# MaxFirst: an Efficient Method for Finding Optimal Regions

**Zhou Zenan**

**NATIONAL UNIVERSITY OF SINGAPORE**

**2010**

# MaxFirst: an Efficient Method for Finding Optimal Regions

**Zhou Zenan**

*(B.COMP, BJTU)*

**A THESIS SUBMITTED**

**FOR THE DEGREE OF MASTER OF SCIENCE**

**DEPARTMENT OF COMPUTER SCIENCE**

**NATIONAL UNIVERSITY OF SINGAPORE**

**2010**

# Acknowledgment

The first people I should thank are Prof. Wynne Hsu and Prof. Mong Li Lee. Without them, this thesis would not have been possible. I appreciate their vast knowledge in many areas, and their insights, suggestions and guidance that helped to shape my research skills.

I thank all the students in the database lab, whose presence and fun loving spirit made the otherwise grueling experience tolerable. I enjoyed all discussions we had on various topics and had lots of fun being a member of this fantastic group. I would especially like to thank Wang Guangsen, Li Xiaohui, Han Zhen, Zhou Ye, Chen Wei, Patel Dhaval and all the other current members in DB lab 2. Their academic and personal help are of great value to me. They are such good and dedicated friends.

Last but not least, I thank my family for always being there when I needed them most, and for supporting me through all these years.

# Summary

The mass adoption of GPS on vehicles and mobile devices has made it very easy to collect location data. Many challenges arise in the management of location data, in particular when it involves the dynamic locations of moving objects. The efficient processing of location-based queries is one of the challenges that are important for system performance and the provision of location-based services. One particular challenge in managing location data is the efficient processing of location-based queries. Besides the classical snapshot range query and k nearest neighbors (kNN) query, continuous versions of these queries, i.e. continuous range query and continuous kNN query, are also useful in the moving objects databases. In this thesis, we focus on the problem of finding optimal regions.

The optimal location problem [15] aims to find a location $q$ in $S$ that maximizes the number of objects in BRNN($q$, $\mathcal{O}$, $\mathcal{P} \cup \{q\}$ ). The MaxBRNN problem [10, 11, 55], which is also called the optimal region problem, is to find the region $Q$ in $S$ where any location in $Q$ is an optimal location. The region obtained by MaxBRNN is called the *optimal region*. It is clear that solving the MaxBRNN problem also solves

the optimal location problem.

The MaxBRNN problem has many interesting applications. For example, if $\mathcal{O}$ is a set of customers and $\mathcal{P}$ is a set of convenient stores, then the result of the MaxBRNN problem is the region where setting up a new convenient store can attract the maximal number of customers by proximity.

In this thesis we propose an efficient algorithm called MaxFirst for solving the MaxBRNN problem, and we also discuss the problem of generalizing the MaxBRNN problem to a MaxBRkNN problem. Although [55] has provided a variant of MaxBRNN based on the BRkNN queries, we provide a more practical and general definition of the MaxBRkNN problem and show that our MaxFirst algorithm can be used immediately to solve the MaxBRkNN problem.

# Contents

**Acknowledgments** **3**

**Summary** **4**

**Contents** **6**

**List of Figures** **9**

**List of Tables** **11**

**1 Introduction** **12**

1.1 Motivation: Management of Location Data . . . . . 13

1.2 Moving Objects and Location Data . . . . . 13

1.3 Applications of Moving Objects Location Data . . . . . 14

1.4 Challenges in the Management of Location Data . . . . . 15

1.5 Objectives and Contributions . . . . . 16

1.6 Problem Definition . . . . . 19

1.7 Organization . . . . . 21

**2 Related Work** **22**

2.1 R-tree . . . . . 22

2.2 Snapshot k Nearest Neighbor Queries . . . . . 24

2.3 MaxBRNN . . . . . 25

**3 MaxFirst** **29**

3.1 Notation and Definitions . . . . . 29

3.2 Find Optimal Sub-Regions . . . . . 33

3.2.1 Algorithm . . . . . 37

3.2.2 Partitioning of a Quadrant . . . . . 39

3.2.3 Proof of Correctness . . . . . 41

3.3 Find the Whole Optimal Region . . . . . 43

3.4 Complexity Analysis . . . . . 46

**4 Generalization to MaxBRkNN** **48**

**5 Performance Study** **51**

5.1 Effect of m on MaxFirst . . . . . 53

5.2 Effect of the Number of Consumer Objects . . . . . 54

5.3 Effect of the Number of Service Sites . . . . . 55

5.4 Results on Real World Datasets . . . . . 56

5.4.1 Results on MaxBRkNN Problem . . . . . 57

**6 Conclusion** **59**

# List of Figures

3.1 An example of NLCs. . . . . 30

3.2 An example to compute a location's score w.r.t. a NLC. . . . . 31

3.3 An example of a region's min-score and max-score. . . . . 32

3.4 An example of using MaxFirst to find an optimal sub-region. . . . . 37

3.5 Example to illustrate the intersection point problem. . . . . 39

3.6 Example to compute the complete optimal region from an optimal sub-region. . . . . 44

4.1 An object has $k$ NLCs in MaxBRkNN. . . . . 50

5.1 Effect of $m$, normal distribution. . . . . 53

5.2 Effect of $|\mathcal{O}|$, uniform distribution. . . . . 54

5.3 Effect of $|\mathcal{O}|$, normal distribution. . . . . 54

5.4 Effect of $|\mathcal{P}|$, uniform distribution. . . . . 55

5.5 Effect of $|\mathcal{P}|$, normal distribution. . . . . 55

5.6 Effect of $|\mathcal{P}|/|\mathcal{O}|$, UX dataset. . . . . 56

5.7 Effect of $|\mathcal{P}|/|\mathcal{O}|$, NE dataset. . . . . 56

5.8 Effect of $k$, same probabilities. . . . . 58

5.9 Effect of $k$, different probabilities. . . . . 58

# List of Tables

5.1 Parameter settings . . . . . 52

5.2 Summary of real datasets . . . . . 52

Chapter 1

# Introduction

Spatial database and its applications in Geographic Information Systems (GIS) [39] have been a topic of research for many years. The primary focus of conventional spatial database research was on the storage and retrieval of static spatial data that are updated infrequently. Recently, advances in wireless communication, mobile devices, and location systems have enabled us to trace the location of moving objects such as vehicles, people, and animals. This means that spatial databases need to capture the location of moving objects,then we can provide Location-Based Services (LBS) [43] for mobile users.

One particular challenge in managing location data is the efficient processing of location-based queries. Besides the classical snapshot range query and k nearest neighbors (kNN) query, continuous versions of these queries, i.e. continuous range query and continuous kNN query, are also useful in the moving objects databases. In addition, new kinds of location based queries, such as reverse kNN (RkNN) query [30], optimal-location query [15] and optimal-region query [56], also have interesting

applications.

## 1.1 Motivation: Management of Location Data

In the last decade we have witnessed the increasing popularity of mobile devices and location systems. The combination of them enables new location-aware environments where all objects of interest can determine their locations. Both companies and individuals can benefit from having relevant location data. However, managing the location data is challenging because in many applications the objects of interest are moving and their locations change frequently.

## 1.2 Moving Objects and Location Data

In the database research literature, the term "moving objects" refers to objects that move. A car with a GPS receiver and a person with a GPS-enabled cellphone are examples of moving objects. Moving objects refer to a broader range of objects than those with GPS receivers. Other examples of moving object include RADAR [6], Cricket [37], and Active Bats [2]. In addition, many objects in computer games can also be seen as moving objects because they move in the game scenario and their locations are known (at least to the game engine). Nowadays, GPS receivers are not only installed on vehicles, they are equipped on many mobile devices such as cellphones and PDAs. Scientists have put location sensors on wild animals. The vehicles, mobile devices, and sensors are all source of dynamic location data.

## 1.3 Applications of Moving Objects Location Data

Applications may use moving objects location data. They can be divided into two groups: monitoring of moving objects for various reasons (such as safety or productivity), and providing services for the mobile users based on their locations. Applications that benefit from the monitoring of moving objects' locations include traffic control, resource allocation, research of wild life, and a lot more. Locations of moving objects provide information not only on the objects themselves but also on the environments around them. For example, monitoring the locations of vehicles not only lets us query the positions of the vehicles but also enables us to analyze the traffic condition during various time periods in different areas. It is reported in the CarTel project [26] that the location data of a set of vehicles helps the users to find the less congested routes and also facilitate the discovery of potholes on the roads. Location-Based Service (LBS) [38, 43] is believed to be one of the killer applications for mobile computing and wireless data services. Often, mobile users want to find out what services are available around their current locations. For example, a driver may want to know where is the nearest gas station; a soldier in a battlefield may want to know what are within 100 meters from him; a person sitting in a coffee shop may want to know whether any of his/her friends happens to be close to the coffee shop so that he/she can meet the friend and hang out together. Knowing the locations of customers is also very important in mobile-commerce (mobile-commerce is visioned to be the "next big thing"). Mobile customers could find the recommendations (and even advertisements) based on their locations more relevant.

## 1.4 Challenges in the Management of Location Data

Managing the location data of moving objects turns out to be a difficult problem due to the dynamic nature of the moving objects. Existing database technologies are invented for data that change infrequently and their performance deteriorates when applying on moving objects. For example, the R-tree [20] is an index structure widely used in databases systems. However, the R-tree is designed to index data with fixed bounding rectangles that are rarely updated. The update operation in R-tree is expensive, so the R-tree does not perform well when used to index moving objects whose location change constantly with time. A few challenges have been identified for the efficient management of moving object data. They include the modeling and storage of moving objects [4, 17, 18, 24, 45], tracking of moving objects [14, 27, 51, 53], indexing of moving objects [3, 12, 41, 46, 50], processing of location-based queries [7, 16, 19, 25, 28, 36, 59], reducing the communication cost [25, 32, 59] in tracking and query processing, managing uncertainty of location data [13, 35, 52], and protecting the location privacy [9, 33] of mobile users. Researchers have used the term Moving Objects Databases (MOD) [17, 54] to refer to the database systems specially designed for the management of moving objects.

## 1.5 Objectives and Contributions

In this thesis, we focus on Finding Optimal Regions. Given a set of objects $\mathcal{O}$ and a set of objects $\mathcal{P}$ in space $S$, a Bichromatic Reverse Nearest Neighbor query [31] issued by object $p \in \mathcal{P}$ finds the set of objects in $\mathcal{O}$ for which $p$ is their nearest neighbor in $\mathcal{P}$. Formally, BRNN($p$, $\mathcal{O}$, $\mathcal{P}$) = $\{o \in \mathcal{O} : p \in NN(o, \mathcal{P})\}$ where $NN(o, \mathcal{P})$ means the object in $\mathcal{P}$ that is the nearest to $o$.

The optimal location problem [15] aims to find a location $q$ in $S$ that maximizes the number of objects in BRNN($q$, $\mathcal{O}$, $\mathcal{P} \cup \{q\}$ ). The MaxBRNN problem [10, 11, 55], which is also called the optimal region problem, is to find the region $Q$ in $S$ where any location in $Q$ is an optimal location. The region obtained by MaxBRNN is called the *optimal region*. It is clear that solving the MaxBRNN problem also solves the optimal location problem.

The MaxBRNN problem has many interesting applications. For example, if $\mathcal{O}$ is a set of customers and $\mathcal{P}$ is a set of convenient stores, then the result of the MaxBRNN problem is the region where setting up a new convenient store can attract the maximal number of customers by proximity.

In this thesis, we propose an efficient algorithm called MaxFirst for solving the MaxBRNN problem. Algorithm MaxFirst first finds a part of the optimal region and then finds the whole optimal region using the information accumulated during the course of finding a part of the optimal region.

MaxFirst is based on the fact that the optimal region is covered by a set of

nearest location circles [10, 11, 55]. A nearest location circle (NLC) of an object $o \in \mathcal{O}$ is the circle centered at $o$ with the distance from $o$ to its nearest neighbor in $\mathcal{P}$ as radius. The optimal region is the region covered by the maximal number of NLCs. If the objects in $\mathcal{O}$ have weights, the NLCs also have weights. In this case, the optimal region is the region that maximizes the sum of the weights of the NLCs that cover the region.

One key insight is that partitioning the space into small sub-region will always result in a sub-region that is a part of the optimal region as long as the sub-region are small enough. A sub-region is small enough when it is covered by all the NLCs that intersect it.

In order to find a region that is a part of the optimal region while avoiding partitioning the space into too many small sub-regions, MaxFirst recursively partitions the space into quadrants and finds the NLCs that intersect each quadrant. We use these NLCs to estimate the lower bound and upper bound of the size (or total weight) of a quadrant's BRNN. The estimated lower bounds and upper bounds let us concentrate on the quadrants that potentially contain a part of the optimal region. MaxFirst always partitions the quadrant with the maximal upper bound, until it find a quadrant that is a part of the optimal region.

Once a part of an optimal region has been found, we have found the set of NLCs that contain it. The whole optimal region is simply the overlap of these NLCs. We find the whole optimal region by computing the overlap of these NLCs.

Compared to existing solutions [10, 11, 55], MaxFirst has the following ad-

vantages. First, MaxFirst does not make any assumption on the distribution of the NLCs. The state-of-the-art algorithm, MaxOverlap [55], assumes that every NLC intersects with at least one of the other NLCs, and it may return incorrect result when this assumption does not hold. Second, MaxFirst can be several hundred (sometimes even several thousand) times faster than the existing algorithms [10, 11, 55]. While it takes existing algorithms hours (or even days) to solve the MaxBRNN problem when the data size is big, MaxFirst always solves the MaxBRNN problem at the scale of seconds. Third, MaxFirst is very easy to understand. MaxFirst partitions the space into small quadrants (like in the Quadtree indexing structure [42]) and concentrates on the quadrants that may contain a part of the optimal region.

Besides proposing an efficient solution for the MaxBRNN problem, we also discuss the problem of generalizing the MaxBRNN problem to a MaxBRkNN problem. Although [55] has provided a variant of MaxBRNN based on the BRkNN queries, we provide a more practical and general definition of the MaxBRkNN problem and show that our MaxFirst algorithm can be used immediately to solve the MaxBRkNN problem.

Our major contributions can be summarized as follows:

- We propose an efficient algorithm called MaxFirst for the MaxBRNN problem based on space partitioning.
- We show how to estimate the lower bound and upper bound of the size of a region's BRNN, and how to use the bounds to direct the partitioning of space and do pruning.

- We show how to partition a region effectively to handle the problems that certain intersections of NLCs may cause.
- We generalize the MaxBRNN problem to the MaxBRkNN problem, and show how to use MaxFirst to solve it.
- We evaluate the performance of the MaxFirst algorithm with extensive experiments.

## 1.6 Problem Definition

The MaxBRNN problem [55] (called the MAXCOV problem in [10]) and the optimal-location problem [15] are defined using the BRNN queries [31].

Let $\mathcal{O}$ be a set of weighted (consumer) objects and $\mathcal{P}$ be a set of (service site) objects. A Bichromatic Reverse Nearest Neighbor (BRNN) query at point $p \in \mathcal{P}$ finds the objects in $\mathcal{O}$ that take $p$ as their nearest neighbor in $\mathcal{P}$. Formally, let $NN(o, \mathcal{P})$ be the set of objects in $\mathcal{P}$ that are the nearest to the object $o \in \mathcal{O}$, the result set of a BRNN query at $p \in \mathcal{P}$ is:

$$BRNN(p, \mathcal{O}, \mathcal{P}) = \{o \in \mathcal{O} : p \in NN(o, \mathcal{P})\} \tag{1.1}$$

Note that $NN(o, \mathcal{P})$ is a set of objects since it is possible to have multiple objects in $\mathcal{P}$ that have the same shortest distance to $o$.

Let $w(o)$ represent the weight of an object $o \in \mathcal{O}$, the size of $p$'s BRNN, or the *influence* of $p$, is defined as the sum of the weights of the objects in BRNN$(p, \mathcal{O}, \mathcal{P})$. Formally, the influence of an object $p \in \mathcal{P}$ is:

$$\sum_{o \in BRNN(p, \mathcal{O}, \mathcal{P})} w(o) \tag{1.2}$$

For a location $q \notin \mathcal{P}$, its influence is defined as the influence of $q$ after adding it into set $\mathcal{P}$. The following expression formally defines the influence of $q$.

$$\sum_{o \in BRNN(q, \mathcal{O}, \mathcal{P} \cup \{q\})} w(o) \tag{1.3}$$

The **optimal location problem** is to find a location $q \notin \mathcal{P}$ with the maximum influence.

Two concepts called *consistent region* and *maximal consistent region* are defined in [55] to facilitate the definition of the MaxBRNN problem. A region $Q$ is a consistent region if it satisfies the following condition: for any two locations $q_1$ and $q_2$ in $Q$, BRNN$(p_1, \mathcal{O}, \mathcal{P} \cup \{q_1\})$= BRNN$(p_2, \mathcal{O}, \mathcal{P} \cup \{q_2\})$. A consistent region $Q$ is said to be a maximal consistent region if there does not exist a region R such that R covers Q and R is a consistent region.

The MaxBRNN problem [55] (called the MAXCOV problem in [10]) is to find a maximal consistent region that contains the optimal locations. The resultant region

is called the *optimal region*.

## 1.7 Organization

The thesis is organized as follows. Chapter 2 surveys the related work. Chapter 3 presents our MaxFirst algorithm. Chapter 4 extends the MaxBRNN problem to a MaxBRkNN problem. Experimental results are shown in Chapter 5. Finally, we conclude this paper in Chapter 6.

Chapter 2

# Related Work

In this chapter we review the existing works that are related to this thesis. We first introduce the indexing structures R-tree for location data in Chapter 2.1 and describe fundamental KNN algorithms in Chapter 2.2. Then we survey the existing algorithms for finding the optimal regions in Chapter 2.3.

## 2.1 R-tree

R-tree is a kind of tree data structure that is used for spatial access methods, i.e., for indexing multi-dimensional information; for example, the (X, Y) coordinates of geographical data.

The data structure splits space with hierarchically nested, and possibly overlapping, minimum bounding rectangles (MBRs, otherwise known as bounding boxes, i.e. "rectangle", what the "R" in R-tree stands for).

Each node of an R-tree has a variable number of entries (up to some pre-defined maximum). Each entry within a non-leaf node stores two pieces of data: a way of identifying a child node, and the bounding box of all entries within this child node.The insertion and deletion algorithms use the bounding boxes from the nodes to ensure that "nearby" elements are placed in the same leaf node (in particular, a new element will go into the leaf node that requires the least enlargement in its bounding box). Each entry within a leaf node stores two pieces of information; a way of identifying the actual data element (which, alternatively, may be placed directly in the node), and the bounding box of the data element.

Similarly, the searching algorithms (e.g., intersection, containment, nearest) use the bounding boxes to decide whether or not to search inside a child node. In this way, most of the nodes in the tree are never "touched" during a search. Like B-trees, this makes R-trees suitable for databases, where nodes can be paged to memory when needed.

Different algorithms can be used to split nodes when they become too full, resulting in the quadratic and linear R-tree sub-types.R-trees do not historically guarantee good worst-case performance, but generally perform well with real-world data. However, a new algorithm was published in 2004 that defines the Priority R-Tree, which claims to be as efficient as the currently most efficient methods and is at the same time worst-case optimal.

## 2.2 Snapshot k Nearest Neighbor Queries

Here we survey the algorithms for processing a snapshot kNN query.

The algorithms proposed for R-trees [40, 44, 22] are more fundamental because many of the later works are based on these algorithms. They are also more relevant to this thesis because they were designed mainly for geometry data and the techniques provided in them are also applied in our works.

The branch-and-bound algorithm developed by Roussopoulos et al. in [40] for R-tree probably is the most influential work on kNN query processing. The authors use two metrics, namely *mindist* and *minmaxdist*, to prune subtrees when traversing a R-tree in a depth-first manner. The $mindist(q; N)$ is the minimum distance from kNN query point q to node N. The $minmaxdist(q; N)$ is the minimum of the maximum possible distances from q to each face of the MBR of the node N. One property of the R-tree is that there is at least one data point on each face of a node's MBR (simply because the MBR is the minimum bounding rectangle). Because of this property, in each node N there must exist a data point p such that $mindist(q; N) \leq dist(q; p) \leq minmaxdist(q; N)$ where $dist(q; p)$ means the distance between q and p. The following three heuristics are used when searching for the NN (i.e. k = 1) of q. First, a node NA can be discarded if $mindist(q; NA) > minmaxdist(q; NB)$. Second, an object p can be discarded if $dist(q; p) > minmaxdist(q; NB)$. Third, a node NA can be discarded if $mindist(q; NA) > NN_{dist}$ where $NN_{dist}$ is the distance from q to the nearest neighbor found so far.

Cheung and Fu proved in [44] that the third heuristic suffices to find the NN of the query point while achieving the same pruning power as the original algorithm in [40]. In later kNN algorithms the *minmaxdist* metric is not used anymore and only *mindist* is used to prune sub-spaces.

In [22], Hjaltason and Samet propose another branch-and-bound kNN algorithm in the context of solving the distance browsing (retrieve data objects in the order of increasing distance to a query point) problem. Their kNN algorithm also uses *mindist* metric to prune nodes but employs a best-first traversal on the R-tree. A priority queue is used to order the R-tree nodes (based on the *mindist* metric) that are not pruned or explored. The advantage of using the best-first traversal instead of the depth-first traversal is that the algorithm makes global decisions on which node to explore.

## 2.3 MaxBRNN

Reverse Nearest Neighbor (RNN) and Bichromatic RNN (BRNN) queries (and their variants RkNN and BRkNN) have attracted much research attention recently [47, 48, 49, 1, 8, 58, 29, 57]. [31] [48] [57] propose algorithms for processing a BRNN query. These algorithms can find the BRNN objects of a query point efficiently but cannot be used to solve the optimal location and MaxBRNN problem directly. This is because the number of points in the search space is infinite. It is infeasible to retrieve the BRNN for every point and then find the one with the maximum size.

In [10], this problem is shown to be 3SUM-hard where it is proved that solving a 3SUM problem over dataset of size $N$ requires $O(N^2)$ time. That is, it is impossible that we can solve problem MaxBRNN with a subquadratic algorithm. [10] proposes a method based on the arrangement of NLCs of the client points. This method involves three major steps. The first step is to construct a set of NLCs for client points. Similar to our method, this step can be done in $O(|\mathcal{O}|\log|\mathcal{P}|)$ time. The second step is to find an arrangement according to a set of NLCs. The best-known efficient method to find an arrangement [34] has the running time of $O(N^2)$ time where $N$ is the number of points in the dataset. In our case, since each point corresponds to an NLC, $N$ is equal to $|\mathcal{O}|$. The third step is to find the best region by traversing from a Voronoi cell to another cell by the face between these two cells iteratively. Since the algorithm heavily relies on the total number of possible faces between adjacent Voronoi cells used in the arrangement and the total number of possible faces is $O(2^{\gamma(|\mathcal{O}|)})$ where $(\gamma|\mathcal{O}|)$ is a function on $|\mathcal{O}|$ and is $\Omega(\mathcal{O})$, the method is exponential in terms of $|\mathcal{O}|$. Specifically, the complexity is $O(|\mathcal{O}|log|\mathcal{P}|+|\mathcal{O}|^2+2^{\gamma(|\mathcal{O}|)})$. This method is not scalable with respect to dataset size.

Cabello et al [10, 11] defined the MaxBRNN problem (they called it the MAX-COV problem) and presented a solution for Euclidean space. Their solution first computes the NLCs for all the objects in $O$, and then computes the arrangement of the NLCs [5]. Finally, for each 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 limitation of this approach is that computing the arrangement of a large number of NLCs can be very expensive. This makes the

algorithm not scalable with the dataset size.

Wong et al. [55] proposed an algorithm to the MaxBRNN problem in Euclidean space. The algorithm is called MaxOverlap. It solves the MaxBRNN problem using a technique called region-to-point transformation. The basic idea is to find an intersection point of the NLCs that has the maximal influence. MaxOverlap works with the following steps:1) use a R-tree $R_o$ to index the consumer objects $\mathcal{O}$ and another R-tree $R_p$ to index the service site objects $\mathcal{P}$; 2)performing a nearest neighbor query to find the nearest p in $\mathcal{P}$ for each object o in $\mathcal{O}$ to computes the NLCs; 3)use a R-tree $R_{NLCs}$ to index all the NLCs; 4)compute the intersection points of all the NLCs; 5) for each intersection point, use $R_{NLCs}$ find the NLCs that cover it; 6) among the sets of NLCs, find the set whose total weight is the largest; 7) compute the overlap of the set of NLCs found in the previous step. The time complexity is $O(|\mathcal{O}|log|\mathcal{P}| + k^2|\mathcal{O}| + k|\mathcal{O}|log|\mathcal{O}|)$, k is the greatest number of NLCs overlapping with a NLC. It is shown in [55] that MaxOverlap is much more efficient than those presented in [10, 11] and [15].

MaxOverlap is an interesting algorithm, but it has a limitation. It implicitly assumes that every NLC will overlap with at least one of the other NLCs, since MaxOverlap searches for an optimal location in the set of intersection points of the NLCs. However, it is possible (although the probability is low) that a NLC does not intersect with any other NLC at all and the NLC contains optimal locations. Under such circumstances MaxOverlap may return the wrong answer. In addition MaxOverlap does not scale well with the number of objects in $\mathcal{O}$.

In this thesis, we propose a solution to the MaxBRNN problem in Euclidean space. Our algorithm, MaxFirst, also uses the NLCs to find the answer to the MaxBRNN problem. However, instead of computing the complex arrangement of the NLCs or all the intersection points of the NLCs, we use a space partitioning method to find the optimal regions. Furthermore, our algorithm does not make any assumption of the data distribution. MaxFirst also efficient and scalable. Experimental study shows that MaxFirst is much faster than the state-of-the-art MaxOverlap algorithm, and scales well with data size.

Chapter 3

# MaxFirst

In this chapter we present our solution to the MaxBRNN problem. Our algorithm, called MaxFirst, solves the problem in two phases. It first finds a region that is a part of the optimal region by partitioning the space selectively and recursively into small regions and estimating the lower bound and upper bound of each region's BRNN. It then computes the complete optimal region using the information accumulated in the first phase.

We first introduce the definitions that we will use in the description of the algorithms in Chapter 3.1, then describe the two phases of our algorithm in Chapters 3.2 and 3.3.

## 3.1 Notation and Definitions

Besides the notation and terms that we introduced in Chapter 1.6, we define additional terms to facilitate the discussion of our algorithms. In particular, we define

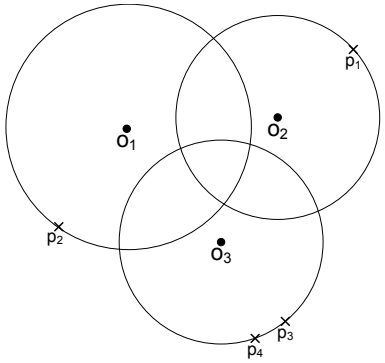



Figure 3.1: An example of NLCs.

the nearest location circle (NLC), a point's score, and a region's score with respect to a set of NLCs.

**Definition** Given an object $o \in \mathcal{O}$, its **nearest location circle** (NLC) $c$, is the circle centered at the location of $o$ with $dist(o, NN(o, \mathcal{P}))$ as the radius where $dist(o, NN(o, \mathcal{P}))$ is the distance from $o$ to its nearest neighbor in $\mathcal{P}$. The **score** of $c$, denoted by $score(c)$, is the weight of $o$.

Figure 3.1 shows a simple example where $\mathcal{O} = \{o_1, o_2, o_3\}$ and $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$. $o_1$'s nearest neighbor in $\mathcal{P}$ is $p_2$, so its NLC is the circle centered at $o_1$ with $d(o_1, p_2)$ as the radius. It is possible that several objects in $\mathcal{P}$ have the same shortest distance to an object in $\mathcal{O}$. For example, $o_3$'s nearest neighbor in $\mathcal{P}$ is $p_3$ and $p_4$. They have the same shortest distance to $o_3$.

**Definition** Let $c$ be the NLC of an object $o$. Given a location $q$, $q$**'s score with**

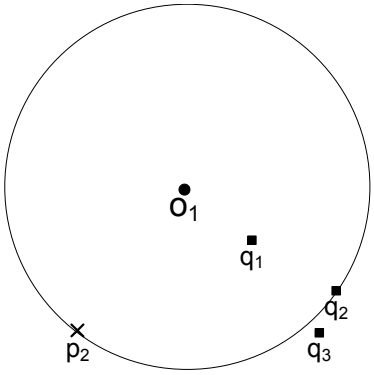



Figure 3.2: An example to compute a location's score w.r.t. a NLC.

**respect to** $c$ is defined as follows:

$$score(q,c) = \begin{cases} score(c) & \text{if } q \text{ is inside } c \\ \frac{1}{|NN(o,\mathcal{P})|+1} & \text{if } q \text{ is on the perimeter of } c \\ 0 & \text{if } q \text{ is outside } c \end{cases}$$

where $|NN(o,\mathcal{P})|$ is the number of objects in $\mathcal{P}$ that are the nearest to $o$.

Consider Figure 3.2. Let $c$ be the NLC of object $o_1$. The score of $q_1$ w.r.t. $c$ is $score(c)$ because it is inside the NLC. The score of $q_2$ w.r.t. $c$ is $\frac{1}{1+1}$, because $q_2$ is on the perimeter of $c$ and $|NN(o_1,\mathcal{P})| = 1$. $q_3$ is outside $c$, hence its score w.r.t. $c$ is 0.

**Definition** Given a set of NLCs $C$ and a location $q$, $q$**'s score with respect to** $C$ is:

$$Score(q,C) = \sum_{c \in C} score(q,c)$$

**Definition** Given a region $Q$ and a set of NLCs $C$, the region's **MaxScore** and

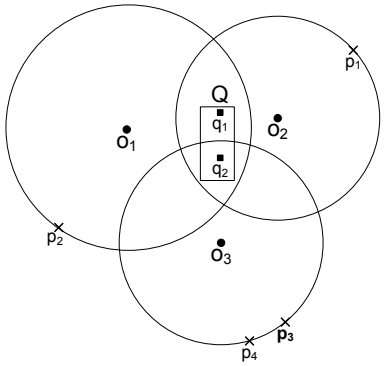



Figure 3.3: An example of a region's min-score and max-score.

**MinScore** are defined as:

$$MaxScore(Q) = max_{q \in Q} \ \ Score(q, C)$$

$$MinScore(Q) = min_{q \in Q} \ \ Score(q, C)$$

Figure 3.3 shows an example. If the weights of $o_1$, $o_2$ and $o_3$ are all 1, the max-score of region $Q$ (the rectangle in the figure) will be 3, and its min-score will be 2. $q_2$ is one of the points in $Q$ that has the maximal score, and $q_1$ is one of the points in $Q$ that has the minimal score.

If a region's min-score is equal to its max-score, then all the points in the region have the same score, and the region is a consistent region (see Chapter 1.6 for the definition of consistent region).

Note that there are an infinite number of points in a region, therefore it is infeasible to compute a region's max-score and min-score based on the definition. We will show in Chapter 3.2 how to compute a lower bound of a region's min-score

and an upper bound of a region's max-score when given a set of NLCs.

With the above definitions, a point's score is the size of its BRNN, and a region's score is the size of the region's BRNN. We next show how we estimate the scores and use the scores to find a part of an optimal region.

## 3.2 Find Optimal Sub-Regions

Our main idea is to utilize space partitioning iteratively to find optimal sub-regions and use these sub-regions to re-construct the entire optimal region. We use space partitioning to find a part of an optimal region. By partitioning the space into sub-regions that are small enough, one of the sub-regions Q must be a part of an optimal region. Then use Q to perform a region query on the R-tree over all the NLCs to get a set of NLCs that create the optimal region. The challenge is to determine whether a sub-region is optimal. Another challenge is to identify the regions that potentially contain an optimal sub-region. Only such regions need to be further partitioned.

Each region has two scores: MaxScore and MinScore. In each iteration, our algorithm MaxFirst estimates the lower and upper bound of these scores, denoted as $\widehat{max}$ and $\widehat{min}$ respectively, and partitions only the regions with the maximum $\widehat{max}$. It uses $\widehat{max}$ and $\widehat{min}$ to prune regions that cannot contain an optimal sub-region. When a region's $\widehat{max}$ is equal to its $\widehat{min}$, and the score is the maximum in the whole data space, then the region is an optimal sub-region.

The NLCs of the objects in $\mathcal{O}$ are used to compute the regions' MaxScore and MinScore. The algorithm starts by computing all the NLCs as follows. We use a R-tree to index the objects in $\mathcal{P}$ [21]. For each object $o$ in $\mathcal{O}$, we retrieve its nearest neighbor in $\mathcal{P}$ using the R-tree with the best-first branch-and-bound NN algorithm [23] and compute $o$'s NLC.

After obtaining all the NLCs, we index them using a R-tree $R_{NLCs}$ and start the score estimation and space partitioning process. This is necessary because we need to quickly determine the $\widehat{max}$ and $\widehat{min}$ of every region. A region under consideration is partitioned into four equal-size sub-regions similar to the Quadtree indexing structure [42]. For certain special regions, we use a different partition method that splits such a region at a specific point into four sub-regions. We will discuss this further in Chapter 3.2.2.

Initially, we partition the whole data space into four quadrants. Given a quadrant Q, we estimate its min-score and maxscore as follows. Perform a region query for Q on $R_{NLCs}$ to get the NLCs that contain Q or intersect Q.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 NLCs in Q.C as the lower bound of $Q$'s MinScore, and the sum of the scores of NLCs in Q.I as the upper bound of $Q$'s MaxScore. We establish the correctness of these bounds with Theorem 3.2.1.

**Theorem 3.2.1.** *Given a region $Q$ and a set of NLCs $N$, let $Q.C$ be the set of NLCs in $N$ that contain $Q$ and $Q.I$ be the set of NLCs in $N$ that intersect $Q$. Let $Q.min$*

*and $Q.max$ denote $Q$'s MinScore and MaxScore. Then the lower bound $Q.\widehat{m}in$ and upper bound $Q.\widehat{m}ax$ are given by*

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

*and*

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

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

*Proof.* Let $q_1$ 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 $q_1$, so the score of $q_1$ is at least $\sum_{c \in Q.C} score(c)$. This proves $\sum_{c \in Q.C} score(c) \leq Q.min$.

Let $q_2$ be a location in $Q$ with the maximal score among all the locations in $Q$. The score of $q_2$ is the sum of the scores it gets from the following two sets of NLCs: the NLCs that contains $q_2$ and the NLCs where $q_2$ is on their perimeters. All the NLCs in these two sets intersect $q_2$ and therefore intersect $Q$. Hence $Q.I$ is a superset of the set of NLCs where $q_2$ gets score. This means the score of $q_2$ is at most $\sum_{c \in Q.I} score(c)$. This proves $Q.max \leq \sum_{c \in Q.I} score(c)$. □

To estimate the lower bound of a regions MinScore and the upper bound of the regions MaxScore, we need to find the set of NLCs C that cover the region and the set of NLCs I that intersect the region. We index the NLCs (in fact their minimum bounding boxes) with an R-tree. The set of NLCs that intersect with a region can be retrieved using the R-tree with an region query. Since the R-tree only indexes

rectangles, we refine the query result set (which is a set of identifiers of NLCs) by checking whether the corresponding NLCs really intersect the region. Since C is a subset of I, we find C by checking the NLCs in I whether they cover the region.Our algorithm uses the bounds Q.$\widehat{min}$ and Q.$\widehat{max}$ to prune regions that cannot contain an optimal location.

We have two pruning criteria. The first criterion is provided in Theorem 3.2.2. This is the main pruning method in our algorithm.

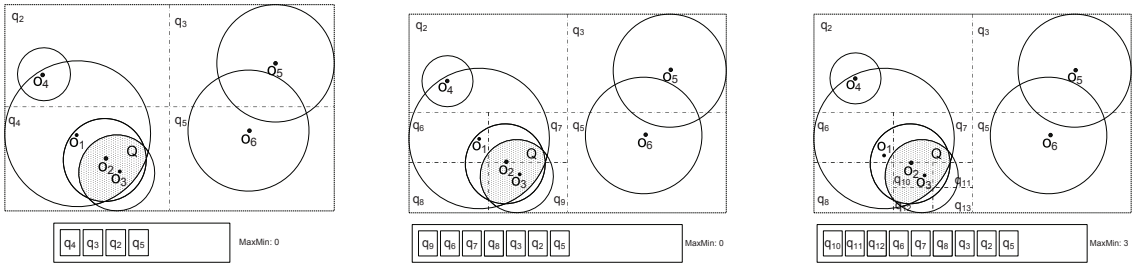
**Theorem 3.2.2.** *Given two regions $Q_1$ and $Q_2$, if $Q_1.\widehat{min} > Q_2.\widehat{max}$, then $Q_2$ does not contain an optimal sub-region.*

*Proof.* We prove Theorem 3.2.2 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. □

The second pruning criterion uses the set of NLCs that cover a region and the set of NLCs that intersect a region to do pruning. It is formalized in Theorem 3.2.3.

**Theorem 3.2.3.** *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




(a) (b) (c)

Figure 3.4: An example of using MaxFirst to find an optimal sub-region.

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. □

### 3.2.1 Algorithm

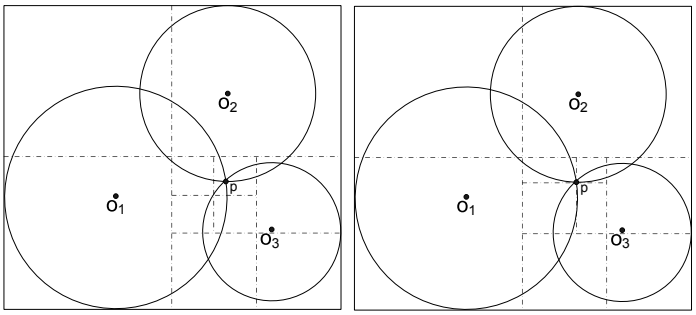
Algorithm MaxFirst always partitions the quadrant with the maximal score, hence the name $MaxFirst$. Figure 3.4 shows how MaxFirst partitions the region recursively to find sub-regions of $Q$. We use a priority queue to order the quadrants that need to be examined. Each quadrant is described using a triplet $< quadrant\_id, \widehat{max}, \widehat{min} >$.

Figure 3.4(a) depicts six NLCs and an optimal region $Q$ (shaded area). We start by partitioning the space into four quadrants. For every quadrant we use it to issue a region query on $R_{NLCs}$ to get a set of NLCs that contain this quadrant and another set of NLCs that intersect with this quadrant, then estimate MaxScore and MinScore of every quadrant: $< q_2, 2, 0 >$, $< q_3, 2, 0 >$, $< q_4, 3, 0 >$, $< q_5, 2, 0 >$. A variable called $MaxMin$ is used to keep track of the maximum $\widehat{max}$ value seen so far. Initially, $MaxMin$ is set to 0. Since $q_4$ has the maximum $\widehat{max}$ value, it is selected

for partitioning next (see Figure 3.4(b)). $q_4$ is split into four smaller quadrants $q_6$, $q_7$, $q_8$, and $q_9$. These quadrants have the same $\widehat{m}ax$ and $\widehat{m}in$ as $q_4$, so they all have the same maximum $\widehat{m}ax$, and $MaxMin$ does not change. Suppose we choose $q_9$ to be further partitioned. Figure 3.4(b) shows the resulting quadrants $< q_{10}, 3, 3 >$, $< q_{11}, 3, 0 >$, $< q_{12}, 3, 0 >$, $< q_{13}, 3, 0 >$. After this partitioning, $MaxMin$ becomes 3. When $q_{10}$ is examined, both its $\widehat{m}ax$ and $\widehat{m}in$ are equal to $MaxMin$, hence it is an optimal sub-region, and is put into the result set. After this, all other quadrants can be pruned. $q_2$, $q_3$ and $q_5$ are pruned because their respective $\widehat{m}ax$ is smaller than $MaxMin$. Other quadrants are pruned because the set of NLCs that intersect them is the same as the set of NLCs that intersects (in fact cover) $q_{10}$.

The above example illustrates that MaxFirst concentrates on the quadrant that has the maximal $\widehat{m}ax$ value. This allows us to concentrate on the regions that possibly contain an optimal sub-region.

Two criteria are used to prune the quadrants. The first criterion (Theorem 3.2.2) uses $MaxMin$ and $\widehat{m}ax$ to avoid examining the quadrants that do not contain an optimal location, e.g., $q_2$, $q_3$ and $q_5$ in Figure 3.4. The second pruning criterion (Theorem 3.2.3 ) 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, $q_6$, $q_7$, $q_8$, $q_{11}$, $q_{12}$ and $q_{13}$ in Figure 3.4 belong to this category. They all contain an optimal sub-region, but the complete optimal region is the same as the one that contains $q_{10}$ which we have already discovered.




(a) (b)

Figure 3.5: Example to illustrate the intersection point problem.

### 3.2.2 Partitioning of a Quadrant

An important detail in Phase 1 of MaxFirst is 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 $Q.I - Q.C$ intersect at a point $p$ inside $Q$ (with no overlap area). 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 $Q.I - Q.C$ have no overlap area, 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*.

Figure 3.5(a) shows an example where three NLCs intersect at $p$ and they have no overlap area. If we always partition a quadrant at its center point, we may always get a quadrant that contains $p$ and we will always partition that quadrant.

We tackle the intersection point problem by splitting $Q$ at the $p$. In MaxFirst,

a quadrant does not include its perimeter. Note that excluding the perimeter of quadrants does not affect the correctness of MaxFirst, because it must be a *region* that gets the maximal score. After partitioning $Q$ at $p$, no quadrant will contain $p$, and the $\widehat{max}$ value of the sub-regions will be smaller than $Q.\widehat{max}$.

We observe that the intersection point problem occurs when a region is continuously partitioned. This happens under two conditions: (1) The partitioned quadrants intersect the same set of NLCs. (2) The quadrants have the same $\widehat{min}$ value. The first condition implies that the quadrants have the same $\widehat{max}$ value, and the probability that we are recursively splitting the same region is high. The second condition implies that the NLCs intersecting the quadrants probably have no common overlap area.

When the above two conditions are satisfied, we perform a check to determine if the NLCs intersect at a point. If so, we split the quadrant at that point. Otherwise, we continue splitting the quadrant at its center. Figure 3.5(b) shows how we split a quadrant at the intersection point $p$.

In Algorithm 1 , we use a threshold $m$ to control the number of times a quadrant is allowed to be partitioned with the same $\widehat{min}$ value and the same set of intersecting NLCs. When the threshold is exceeded, the algorithm will check whether the NLCs intersect 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. In Chapter 5, we include an experiment to study the effect of $m$ on the performance of MaxFirst.

Algorithm 1 shows the details of MaxFirst's Phase 1. It takes a set of NLCs as input and returns a set of regions each of which is an optimal sub-region. A heap ordered by $\widehat{max}$ is used to prioritize the quadrants. 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 $R$.

### 3.2.3 Proof of Correctness

In order to prove the correctness of Algorithm 1, we prove that the algorithm will terminate and return a quadrant that is an optimal sub-region. 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 $Q.\widehat{max} = Q.max = Q.min = Q.\widehat{min}$, so $Q$ is a consistent region and its score is $Q.\widehat{max}$. Since $Q.\widehat{max}$ is the maximum, $Q$ is a region whose score is the maximum, so it is an optimal sub-region. 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 sub-regions, 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

**Algorithm 1**: MaxFirst - Phase 1

**input** : Set of NLCs of all objects in $\mathcal{O}$
**output**: Set of optimal sub-regions

1 $H := \emptyset$ `/* a heap containing quadrants using` $\widehat{max}$ `as key */`
2 $MaxMin := 0$
3 $R :=$ an empty set of quadrants `/* result set */`
4 $Q :=$ the whole data space
5 $Q.\widehat{min} := 0$; $Q.\widehat{max} :=$ infinite
6 *count* :=0 `/* the number of continuous split */`
7 $Q_{split} = Q$ `/* the previous split region */`
8 build an R-tree $R_{NLCs}$ over all the NLCs.
9 use $Q_{split}$ to issue a region query on $R_{NLCs}$ to estimate $Q_{split}.\widehat{min}$ and $Q_{split}.\widehat{max}$
10 insert $Q$ into $H$
11 **while** *H is not empty* **do**
12 $Q :=$ remove top entry from $H$
13 *split* := false `/* flag of split or not */`
14 **if** $Q.\widehat{max} > MaxMin$ **then**
15 *split* := true
16 **else if** $Q.\widehat{max} = MaxMin$ **then**
17 **if** $Q.\widehat{min} = Q.\widehat{max}$ **then**
18 add $Q$ to $R$ `/*` $Q$ `is a result */`
19 **else**
20 **if** $\nexists\ Q' \in R$ *such that* $Q'.C{=}Q.I$ **then**
21 *split* := true
22 **if** *split* **then**
23 **if** $Q.I{=}Q_{split}.I$ *AND* $Q.\widehat{min}{=}Q_{split}.\widehat{min}$ **then**
24 count := count +1
25 **else**
26 count := 0
27 **if** *count* $< m$ **then**
28 $Qs$: = partition $Q$ at its center
29 **else**
30 **if** *all NLCs in* $Q.I - Q.C$ *intersect at a point* $p$ *in* $Q$ **then**
31 $Qs$: = partition $Q$ at $p$
32 **else**
33 $Qs$: = partition $Q$ at its center
34 count := 0
35 $Q_{split} := Q$
36 **foreach** *quadrant qd in Qs* **do**
37 use qd to issue a region query on $R_{NLCs}$ to get $qd.C$ and $qd.I$
38 estimate $qd.\widehat{min}$ and $qd.\widehat{max}$
39 **if** $qd.\widehat{min} > MaxMin$ **then**
40 $MaxMin := qd.\widehat{min}$
41 insert $qd$ into $H$
42 return $R$

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 $Q_s.\widehat{m}ax = Q_s.\widehat{m}in$. This proves that we will get a quadrant $Q$ such that $Q.\widehat{m}ax = Q.\widehat{m}in$ and $Q.\widehat{m}ax$ is the maximum $\widehat{m}ax$.

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

1. maximum $\widehat{m}ax$ decreases.
2. maximum $\widehat{m}in$ increases.
3. maximum $\widehat{m}ax$ and $\widehat{m}in$ converge to a same value.

## 3.3 Find the Whole Optimal Region

The first phase of MaxFirst returns a set of quadrants each of which is an optimal sub-region. The second phase of MaxFirst re-constructs the entire optimal regions using these quadrants.

Given a region $Q$ that is an optimal sub-region, the entire optimal region is simply the intersection of the NLCs that cover $Q$. We can use Q to issue a region

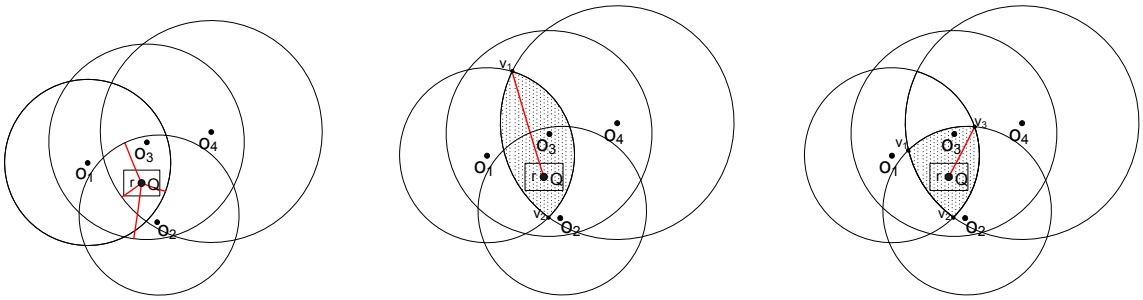



Figure 3.6: Example to compute the complete optimal region from an optimal subregion.

query on the R-tree of all the NLCs to get these NLCs that cover Q. Since the set of NLCs that cover $Q$ is $Q.C$, what we need to do is only to compute the overlap of the NLCs in $Q.C$. We propose an algorithm that uses a subset of the NLCs to compute the complete optimal region.

We observe that the perimeters of many NLCs do not intersect the perimeter of the complete optimal region. Since they do not contribute an edge (in the form of an arc) to the complete overlap region, we do not even need to use them in the computation of the overlap area. Based on this observation, our idea is to compute the overlap of the NLCs that are near to $Q$ and ignore the NLCs whose shortest distances from their perimeters to a point $r$ in $Q$ are larger than the maximum distance from $r$ to the perimeter of the current overlap region.

Figure 3.6 shows how MaxFirst computes the complete optimal region given a quadrant $Q$. The four circles in the figure are the NLCs that cover $Q$. Figure 3.6(a) shows the shortest distances from the center point $r$ of $Q$ to the NLCs' perimeters. The ordering of the NLCs by these distances is: $NLC_4$, $NLC_1$, $NLC_2$, and $NLC_3$. Our algorithm first computes the overlap of $NLC_4$ and $NLC_1$ and the maximum

distance from $r$ to the perimeter of the overlap region. They are shown in Figure 3.6(b). Next, $NLC_2$ is used to clip the overlap region, as shown in Figure 3.6(c). After this, the maximal distance from $r$ to the perimeter of the overlap region is shorter than the shortest distance from $r$ to $NLC_3$'s perimeter. We know that the current overlap region is the final overlap region.

**Algorithm 2**: MaxFirst - Phase 2

**input** : An optimal sub-region
**output**: The complete optimal region

```
1  r := the center of Q
2  H := ∅          /* a heap containing NLCs using distance as key */
3  use Q to issue a region query on the R-tree of all the NLCs to find the NLCs
   that cover Q
4  foreach NLC c in Q.C do
5      d := shortest distance from r to the perimeter of c
6      insert entry (c, d) to H
7  remove entry (c1, d1) from H
8  remove entry (c2, d2) from H
9  R := overlap of c1 and c2
10 dmax := the maximal distance from r to the perimeter of R
11 while H is not empty do
12     remove entry (c, d) from H
13     if d < dmax then
14         R := overlap of R and c
15         dmax := the maximum distance from r to the perimeter of R
16     else
17         return R
18 return R
```

Algorithm 2 shows the details of MaxFirst's second phase. Lines 1-5 set $r$ to the center of $Q$, and use a heap to order the NLCs based on the shortest distances from their perimeters to $r$. Lines 6-8 compute an overlap region $R$ using the first two NLCs taken from the heap. Line 9 determines the largest distance from $r$ to the perimeter of $R$, denoted by $d_{max}$. We use the NLCs one by one to clip the overlap region $R$ and at the same time update $d_{max}$, until the shortest distance from $r$ to a

NLC is larger than $d_{max}$. The perimeter of the remaining NLCs will not intersect $R$, so $R$ is the final overlap region.

Note the shortest distance from a NLC's perimeter to a point $r$ inside the NLC can be computed in constant time. $d_{max}$, the maximum distance from $r$ to the perimeter of the overlap region $R$, can also be computed efficiently. Also note that the choice of $r$ does not affect the correctness of the algorithm as long as $r$ is a point inside $Q$ which is known to be a part of the complete overlap region.

## 3.4 Complexity Analysis

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 solved in $O(log|\mathcal{P}|)$ using an index.

In MaxFirst's Phase 1 (Algorithm 1), we recursively partition the space to find the set of optimal sub-regions.Let the minimum area of the partitioned region is $A$. Then the maximum number of quadrants that can be formed for a data space of area $S$ is a constant $n = S/A$. For each quadrant, we perform a range query tofind all the NLCs that overlap with it. In the literature, the range query can be executed in $O(k + log|\mathcal{O}|)$ time where $k$ is the greatest result size of a range query. In other words, Phase 1 requires $O(nk+nlog|\mathcal{O}|)$ time. Since $n$ is a constant, the complexity of Phase 1 is $O(k + log|\mathcal{O}|)$

Having found the set of optimal sub-regions, Phase 2 (Algorithm 2) re-constructs the complete optimal regions.This involves finding theintersection of the NLCs that cover the optimal sub-regions found in Phase 1.Since $k$ is the greatest result size of the range query, in other words, $k$ is the maximum number of NLCs that cover the optimal sub-region. Hence this step requires $O(k)$.Hence, the overall running time of algorithm MaxFirst is $O(|\mathcal{O}|log|\mathcal{P}| + log|\mathcal{O}| + k)$.

Chapter 4

# Generalization to MaxBRkNN

We generalize the MaxBRNN problem to the MaxBRkNN problem and show that our MaxFirst algorithm can also be used to solve the MaxBRkNN problem. The basic assumption in the MaxBRNN problem is that each customer only goes to his/her nearest service site. Wong et al. [55] generalize this to the MaxBRNN problem where each customer is equally likely to go to his/her $k$ nearest service sites.

However, in reality, a customer tends to have different preferences for different service sites. We define an *interest model* to captures the probability of customer $o$ going to $o$'s $i$th ($1 \leq i \leq k$) nearest neighbor in $\mathcal{P}$, denoted as $pr_i$, $\sum_i pr_i = 1$. For example, if $\mathcal{O}$ is a set of residents and $\mathcal{P}$ is a set of convenient stores, we may have an interest model where $k = 3$ and $pr_1 = 0.6$, $pr_2 = 0.3$, $pr_3 = 0.1$.

Based on the interest model, we define the MaxBRkNN problem as follows. Given a set $\mathcal{O}$ of customer objects, a set $\mathcal{P}$ of service sites, and an interest model

$M$, the MaxBRkNN problem is to find the optimal regions such that setting up a new service site $q$ in an optimal region $q$ will attract the maximum number of customers. Note that MaxBRNN is a special case of MaxBRkNN where $k = 1$.

Recall that the NLC of an object $o \in \mathcal{O}$ is the circle where $o$ is the center and the distance from $o$ to its nearest neighbor in $\mathcal{P}$ is the radius. When $k = 1$, the NLC is the region where $o$ will be interested if a new service site is set up there. When $k > 1$, the region is the circle where $o$ is the center and the distance from $o$ to its $k$***th*** nearest neighbor in $\mathcal{P}$ is the radius. However when $k > 1$, the location of the new service site in the circle determines how frequent (i.e. the probability) $o$ will go to the service site.

Let us define the $i$th NLC of an object $o \in \mathcal{O}$, denoted as $c_i$, as a circle whose center is $o$ and radius is the distance from $o$ to its $i$th nearest neighbor in $\mathcal{P}$. If a new service site is set up in $c_1$, the probability that $o$ goes to it is $pr_1$, and if the new service site is set up in the annulus formed by $c_{i-1}$ and $c_i$, the probability that $o$ goes to it is $pr_i$.

Figure 4.1 shows an example where $k = 3$. The different shades indicate the different probabilities that $o$ goes to a new service site in it.

Recall that our MaxFirst algorithm works with NLCs and a point (or region) gets the scores from the NLCs that cover it. To make MaxFirst applicable to the MaxBRkNN problem, we only need to assign the proper scores to the $c_i$s ($i \leq k$) so that a point in an annulus gets the right score from the NLCs that cover it.

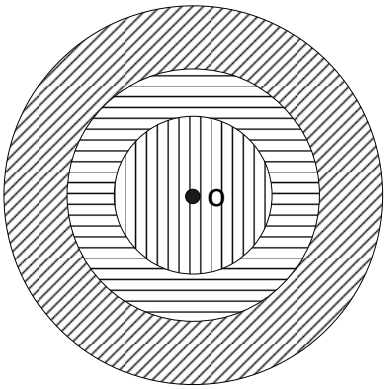



Figure 4.1: An object has $k$ NLCs in MaxBRkNN.

Since $c_{i+1}$, $c_{i+2}$, ..., $c_k$ all cover $c_i$, a point in the annulus formed by $c_{i-1}$ and $c_i$ gets scores from $c_i$, $c_{i+1}$, ..., $c_k$. A proper score assignment to $NLC$s of $o$, therefore, must satisfy the condition: $\sum_{i \leq j \leq k} score(c_j) = pr_i * w(o)$ where $w(o)$ is the weight of $o$ and $score(c_j)$ is the score of $c_j$.

We assign $(pr_i - pr_{i+1}) * w(o)$ as the score of $c_i$. We can verify that $\sum_{i \leq j \leq k} score(c_j) = pr_i * w(o)$. For example, if $k = 3$, and $pr_1 = 0.6$, $pr_2 = 0.3$, $pr_3 = 0.1$, the score of $c_1$, $c_2$, and $c_3$ will be $0.3 * w(o)$, $0.2 * w(o)$, and $0.1 * w(o)$, respectively.

With this score assignment method, the MaxBRkNN problem can be solved using the MaxFirst algorithm. For each object $o$ in $\mathcal{O}$, we compute its NLCs $c_1$, $c_2$, ..., and $c_k$, and assign the proper scores to them. Then we can run the MaxFirst algorithm to get the optimal regions.

Note that the MaxOverlap algorithm in [55] has an implicit assumption that each NLC must intersect one of the other NLCs. Due to this assumption, the MaxOverlap algorithm cannot be used immediately to solve the more general MaxBRkNN problem that we defined, because the $k$ NLCs of an object are homocentric and they do not intersect.

Chapter 5

# Performance Study

We conducted extensive experiments to study the performance of our algorithm MaxFirst. Since MaxOverlap is the state-of-the-art algorithm for the MaxBRNN problem and [55] has shown that it outperforms other existing algorithms [10, 15], we compare MaxFirst with it. We implemented MaxFirst in C++, and used the original C++ implementation of MaxOverlap that we get from the authors of [55]. All experiments are done 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 MaxBRNN problem (and MaxBRkNN problem) under various settings. Since both MaxOverlap and MaxFirst need to compute the NLCs for all the consumer objects, we exclude the time spent on computing NLCs from their running times. Note that it only takes about one minute to compute and index the NLCs, this cost does not affect the relative performances of the algorithms. We investigate the scalability of the algorithms with respect to the number of objects in the consumer

Table 5.1: Parameter settings

| Parameter | Default | Range |
|---|---|---|
| $k$ | 1 | 1-4 |
| Number of consumer objects, $\|\mathcal{O}\|$ | 50K | 10-100K |
| Number of service sites, $\|\mathcal{P}\|$ | 500 | 100-1K |

Table 5.2: Summary of real datasets

| Dataset | Cardinality |
|---|---|
| UX | 19499 |
| NE | 123,593 |

dataset, the number of objects in the service sites dataset, and the value of $k$ (for MaxBRkNN problem). Table 5.1 lists the parameters and their values.

Both real world data and synthetic data are used in the experiments. Table 5.2 lists the details of the real world datasets (downloaded from http://www.rtreeportal.org/spatial.h UX contains points of populated places and cultural landmarks in US and Mexico; NE contains points representing the geographical locations in North East America. We generated synthetic data in uniform distribution and normal distribution. In each set of experiments, the customer dataset and the service site dataset have the same distribution. See Table 5.1 for the sizes of the synthetic datasets. In the experiments we make the size of $\mathcal{P}$ smaller than the size of $\mathcal{O}$, because in reality the number of service sites (e.g., gas stations) is always much smaller than the number of consumer objects (e.g., vehicles). We find that the weights of the consumer objects do not affect the relative performance of the algorithms, so we only show the experiments where the weight of the consumer objects is set to 1.

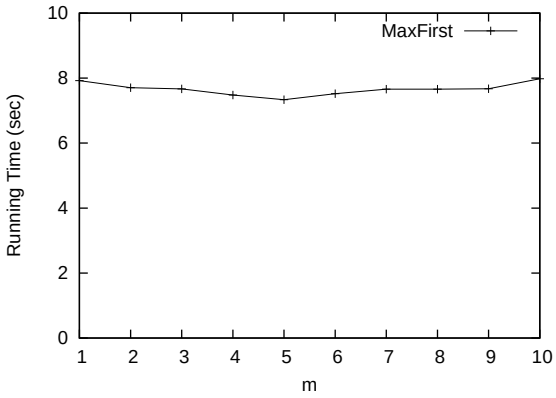



Figure 5.1: Effect of $m$, normal distribution.

## 5.1 Effect of m on MaxFirst

We first carry out experiments to study the effect of parameter $m$ on MaxFirst's performance. Figure 5.1 shows the result on the default synthetic datasets with uniform distribution. The results we obtain for other datasets are similar.

We observe that $m$ has little effect on the performance of MaxFirst. The runtime of MaxFirst first decreases and then increases as the value of $m$ increases, but the change is small. When $m$ is small (e.g., 2), we have the overhead of frequently checking whether the NLCs intersect at a point, and when $m$ is large (e.g., 7), we will split a region continuously resulting in many sub-regions. The nice thing is that the effect of $m$ is small and it is safe to assign any small value to it. This is expected because the probability that many NLCs intersect at an intersection point is low. For the rest of the experiments, we set $m$ to 4.

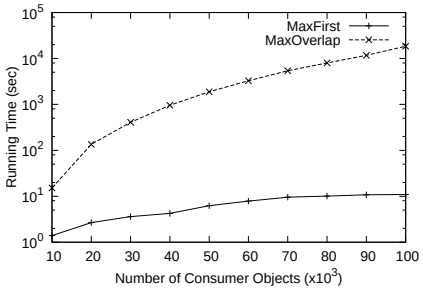



Figure 5.2: Effect of $|\mathcal{O}|$, uniform distribution.

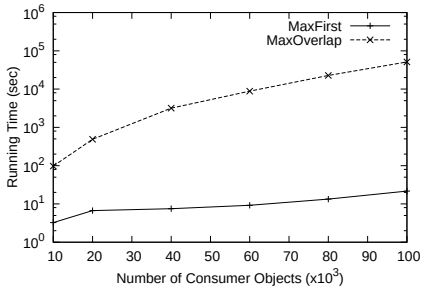



Figure 5.3: Effect of $|\mathcal{O}|$, normal distribution.

## 5.2 Effect of the Number of Consumer Objects

Next, we study the effect of $\mathcal{O}$ on the performance of the algorithms. We fix the number of service sites $\mathcal{P}$ at 500, and vary the number of customer objects $|\mathcal{O}|$ from 10K to 100K. Figures 5.2 and 5.3 show the algorithms' performance on datasets for 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 consumer objects is large. As the number of consumer objects increases, the running times of both the algorithms increase, but the running time of MaxFirst increases very slowly while the running time of MaxOverlap increases rapidly. MaxFirst is much more scalable with the number of consumer objects because MaxFirst only partitions the regions that potentially contains a part of an optimal region. Intuitively, MaxFirst only partitions the region where the density of NLCs is the highest. Although the number of NLCs increases with the number of consumer objects, the number of

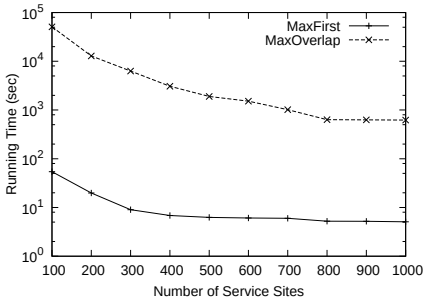



Figure 5.4: Effect of $|\mathcal{P}|$, uniform distribution.

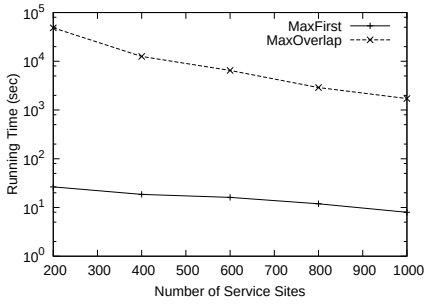



Figure 5.5: Effect of $|\mathcal{P}|$, normal distribution.

regions where the density of NLC is the highest will not increase, and the size of such regions will not increases. MaxOverlap does not scale well with the number of consumer objects because it needs to compute all the intersection oints of every pair of NLCs. As the number of NLCs increases, there will be a lot more intersection points.

Comparing Figures 5.2 and 5.3, we observe that data distribution affects the algorithms' performances. Both algorithms spend more time on datasets with normal distribution. For MaxFirst, a normal distribution means that there will be more NLCs in the region with the highest density of NLCs. For MaxOverlap, a normal distribution means that there will be more intersections points in the dense area.

## 5.3 Effect of the Number of Service Sites

To 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 5.4 and 5.5 show the algorithms'

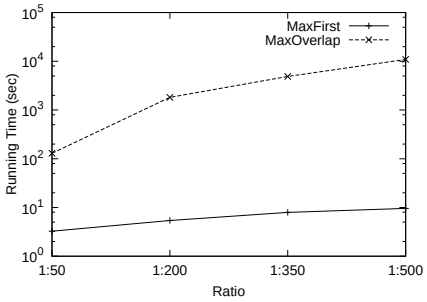



Figure 5.6: Effect of $|\mathcal{P}|/|\mathcal{O}|$, UX dataset.

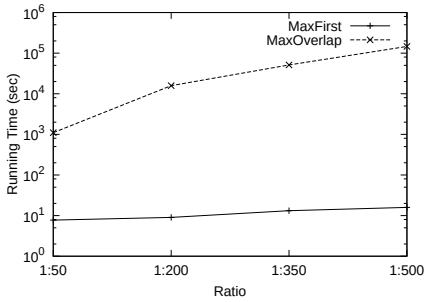



Figure 5.7: Effect of $|\mathcal{P}|/|\mathcal{O}|$, NE dataset.

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 ($|\mathcal{P}|$) increases. When there are more services sites, the NLCs become smaller. This means that the density of NLCs at the region with the highest density will be lower. This is why 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 is the reason the processing of MaxOverlap decreases as $|\mathcal{P}|$ increases.

## 5.4 Results on Real World Datasets

We have seen that both the number of service sites and the number of consumer objects affect the time needed by the algorithms to solve the MaxBRNN problem. Here 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 divide the objects into two parts based on a certain ratio, and take one part as the $\mathcal{P}$ set and the other

part as the $\mathcal{O}$ set, then run the algorithms on them.

Figures 5.6 and 5.7 show the runtimes 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.

Finally, we study the effect of $k$ on the algorithms' performances in solving the general MaxBRkNN problems. Figure 5.8 shows the results on the MaxBRkNN problem where the probabilities in the interest model are the same. The default synthetic datasets with uniform distribution are used. 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 does. 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.

### 5.4.1 Results on MaxBRkNN Problem

Figure 5.9 shows the performance of MaxFirst on the more general MaxBRkNN problem where the probabilities in the interest model are not same. Note that this figure is not plotted in log-scale. There is only one line in the graph as MaxOverlap

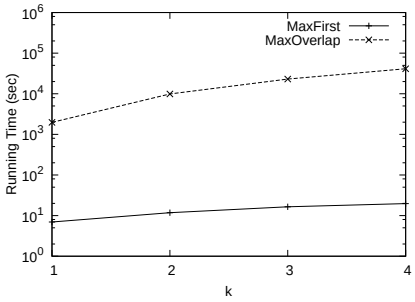



Figure 5.8: Effect of $k$, same probabilities.

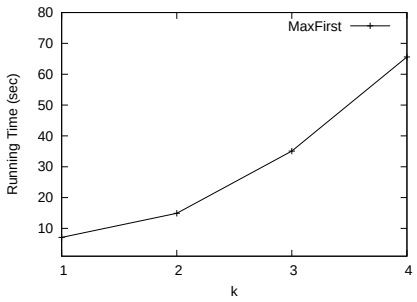



Figure 5.9: Effect of $k$, different probabilities.

cannot be applied to such MaxBRkNN problems. As $k$ increases, there are more NLCs, and the density at the densest region will also be higher, hence it takes MaxFirst more time to find the optimal regions.

Chapter 6

# Conclusion

In this thesis, we have presented an efficient solution for the MaxBRNN problem to find an optimal region where adding a new service site can attract the maximal number of customers. Our algorithm, MaxFirst, solves a MaxBRNN (and a more general MaxBRkNN) problem in two steps. In the first step, MaxFirst finds a small region that is a part of the optimal region by partitioning the space into sub-regions and searches only in promising sub-regions. In the second step, MaxFirst computes the whole optimal region using the information gathered in the first step. Experimental results show that MaxFirst is much more efficient than existing algorithms. Furthermore, MaxFirst scales very well with data sizes, and performs consistently well under various settings.

# Bibliography

[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] Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a sentient computing system. *Computer*, 34(8):50–56, Aug. 2001.

[3] Pankaj K. Agarwal, Lars Arge, and Jeff Erickson. Indexing moving points. In *PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 175–186, New York, NY, USA, 2000. ACM.

[4] Pankaj K. Agarwal, Leonidas J. Guibas, Herbert Edelsbrunner, Jeff Erickson, Michael Isard, Sariel Har-Peled, John Hershberger, Christian Jensen, Lydia Kavraki, Patrice Koehl, Ming Lin, Dinesh Manocha, Dimitris Metaxas, Brian Mirtich, David Mount, S. Muthukrishnan, Dinesh Pai, Elisha Sacks, Jack

Snoeyink, Subhash Suri, and Ouri Wolefson. Algorithmic issues in modeling motion. *ACM Comput. Surv.*, 34(4):550–572, 2002.

[5] 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.

[6] Paramvir Bahl and Venkata N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM*, pages 775–784, 2000.

[7] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal*, 15(3):229–249, 2006.

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

[9] Alastair R. Beresford and Frank Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

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

[11] 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.

[12] Su Chen, Beng Chin Ooi, Kian-Lee Tan, and Mario A. Nascimento. St2b-tree: a self-tunable spatio-temporal b+-tree index for moving objects. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 29–42, New York, NY, USA, 2008. ACM.

[13] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Querying imprecise data in moving object environments. *IEEE Trans. Knowl. Data Eng.*, 16(9):1112–1127, 2004.

[14] Alminas Civilis and Stardas Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Trans. on Knowl. and Data Eng.*, 17(5):698–712, 2005. Senior Member-Christian S. Jensen.

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

[16] Martin Erwig, Ralf Hartmut Güting, Markus Schneider, and Michalis Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *Geoinformatica*, 3(3):269–296, 1999.

[17] Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. A data model and data structures for moving objects databases. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 319–330, New York, NY, USA, 2000. ACM.

[18] Hartmut Guting, Teixeira de Almeida, and Zhiming Ding. Modeling and querying moving objects in networks. *The VLDB Journal*, 15(2):165–190, 2006.

[19] Ralf Hartmut Guting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.

[20] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[21] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[22] Gisli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[23] Gisli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24, 1999.

[24] Kathleen Hornsby and Max J. Egenhofer. Modeling moving objects over multiple granularities. *Annals of Mathematics and Artificial Intelligence*, 36(1-2):177–194, 2002.

[25] Haibo Hu, Jianliang Xu, and Dik Lun Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*, pages 479–490, Baltimore, Maryland, 2005. ACM Press.

[26] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. Cartel: a

distributed mobile sensor computing system. In *SenSys*, pages 125–138, New York, NY, USA, 2006. ACM.

[27] Christian S. Jensen and Stardas Pakalnis. Trax: real-world tracking of moving objects. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1362–1365. VLDB Endowment, 2007.

[28] Dmitri V. Kalashnikov, Sunil Prabhakar, Susanne E. Hambrusch, and Walid G. Aref. Efficient evaluation of continuous range queries on moving objects. In *DEXA '02: Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 731–740, London, UK, 2002. Springer-Verlag.

[29] 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.

[30] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 201–212. ACM Press, Dallas, Texas, United States, 2000.

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

[32] Kyriakos Mouratidis, Dimitris Papadias, Spiridon Bakiras, and Yufei Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors.

*IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 17(11):1451– 1464, 2005.

[33] Ginger Myles, Adrian Friday, and Nigel Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2(1):56–64, 2003.

[34] M.Goodrich N.M.Amato and E.A.Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In *Discreat Algorithms*. ACM Press, 2000.

[35] Dieter Pfoser and Christian S. Jensen. Capturing the uncertainty of moving-object representations. In *SSD '99: Proceedings of the 6th International Symposium on Advances in Spatial Databases*, pages 111–132, London, UK, 1999. Springer-Verlag.

[36] Kriengkrai Porkaew, Iosif Lazaridis, and Sharad Mehrotra. Querying mobile objects in spatio-temporal databases. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 59–78, London, UK, 2001. Springer-Verlag.

[37] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43, New York, NY, USA, 2000. ACM.

[38] Bharat Rao and Louis Minakakis. Evolution of mobile location-based services. *Commun. ACM*, 46(12):61–65, 2003.

[39] Philippe Rigaux, Michel O. Scholl, and Agnes Voisard. *Spatial databases with application to GIS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[40] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79. ACM, San Jose, California, United States, 1995.

[41] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 331–342. ACM Press, Dallas, Texas, United States, 2000. TPR-tree.

[42] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16, 1984.

[43] Jochen Schiller and Agnès Voisard. *Location Based Services*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[44] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 154–165. ACM Press, 1998.

[45] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and querying moving objects. In *ICDE '97: Proceedings of the Thirteenth International Conference on Data Engineering*, pages 422–432, Washington, DC, USA, 1997. IEEE Computer Society.

[46] Zhexuan Song and Nick Roussopoulos. Hashing moving objects. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 161–172, London, UK, 2001. Springer-Verlag.

[47] 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.

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

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

[50] Yannis Theodoridis, Timos K. Sellis, Apostolos Papadopoulos, and Yannis Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *SSDBM '98: Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 123–132, Washington, DC, USA, 1998. IEEE Computer Society.

[51] Dalia Tiesyte and Christian S. Jensen. Challenges in the tracking and prediction of scheduled-vehicle journeys. In *PERCOMW '07: Proceedings of the Fifth*

*IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 407–412, Washington, DC, USA, 2007. IEEE Computer Society.

[52] Goce Trajcevski, Ouri Wolfson, Klaus Hinrichs, and Sam Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems*, pages 463 – 507, 2004.

[53] Ouri Wolfson, Liqin Jiang, A. Prasad Sistla, Sam Chamberlain, Naphtali Rishe, and Minglin Deng. Databases for tracking mobile units in real time. In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 169–186, London, UK, 1999. Springer-Verlag.

[54] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: issues and solutions. In *Proc. Tenth International Conference on Scientific and Statistical Database Management*, pages 111–122, 1–3 July 1998.

[55] 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.

[56] 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*, 2(1):1126–1137, 2009.

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

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

[59] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee. Location-based spatial queries. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 443–454, New York, NY, USA, 2003. ACM.