dist(s(v_4)\}$, is less than the shortest distance from s to circumference of o_3's NLC. This implies that we have found the optimal region since no further NLC can refine the boundary of this overlap area.

Algorithm 2 shows the details of computing an optimal region given a quadrant. Lines 1-5 set s to the center of Q, and use a heap to order the NLCs based on the shortest distances from their circumference to s. Lines 6-8 compute an overlap area R using the first two NLCs taken from the heap. Line 9 determines the largest distance from s to the boundary of R, denoted by d_{max}. We iteratively select the next NLC from the heap to refine the overlap area R and update d_{max}, until the shortest distance from s to the NLC is larger than d_{max}. We output R as the optimal region.

Algorithm 2: MaxFirst - Phase II

```
input : A quadrant
output: An optimal region
1  $s$ := the center of $Q$
2  $H$ := $\emptyset$    /* a heap containing NLCs using
   distance as key */
3  foreach NLC $c$ in $Q.C$ do
4      $d$ := shortest distance from $s$ to the circumference of
       $c$
5      insert entry ($c$, $d$) to $H$
6  remove entry ($c_1$, $d_1$) from $H$
7  remove entry ($c_2$, $d_2$) from $H$
8  $R$ := overlap of $c_1$ and $c_2$
9  $d_{max}$ := the maximum distance from $s$ to the boundary
   of $R$
10 while $H$ is not empty do
11     remove entry ($c$, $d$) from $H$
12     if $d < d_{max}$ then
13         $R$ := overlap of $R$ and $c$
14         $d_{max}$ := the maximum distance from $s$ to the
           boundary of $R$
15     else
16         return $R$
17 return $R$
```

V. Theoretical Analysis

In this section, we will prove the correctness and completeness of MaxFirst and analyze its time complexity.

A. Proof of Correctness

The correctness of MaxFirst depends on finding the quadrants with the maximum score correctly. Here, we prove that the algorithm will terminate and return a quadrant that has the maximum score. This requires us to show that after a finite number of splits of the quadrant with the maximum $\widehat{max}$, we will get a quadrant Q such that $Q.\widehat{max}$=$Q.\widehat{min}$ and $Q.\widehat{max}$ is the maximum $\widehat{max}$ among all the quadrants. When $Q.\widehat{max}$=$Q.\widehat{min}$, we have $Q.\widehat{max} = Q.max = Q.min =$

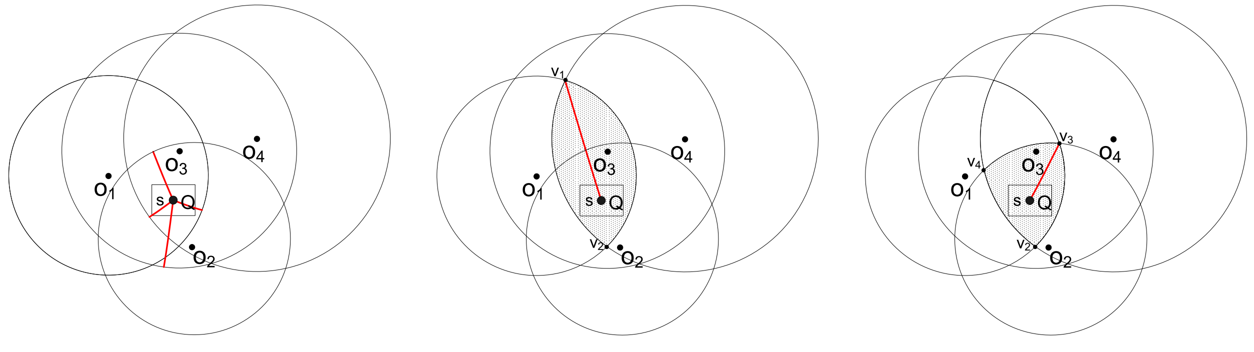


Fig. 9. Example to compute the optimal region from a quadrant.

$Q.\widehat{min}$, so Q is a consistent region and its score is $Q.\widehat{max}$. Since $Q.\widehat{max}$ is the maximum, Q is a quadrant with the maximum score. Now let us prove that we will get such a Q.

Let Q_s be the quadrant whose $\widehat{max}$ is the maximum. If $Q_s.\widehat{max} = Q_s.\widehat{min}$, we are done. If $Q_s.\widehat{max} > Q_s.\widehat{min}$ (note that $Q_s.\widehat{max}$ cannot be smaller than $Q_s.\widehat{min}$), then we have $Q_s.I \supset Q_s.C$. If the NLCs in $Q_s.I - Q_s.C$ intersect at several points in Q_s, a limited number of splits of Q_s will eventually put the intersection points into some quadrants, so we will get quadrants that contain either one or zero intersection point. If Q_s contains only one intersection point of the NLCs in $Q_s.I - Q_s.C$, MaxFirst will partition Q_s at that intersection point, so we will finally get quadrants that contain no intersection point.

Now let us consider a Q_s such that the NLCs in $Q_s.I - Q_s.C$ do not intersect in Q_s. Since the NLCs in $Q_s.I - Q_s.C$ do not intersect in Q_s, after a limited number of splits of Q_s, we will get a Q_s whose $Q_s.I - Q_s.C$ contains only one NLC. Let c be the NLC in $Q_s.I - Q_s.C$. Since c must cover a part of Q_s, after a limited number of splits of Q_s, we will get a Q_s that is contained by c. Now $Q_s.I - Q_s.C$ is empty and $Q_s.I = Q_s.C$, we have $Q_s.\widehat{max} = Q_s.\widehat{min}$. This proves that we will get a quadrant Q such that $Q.\widehat{max} = Q.\widehat{min}$ and $Q.\widehat{max}$ is the maximum $\widehat{max}$.

Intuitively, the correctness of MaxFirst is guaranteed by the following properties of $\widehat{min}$ and $\widehat{max}$ during the splits of the quadrants:

1) maximum $\widehat{max}$ decreases.
2) maximum $\widehat{min}$ increases.
3) maximum $\widehat{max}$ and $\widehat{min}$ converge to a same value.

B. Proof of Completeness

We have proved that MaxFirst will find a quadrant with the maximum score correctly. Next, we prove that for each optimal region, MaxFirst will find a quadrant with the maximum score that is contained within the optimal region. Let Q_i and Q_j be the quadrants of two optimal regions. Without loss of generality, suppose MaxFirst has found Q_i. We will show that MaxFirst will also find Q_j instead of pruning it.

Recall that MaxFirst uses two pruning criteria to prune a quadrant. The first criterion basically says that Q_j can be pruned if $Q_i.min > Q_j.max$. Since both Q_i and Q_j are quadrants of optimal regions, we have $Q_i.max = Q_i.min = Q_j.max = Q_j.min$, therefore this pruning criterion does not apply.

The second criterion basically states that Q_j can be pruned if $Q_i.C \supseteq Q_j.I$. Since Q_i and Q_j are quadrants of different optimal regions, $Q_i.C \neq Q_j.C$. In other words, $Q_i.C \not\supseteq Q_j.C$. We also know that Q_j is a quadrant with the maximum score, hence $Q_j.C = Q_j.I$. So, we have $Q_i.C \not\supseteq Q_j.C = Q_j.I$. Hence the second pruning criterion also do not apply.

Thus, Q_j will not be pruned and MaxFirst will return it as a quadrant and the corresponding optimal region will be also found. This proves that MaxFirst will find all the optimal regions.

C. Time Complexity

Algorithm MaxFirst has a pre-processing step to construct NLCs by performing a nearest neighbor query to find the nearest p in $\mathcal{P}$ for each object o in $\mathcal{O}$. This step requires $O(|\mathcal{O}|log|\mathcal{P}|)$ assuming the nearest neighbor query can be evaluated in $O(log|\mathcal{P}|)$ using an R-tree index.

In MaxFirst's Phase 1 (Algorithm 1), we recursively partition the space to find the set of quadrants with the maximum score. The number of times we need to partition corresponds to the height of a quadtree. In [7], the asymptotic height of the quadtree is $log|\mathcal{O}|$.

For each quadrant, we perform a range query to find all the NLCs that overlap with it. In the literature, the range query can be executed in $O(w+log|\mathcal{O}|)$ time where w is the greatest result size of a range query. In other words, Phase 1 requires $O(wlog|\mathcal{O}| + log^2|\mathcal{O}|)$ time.

Phase 2 (Algorithm 2) constructs the optimal regions from the set of quadrants found in Phase 1. This involves finding the overlap area of the NLCs which requires $O(w)$.

Hence, the overall running time of algorithm MaxFirst is $O(|\mathcal{O}|log|\mathcal{P}| + wlog|\mathcal{O}| + log^2|\mathcal{O}| + w)$.

VI. Performance Study

We have conducted extensive experiments to study the performance of our algorithm MaxFirst. We also compare

MaxFirst with the state-of-the-art algorithm MaxOverlap for the MaxBRNN problem [13]. We have implemented MaxFirst in C++, and used the original C++ implementation of MaxOverlap that we got from the authors of [13]. All experiments are carried out on a Linux machine with an Intel(R) Core2 Duo 2.33 GHz CPU and 3.2GB memory.

The aim of the experiments is to study the time needed by the algorithms to solve the MaxBRkNN problem under various settings. We use the total processing time (including the time spent on computing the NLCs) as the performance measure. We investigate the scalability of the algorithms with respect to the number of objects in the customer dataset, the number of objects in the service site dataset, and the value of k. We use both real world data and synthetic data in the experiments.

We generate synthetic data having uniform distribution and normal distribution. In each set of experiments, the customer dataset and the service site dataset have the same distribution. We also make the size of $\mathcal{P}$ smaller than the size of $\mathcal{O}$, because in reality the number of service sites is always much fewer than the number of customer objects. We find that the weights of the customer objects do not affect the relative performance of the algorithms, so we only show the experiments where the weight of the customer objects is set to 1. Table II lists the parameters and their values.

We download two real world datasets UX and NE from http://www.rtreeportal.org/spatial.html. Table III lists the details of the real world datasets. UX contains points of populated places and cultural landmarks in US and Mexico; NE contains points representing the geographical locations in North East America.

The default probability model used in our experiments is $\{1/C, 1/2C, 1/3C, ..., 1/kC\}$ where $C = \sum_{i=1}^{k} 1/i$ is a normalization constant. This probability model reflects that the likelihood of a customer going to its i^{th} service site is inversely proportional to its distance.

Note that we can compare MaxFirst with MaxOverlap only when the customer objects are equally likely to visit any of its k nearest service sites. The effect of k and the effect of probability models are studied and presented in Section VI-C and Section VI-D.

TABLE II
PARAMETER SETTINGS

Parameter	Default	Range
k	1	1-15
Number of customer objects, $\lvert\mathcal{O}\rvert$	50K	10-100K
Number of service sites, $\lvert\mathcal{P}\rvert$	500	100-1K

TABLE III
SUMMARY OF REAL WORLD DATASETS

Dataset	Cardinality
UX	19499
NE	123,593

A. Effect of the Number of Customer Objects

We first study the effect of $|\mathcal{O}|$ on the performance of MaxFirst and MaxOverlap under different distributions. We fix the number of service sites $\mathcal{P}$ at 500, and vary the number of customer objects from 10K to 100K. Figures 10(a) and 10(b) show the algorithms' performances for datasets with uniform and normal distributions respectively. Note that the figures are plotted in log-scale.

Clearly, MaxFirst outperforms MaxOverlap, and the performance difference between them is huge (up to several orders of magnitude) when the number of customer objects is large. As the number of customer objects increases, the running times of both the algorithms increase, but the rate of increase for MaxFirst is much slower than that for MaxOverlap. We see that MaxFirst is scalable with the number of customer objects. This is because MaxFirst only partitions the quadrants that potentially may contain an optimal region. Intuitively, MaxFirst only partitions the quadrants where the density of NLCs is the highest. Although the number of NLCs increases with the number of customer objects, the number of quadrants that needs to be partitioned is $log|\mathcal{O}|$ [7]. On the other hand, MaxOverlap needs to compute all the intersection points of every pair of NLCs, which requires $O(|\mathcal{O}|^2)$.

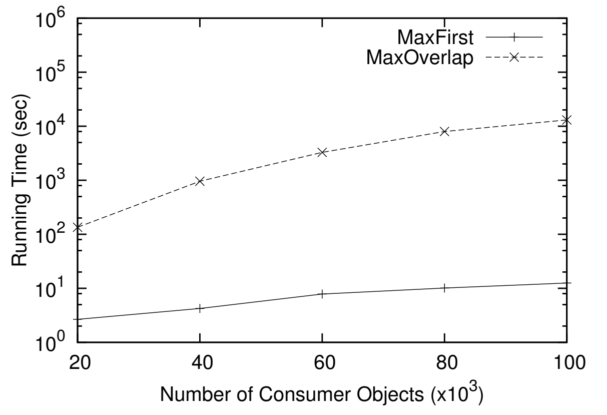
Comparing Figures 10(a) and 10(b), we observe that data distribution affects the algorithms' performances. Both algorithms spend more time on datasets with normal distribution. For MaxOverlap, a normal distribution means that there will be more intersections points in the dense area. For MaxFirst, a normal distribution means that there will be more NLCs in the region with the highest density of NLCs, but the algorithm remains scalable.

B. Effect of the Number of Service Sites

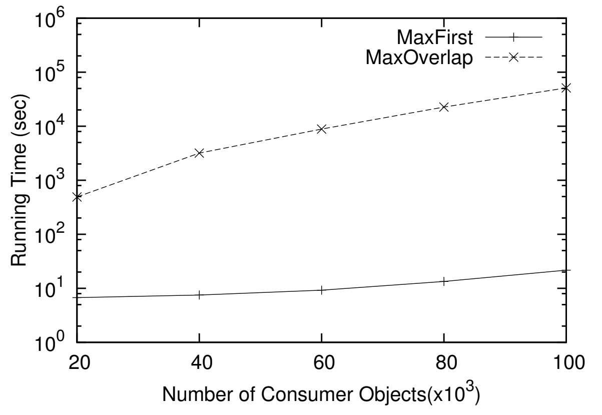
Next, we study the effect of the number of service sites $|\mathcal{P}|$ on the the performance of MaxFirst and MaxOverlap. We fix the number of customer objects at 50K, and vary the number of service sites from 100 to 1000. Figures 11(a) and 11(b) show the algorithms' performance on datasets with uniform and normal distributions respectively.

We observe that the processing times of both MaxFirst and MaxOverlap decrease as the number of service sites increases. When there are more services sites, the NLCs become smaller, leading to less overlap. This translates to fewer number of quadrants that needs to be further partitioned. Hence, the processing time of MaxFirst decreases as $|\mathcal{P}|$ increases. Smaller NLCs also mean that the NLCs will have smaller number of intersection points, and this explains why the processing time of MaxOverlap decreases as $|\mathcal{P}|$ increases.

Comparing Figures 11(a) and 11(b), we observe that the algorithms' running times decrease faster with $|\mathcal{P}|$ for uniform distribution. This is because when $\mathcal{O}$ and $\mathcal{P}$ are uniformly distributed, the size of the NLCs decreases rapidly when the number of service sites increases, thus reducing the running times. When the data is normally distributed, most of the objects in both $\mathcal{O}$ and $\mathcal{P}$ are concentrated in a small area. For

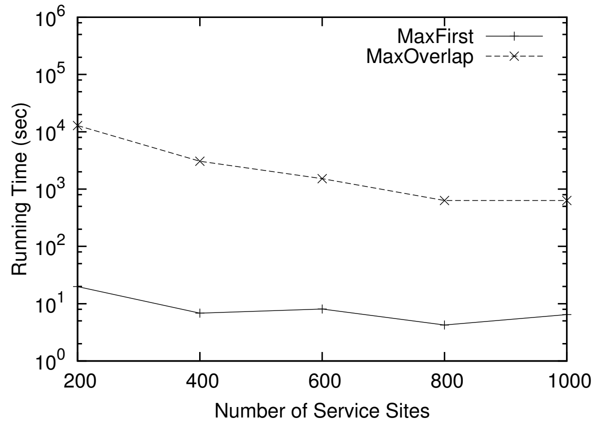


(a) Uniform distribution

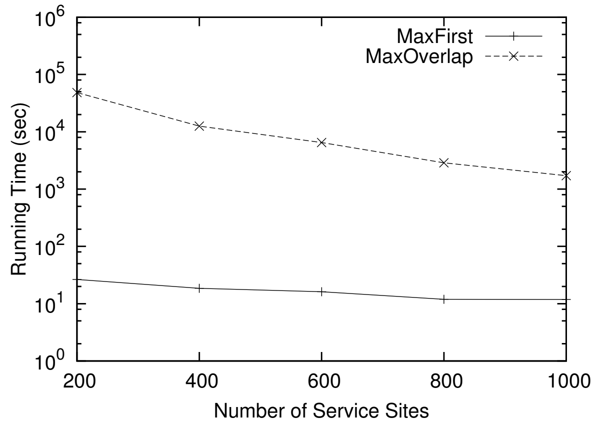


(b) Normal distribution

Fig. 10. Effect of $|\mathcal{O}|$ on runtime (log-scale)



(a) Uniform distribution



(b) Normal distribution

Fig. 11. Effect of $|\mathcal{P}|$ on runtime (log-scale)

this reason, increasing the number of service sites does not affect the size and density of NLCs in that area significantly. Hence, the algorithms' running times decrease slowly with $|\mathcal{P}|$.

C. Effect of k

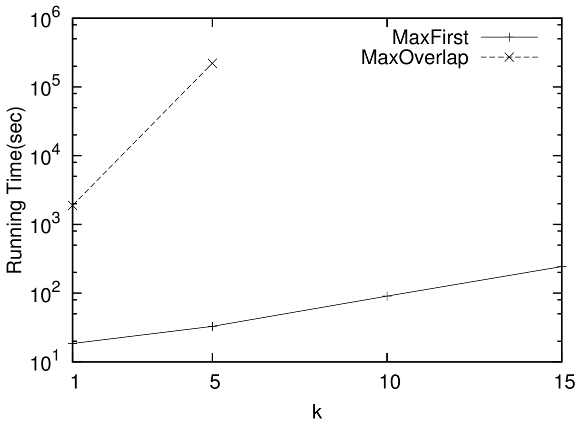
Here, we study the effect of k on the algorithms' performances in solving the MaxBRkNN problems. Since MaxOverlap assumes equal probabilities for the k nearest service site, we set the probabilities in the probability model for MaxFirst to be the same. This will ensure that MaxFirst and MaxOverlap return the same optimal regions. The default synthetic datasets with uniform distribution are used.

Figure 12(a) shows the results in log-scale. We see that the processing times of both MaxFirst and MaxOverlap increase with k, and the processing time of MaxOverlap increases much faster than MaxFirst. As the value of k increases, the sizes of the NLCs become larger. As a result, the NLCs will have more intersection points, so the performance of MaxOverlap deteriorates very fast. For MaxFirst, as k increases, it needs to compute the scores of more NLCs, hence it takes more time to find the optimal regions. Note that the graph for MaxOverlap cannot be completed because when the value of k increases, MaxOverlap needs days to solve the MaxBrkNN problem.

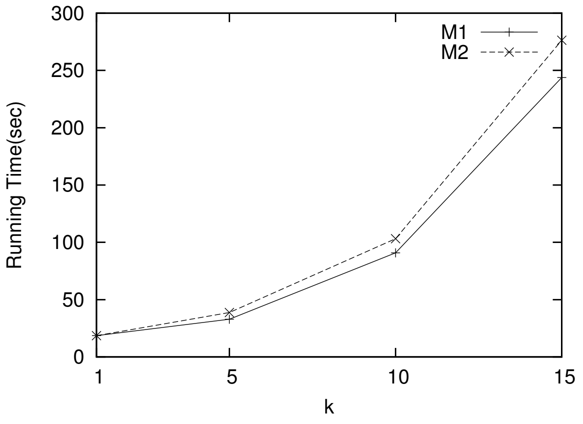
D. Effect of Probability Models

Finally, we study the effect of probability models on MaxFirst's performances in solving the MaxBRkNN problems. Note that MaxOverlap does not apply in such cases. Two series of probability models, $M1$ and $M2$, are generated for k = 1, 5, 10, 15. A probability model of size k in $M1$ is given by $\{k/D, (k-1)/D, (k-2)/D, ..., 1/D\}$ where $D = k(k+1)/2$, while a probability model of size k in $M2$ is given by $\{1/C, 1/2C, 1/3C, ..., 1/kC\}$ where $C = \sum_{i=1}^{k} 1/i$. Note that $M1$ is the normalization of $\{1, 1/2, 1/3, ..., 1/k\}$, and $M2$ is the normalization of $\{k, k-1, k-2, ..., 1\}$.

Figure 12(b) shows the performance of MaxFirst on the MaxBRkNN problem where the probabilities in the probability

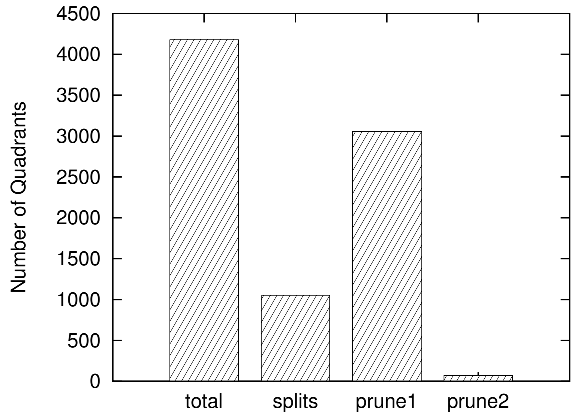


(a) Same probabilities in probability model

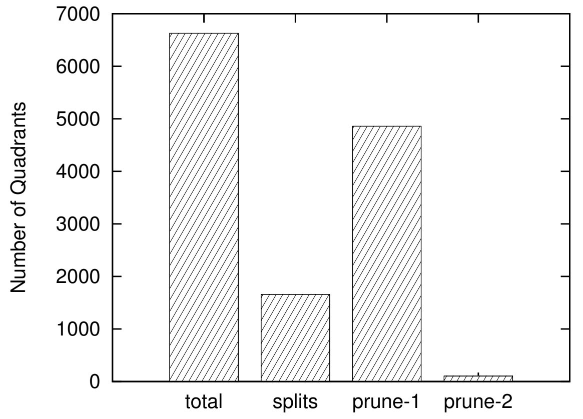


(b) Different probability models

Fig. 12. Effect of k on runtime



(a) Uniform distribution



(b) Normal distribution

Fig. 13. Number of quadrants examined, partitioned and pruned in MaxFirst with default settings.

model are not the same. Note that the graph is not plotted in log-scale, and each line shows how the running times changes when running on a series of probability models. We see that the two lines are close. The performance of MaxFirst is mainly affected by the value of k and not much by the values in the probability models. This means that MaxFirst works consistently well when different probability models are supplied.

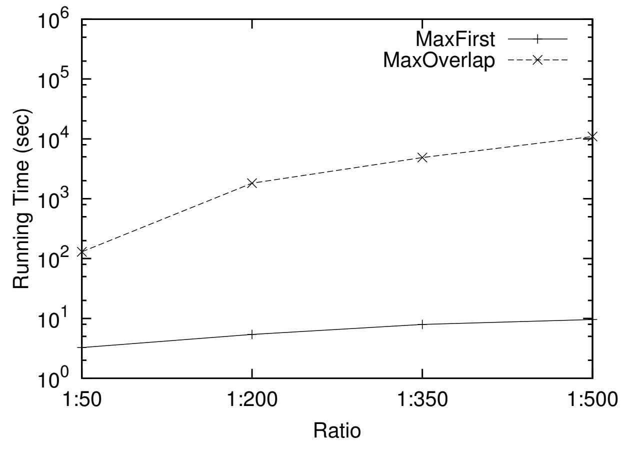
E. Effectiveness of Pruning Strategies in MaxFirst

In this set of experiments, we study the effectiveness of the pruning strategies in MaxFirst by keeping track of the number of quadrants generated, partitioned, and pruned in the course of finding the optimal regions.

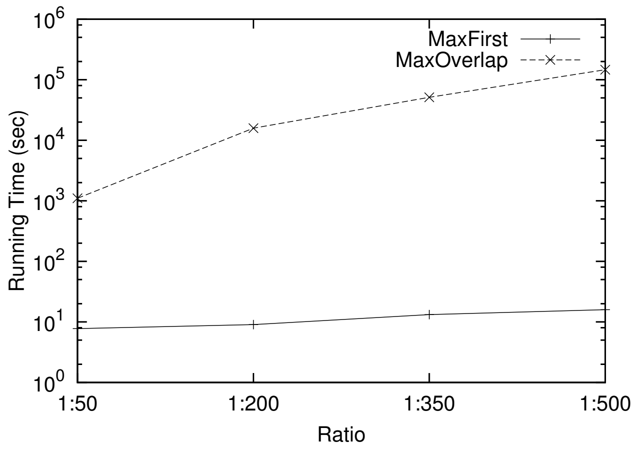
Figure 13(a) shows the respective numbers of quadrants generated, partitioned, and pruned when running MaxFirst on the default synthetic dataset with uniform distribution. Label “total” means the total number of quadrants that are generated, “splits” is the number of quadrants that needs further partitioning, “pruned1” is the number of quadrants that are pruned using Theorem 2, and “pruned2” is the number of quadrants that are pruned using Theorem 3.

From the results, we observe that the number of quadrants that require further partitioning is 1044, which is approximately 2% of the number of customer objects. Majority of the quadrants that cannot contain an optimal region has been pruned by Theorem 2. This makes MaxFirst very efficient in solving MaxBR(k)NN problems.

We repeat the experiment for the default synthetic dataset with normal distribution. Figure 13(b) shows the respective numbers of quadrants. Comparing Figure 13(b) with Figure 13(a), we see that even though the number of quadrants generated is higher for normal distribution, the number of quadrants that need further partitioning is still low (about 3% of $|\mathcal{O}|$). Hence, the overall effect of data distribution on MaxFirst is limited.



(a) UX



(b) NE

Fig. 14. Effect of $|\mathcal{P}|/|\mathcal{O}|$ for real world datasets

F. Results on Real World Datasets

We have seen that both the number of service sites and the number of customer objects affect the time needed by the algorithms to solve the MaxBRNN problem. In this last set of experiments, we use real world datasets to investigate the effect of the ratio $|\mathcal{P}|/|\mathcal{O}|$ on the algorithms' performances. For each real world dataset, we randomly pick $|\mathcal{P}|$ points as service sites. The remaining points become the customer objects.

Figures 14(a) and 14(b) show the running time of the algorithms on the UX and NE datasets when the ratio varies from $1/50$ to $1/500$. We observe that the processing times of both algorithms increase as the ratio decreases. The ratio has a significant effect on the performance of MaxOverlap while it has limited effect on MaxFirst. As the ratio decreases 10 times from $1/50$ to $1/500$, the running time of MaxOverlap increases about 100 times, while the running time of MaxFirst increases only about 3 times. This shows that MaxFirst performs consistently well under various settings.

VII. Conclusion

In this paper, we have highlighted the need for probability model and weights of customers to better reflect the real world scenario for the MaxBRkNN problem. An efficient algorithm called MaxFirst has been designed to solve this generalized MaxBRkNN problem. MaxFirst has two main phases. In the first phase, MaxFirst recursively partitions the space into quadrants, and uses a scoring mechanism to guide the selection of quadrants for further partitioning. In the second phase, MaxFirst utilizes the quadrants found in the first phase as the seeds for computing the optimal regions. Experiment results have demonstrated that MaxFirst is much more efficient than state-of-the-art algorithm MaxOverlap. Furthermore, MaxFirst scales very well with large datasets, and performs consistently well under various settings.

References

[1] Elke Achtert, Christian Bohm, Peer Kroger, Peter Kunath, Alexey Pryakhin, and Matthias Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD*, 2006.

[2] Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling. In *SODA*, 2000.

[3] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal*, 15, 2006.

[4] Sergio Cabello, José Miguel Díaz-Báñez, Stefan Langerman, Carlos Seara, and Inmaculada Ventura. Reverse facility location problems. In *CCCG*, 2005.

[5] Sergio Cabello, José Miguel Díaz-Báñez, Stefan Langerman, Carlos Seara, and Inmaculada Ventura. cility location problems in the plane based on reverse nearest neighbor queries. *European Journal of Operational Research*, 202, 2009.

[6] Yang Du, Donghui Zhang, and Tian Xia. The optimal-location query. In *SSTD*, 2005.

[7] Philippe Flajolet, Gaston Gonnet, Claude Puech, and JM Robson. Analytic variations on quadtrees. In *Algorithmica*, pages 473–500, 1993.

[8] James M. Kang, Mohamed F. Mokbel, Shashi Shekhar, Tian Xia, and Donghui Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, 2007.

[9] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*. 2000.

[10] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000.

[11] Ioana Stanoi, Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*. 2001.

[12] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse knn search in arbitrary dimensionality. In *VLDB*, 2004.

[13] Raymond Chi-Wing Wong, M. Tamer Özsu, Philip S. Yu, Ada Wai-Chee Fu, and Lian Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2009.

[14] Wei Wu, Fei Yang, Chee-Yong Chan, and Kian-Lee Tan. Finch: Evaluating reverse k-nearest-neighbor queries on location data. In *VLDB*, 2008.

[15] Tian Xia and Donghui Zhang. Continuous reverse nearest neighbor monitoring. In *ICDE*. 2006.

[16] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. On computing top-t most influential spatial sites. In *VLDB*, 2005.