Reverse Thinking in Spatial Queries

by

Shenlu Wang

B.CS. RMIT University, 2012
H.CS. University of New South Wales, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SCHOOL
OF
COMPUTER SCIENCE AND ENGINEERING

UNSW
AUSTRALIA

Wednesday 31st January, 2018

All rights reserved.
This work may not be reproduced in whole or in part,
by photocopy or other means, without the permission of the author.

© Shenlu Wang 2018
In recent years, an increasing number of researches are conducted on spatial queries regarding the influence of query objects. Among these queries, reverse k nearest neighbors (RkNN) query is the one studied the most extensively. Reverse k furthest neighbors (RkFN) queries is the natural complement of RkNN queries. RkNN query is introduced to reflect the influence of the query object. Since this representation is intuitive, RkNN query has attracted significant attention among the database community. Later, reverse top-k queries was introduced, and also used extensively to represent influence. In many scenarios, when we consider the influence of a spatial object, reverse thinking is involved. That is, whether an object is influential to another object is depending on how the other object assesses this object, other than how this object considers the other object. In this thesis, we study three problems involves reverse thinking.

We first study the problem of efficiently computing RkFN queries. We are the first to propose a solution for arbitrary value of k. Based on several interesting observations, we present an efficient algorithm to process the RkFN queries. We also present a rigorous theoretical analysis to study various important aspects of the problem and our algorithm. An extensive experimental study demonstrates that our algorithm outperforms the state-of-the-art algorithm even for k=1. The accuracy of our theoretical analysis is also verified.

We then study the problem of selecting set of representative products considering both diversity and coverage based on reverse top-k queries. Since this problem is NP-hard, we employ a greedy algorithm. We adopt MinHash and KMV Synopses to assist set operations. Our experimental study demonstrates the performance of the proposed algorithm.

We also study the problem of maximizing spatial influence of facility bundle based on RkNN queries. We are the first to study this problem. We prove its NP-hardness, and propose a branch-and-bound best first search algorithm that greedily selects the currently best facility until we get the required number of facilities. We introduce the concept of kNN region. It allows us to avoid redundant calculation with dynamic programming technique. Experiments show that our algorithm is orders of magnitudes better than our baseline algorithm.
Originality Statement

‘I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project’s design and conception or in style, presentation and linguistic expression is acknowledged.’

[Signature]

Signed  ...............................................
Date  ...............................................

31.01.2018
Abstract

In recent years, an increasing number of researches are conducted on spatial queries regarding the influence of query objects. Among these queries, reverse $k$ nearest neighbors (R$k$NN) query is the one studied the most extensively. Reverse $k$ furthest neighbors (R$k$FN) queries is the natural complement of R$k$NN queries. R$k$NN query is introduced to reflect the influence of the query object. Since this representation is intuitive, R$k$NN query has attracted significant attention among the database community. Later, reverse top-$k$ queries was introduced, and also used extensively to represent influence. In many scenarios, when we consider the influence of an spatial object, reverse thinking is involved. That is, whether an object is influential to another object is depending on how the other object assess this object, other than how this object considers the other object. In this thesis, we study three problems involves reverse thinking.

We first study the problem of efficiently computing R$k$FN queries. We are the first to propose a solution for arbitrary value of $k$. Based on several interesting observations, we present an efficient algorithm to process the R$k$FN queries. We also present a rigorous theoretical analysis to study various important aspects of the problem and our algorithm. An extensive experimental study demonstrates that our algorithm outperforms the state-of-the-art algorithm even for $k = 1$. The accuracy of our theoretical analysis is also verified.
We then study the problem of selecting set of representative products considering both diversity and coverage based on reverse top-$k$ queries. Since this problem is NP-hard, we employ a greedy algorithm. We adopt MinHash and KMV Synopses to assist set operations. Our experimental study demonstrates the performance of the proposed algorithm.

We also study the problem of maximizing spatial influence of facility bundle based on $RkNN$ queries. We are the first to study this problem. We prove its NP-hardness, and propose a branch-and-bound best first search algorithm that greedily select the currently best facility until we get the required number of facilities. We introduce the concept of $kNN$ region. It allows us to avoid redundant calculation with dynamic programming technique. Experiments show that our algorithm is orders of magnitudes better than our baseline algorithm.
Publications Involved in Thesis

- Shenlu Wang, Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Dongxi Liu. Efficiently Computing Reverse $k$ Furthest Neighbors, in ICDE, 2016. (Chapter 3)

- Shenlu Wang, Muhammad Aamir Cheema, Ying Zhang, Xuemin Lin. Selecting Representative Objects Considering Coverage and Diversity, in GeoRich@SIGMOD, 2015.(Chapter 4)

- Shenlu Wang, Ying Zhang, Xuemin Lin, Muhammad Aamir Cheema. Maximize Spatial Influence of Facility Bundle Considering Reverse $k$ Nearest Neighbors, submitted to DASFAA, 2018.(Chapter 5)
Acknowledgements

First and foremost, I would like to express my sincere gratitude to Prof. Xuemin Lin for supervising me during the candidature. I really appreciate his insightful advices and suggestions during my Ph.D. study. Moreover, I am deeply impressed by his insight and breadth of knowledge in his research area as well as his continuous dedication to research work. Without his significant encouragements and supports, my research years will never be able to run smooth.

Secondly, I would like to express my heartfelt gratitude to my joint supervisor, Dr. Muhammad Aamir Cheema for his constant encouragement and guidance. His patient and humor just like the light in dark which guide me to pass through any difficulties. He has walked me through all the stages of the writing of this thesis. Without his consistent and illuminating instruction, this thesis could not have reached its present form.

Thirdly, parts of the work in this thesis were conducted in collaboration with Prof. Ying Zhang. I thank him for supporting the work presented in this thesis, and much more.

Besides, special thanks go to my group members. The time we spent together will be memorized forever.

Last but not least, I am very thankful to my parents for their constant love, support
and encouragement, and for being with me always in my life.
# Contents

Abstract v

Publications vii

Acknowledgements ix

List of Figures xv

List of Tables xviii

List of Algorithms xix

1 Introduction 1

1.1 Motivations ................................. 3

1.1.1 Efficiently Computing Reverse $k$ Furthest Neighbors . . . . . 4

1.1.2 Selecting Set of Representative Objects .......................... 6

1.1.3 Maximize Spatial Influence of Facility Bundle .................... 9

1.2 Contributions ............................... 12

1.2.1 Efficiently Computing Reverse $k$ Furthest Neighbors . . . . . 12

1.2.2 Selecting Set of Representative Objects .......................... 13

1.2.3 Maximize Spatial Influence of Facility Bundle .................... 13

1.3 Thesis Organization .......................... 13
# Related Work

## 2.1 Reverse k Furthest Neighbors Queries

- 2.1.1 CHFC
- 2.1.2 SkyRider

## 2.2 Top-k and Reverse Top-k Queries

- 2.2.1 Reverse top-k threshold algorithm
- 2.2.2 Branch-and-Bound algorithm for reverse top-k
- 2.2.3 Top-k most favorite products maximizing coverage
- 2.2.4 Finding the most diverse products

## 2.3 Reverse k Nearest Neighbors Queries

- 2.3.1 Influence zone
- 2.3.2 Slice
- 2.3.3 R$k$NN based top-n

# Efficiently Computing Reverse k Furthest Neighbors

## 3.1 Preliminaries

## 3.2 Overview

## 3.3 Techniques

- 3.3.1 Determine futile query
- 3.3.2 Pruning the search space
- 3.3.3 Verification

## 3.4 Theoretical Analysis

- 3.4.1 Expected area of k-depth contour
- 3.4.2 Expected area of sweeping region

## 3.5 Experiments

- 3.5.1 Experimental settings
- 3.5.2 Shallow facilities
3.5.3 Performance evaluation ........................................ 68
3.5.4 Theoretical analysis evaluation .............................. 74

4 Selecting Set of Representative Objects .................. 77
  4.1 Preliminaries .................................................... 78
    4.1.1 Top-k and reverse top-k queries ....................... 78
    4.1.2 MinHash ................................................... 79
    4.1.3 KMV synopsis .............................................. 79
  4.2 Problem Definition ........................................... 80
  4.3 Algorithms and Techniques ................................ 82
    4.3.1 Compute reverse top-k .................................. 83
    4.3.2 Select representative products ......................... 84
  4.4 Theoretical Analysis ....................................... 84
  4.5 Experiments .................................................. 86
    4.5.1 Effect of |W| ............................................. 87

5 Maximize Spatial Influence of Facility Bundle .......... 94
  5.1 Preliminaries .................................................. 95
    5.1.1 Problem Definition ..................................... 95
    5.1.2 NP-hardness .............................................. 96
    5.1.3 Baseline Algorithm .................................... 97
  5.2 Techniques ................................................... 98
    5.2.1 Solution Overview ..................................... 98
    5.2.2 kNN Region ............................................. 101
    5.2.3 Inherit Candidate RkNN Set .......................... 103
    5.2.4 Maximum Possible Contribution and Contribution .... 106
  5.3 Experiments .................................................. 108

xiii
## List of Figures

1.1 Gas stations near the University of New South Wales .............. 2
1.2 $u_1$ and $u_2$ are R$k$FN ($k = 1$) of $q$ ............................... 4
1.3 $k = 2, t = 2$ .................................................. 7
1.4 $k = 1, t = 2$ .................................................. 7
1.5 MB-R$k$NN .................................................. 11

2.1 CHFC .................................................. 18
2.2 Objects in primal ........................................ 21
2.3 Objects in dual ........................................ 21

3.1 Quadrant test ........................................... 43
3.2 P-ray pruning ($k = 2$) ................................ 43
3.3 Location depth and its relationship with R$k$FN .................... 45
3.4 Half-space pruning ....................................... 49
3.5 $k$-depth contour ($k = 2$) ................................ 49
3.6 Pruning using $k$-depth contour ................................ 51
3.7 Pruning using shallow facilities ................................ 55
3.8 Space pruned ............................................. 56
3.9 Pruner region ............................................. 56
3.10 Pruning sweeping region .................................... 57
3.11 Verification phase .................................................. 57
3.12 Improving verification using shadows ......................... 59
3.13 Area of k-depth contour ....................................... 62
3.14 Expected area of the sweeping region .......................... 64
3.15 Number of shallow facilities .................................. 67
3.16 Real data set ....................................................... 68
3.17 Synthetic data set ............................................... 69
3.18 Real data set ....................................................... 69
3.19 Synthetic data set ............................................... 70
3.20 Non-futile queries (effect of k) ................................. 71
3.21 Non-futile queries (effect of data set) ......................... 72
3.22 Effectiveness of pruning techniques ............................ 72
3.23 Effectiveness of shadows ....................................... 73
3.24 Area of k-depth contour ....................................... 75
3.25 Area of sweeping region (varying k) .......................... 75

4.1 Effect of |W| (in 1000) (small uniform) ......................... 87
4.2 Effect of |W| (in 1000) (large anti-correlated) ................... 88
4.3 Effect of d (small anti-correlated) .............................. 89
4.4 Effect of k (house) ............................................... 90
4.5 Effect of t (small uniform) ..................................... 90
4.6 Effect of t (small anti-correlated) .............................. 91
4.7 Effect of α ....................................................... 91
4.8 Percentage of phase 2 time ................................... 92

5.1 kNN Region of a Line Segment .................................. 102
5.2 kNN region of rectangle ......................................... 103
5.3 Lemma 5.3 ......................................................... 103
5.4 Lemma 5.7 ......................................................... 107
5.5 Effect of $|F| (|U| = |F| \ast 2)$ ...................................... 109
5.6 Effect of $|U|$ ....................................................... 110
5.7 Effect of $k$ ....................................................... 111
5.8 Effect of $t$ ....................................................... 111
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Figure 1.3</td>
<td>7</td>
</tr>
<tr>
<td>1.2</td>
<td>Figure 1.4</td>
<td>7</td>
</tr>
<tr>
<td>4.1</td>
<td>Experimental settings</td>
<td>87</td>
</tr>
<tr>
<td>4.2</td>
<td>Number of page requests</td>
<td>88</td>
</tr>
<tr>
<td>5.1</td>
<td>Experimental settings</td>
<td>109</td>
</tr>
</tbody>
</table>
# List of Algorithms

1. CHFC ......................................................... 19
2. RTA : Reverse top-$k$ Threshold Algorithm .............. 24
3. Influence Zone .............................................. 33
4. Candidate Polygon ......................................... 53
5. Verification ................................................... 60
6. $MB-R_kNN(F^R, U^R, t)$ ................................... 100
7. $inherit(F_c, F_c^e)$ ........................................... 105
Chapter 1

Introduction

Spatial database has been an active research area for more than four decades, studying the problems of management and query needs for spatial applications such as Geographic Information Systems (GIS). Other applications include Computer Aided Design (CAD), Very-Large-Scale Integration (VLSI) design, Multimedia Information System (MMIS), Data Warehousing (DWH), and NASA’s Earth Observation System (EOS). GIS is accepted as an important tool in governmental decision-making, and military planners have embraced GIS technology at all levels of tactical, operational and strategic planning. The current research is aiming at improving the functionality and performance of spatial database management systems.

Figure 1.1 illustrates an example application of spatial database. The figure is obtained by entering a query of “Gas stations near the University of New South Wales” using Google Maps. Gas stations are represented as points labeled with balloons alongside the road network which is represented by lines. The university of New South Wales is represented by a polygon filled with grey color.

In recent years, spatial queries that involves reverse thinking have gained in-
creasing research interests. For example, reverse k nearest neighbors (RkNN) queries has been extensively studied. The concept of RkNN query is first introduced to reflect the influence of the query object. Consider a bichromatic data set where we have a set of points representing facilities and a set of points representing users. To find out the users a facility influences the most, we should consider the users for whom the facility is one of their k nearest facilities, not the k nearest users of the facility. This is intuitive because the k nearest users of the facility may never go to the facility if it is not one of their k nearest facilities, but the users for whom the facility is one of their k nearest facilities are potential users of the facility even if they are far away from the facility, and these users are the reverse k nearest users of the facility. In other words, a facility wants to find out its influenced users should not consider the users close to it but the users who consider it as a nearby facility. And the difference between kNN queries and RkNN queries demonstrated in this example is the meaning of reverse thinking to be employed here.

Similar applies to reverse k furthest neighbors (RkFN) queries. RkFN queries is
the natural complement of R$k$NN queries. To find out the users a facility influences the least, we should consider the users for whom the facility is one of their $k$ furthest facilities, not the $k$ furthest users of the facility. Another scenario is related to representative products. Here, we have a set of points representing products and a set of weighting vectors representing user preferences. To find out the users a product attracts the most, we should consider the users for whom the facility is one of their top-$k$ ranked products.

The rest of this chapter is organized as follows. In Section 1.1 we first motivate the problems and unveil the challenges to each of them. Then we list the contributions of our research to address these challenges in Section 1.2. Last, we illustrate an overall structure of this thesis in Section 1.3.

1.1 Motivations

This thesis focuses on developing efficient solutions to advance algorithmic efficiency on query processing for several queries involves reverse thinking. Specifically, we study the following three problems.

1. Efficiently computing reverse $k$ furthest neighbors (R$k$FN) queries.

2. Selecting set of representative products considering both diversity and coverage based on reverse top-$k$ queries.

3. Maximizing spatial influence of facility bundle based on reverse $k$ nearest neighbors (R$k$NN) queries.
1.1.1 Efficiently Computing Reverse $k$ Furthest Neighbors

Given a set of facilities $F$, a user $u$ is said to be influenced by a query facility $q$ if $q$ is one of the $k$ nearest facilities of $u$. This is because users usually prefer nearby facilities and are more likely to be influenced by the advertisements sent from these facilities. Motivated by this, a reverse $k$ nearest neighbor (R$k$NN) query $q$ returns every user $u$ who is most influenced by $q$, i.e., $q$ is one of the $k$ closest facilities of $u$. R$k$NN query has been extensively studied [AKK+09, CLZZ11, KMS+07, LNY03, SS10, SRAE01, TPLX07, WYCT08, YL01, YCLZ14] ever since it was introduced in [KM00]. Also of interest is to find the users that are least influenced: the reverse $k$-furthest neighbors (R$k$FN) query has also received significant attention [KJG08, LCFK10, LCFK12, TTS09, YLK09]. Given a set of facilities $F$, a set of users $U$ and a query facility $q$, a R$k$FN query returns every user $u$ for which $q$ is one of its $k$-furthest facilities.

We illustrate R$k$FN and R$k$NN queries using the example in Fig. 1.2 that shows three facilities (dots) and four users (triangles). Assuming $k = 1$, R1FN of $q$ are the users $u_1$ and $u_2$ because $q$ is the furthest facility for $u_1$ and $u_2$. Note that although $u_3$ is the furthest user from $q$, it is not R1FN of $q$ because $q$ is not its furthest facility. In fact, as $q$ is the closest facility for $u_3$, $u_3$ is the R1NN of $q$, and is highly influenced by $q$. $u_4$ is also the R1NN of $q$.

![Figure 1.2](image)

Figure 1.2: $u_1$ and $u_2$ are R$k$FN ($k = 1$) of $q$

Similar to R$k$NN query, R$k$FN query also has numerous applications in location-
based services, marketing, resource allocation, clustering, recommendation systems, profile-based management and graphics rendering etc.

**Example 1.** BotFighters is a pervasive location-based mobile game that is designed to be a MMORPG (massively multiplayer online role-playing game) played in an urban environment. The mission of the game is to locate and shoot other players. Since only nearby players can be shot, it is natural that players are more inclined to look for their nearby players (to shoot them or to avoid being shot by them). In other words, the players usually tend to ignore the furthest players. This can be exploited by a user. Specifically, he may issue a $R_k$FN query to find the players for whom he is among the furthest players. Such players are vulnerable targets. Note that this strategy may also be used in real world battlefields.

**Example 2.** Painter’s algorithm, a popular rendering approach, gives priority to furthest objects from the view point. An object that has many $R_k$FNs (i.e., is among the furthest objects from many view points) is likely to be accessed first for rendering. Identifying such objects and keeping them in cache can significantly improve the rendering speed [Gib01, FGR85].

**Other Examples.** $R_k$FN query has been used in reverse facility location problems [BN10, CDL+10] where the location of an obnoxious facility (e.g., a chemical plant) is decided such that the number of its least influenced users (i.e., its $R_k$FNs) is maximized. The computation of maximum spanning tree requires furthest neighbor forest [AMS91, MPSY90] which is a special graph where each node is connected to its reverse furthest neighbors. Recently, a recommender algorithm [SFJA13, SJKA12] was proposed that aims at mitigating the popularity bias and increase diversity by using furthest neighborhood that is created by connecting each user to its $R_k$FNs.

Motivated by the above and other applications [YLK09], $R_k$FN query has
gained significant attention in recent years. However, all of the existing tech-
niques [KJG08, LCFK10, LCFK12, TTS09, YLK09] study RkFN query only for
\( k = 1 \) (called RFN queries hereafter). To the best of our knowledge, we are the
first to study this problem for arbitrary value of \( k \). Although the state-of-the-art
algorithm for RFN query can be extended for \( k > 1 \), we show that based on several
novel observations, we can devise an efficient algorithm that significantly outper-
forms the state-of-the-art algorithm for arbitrary value of \( k \), including \( k = 1 \).

1.1.2 Selecting Set of Representative Objects

In the Web 2.0 era, increasing amount of spatial data are manually or algorithmi-
cally annotated. This results in a rich body of information associated with objects
rather than a minimal flat list of keyword descriptions in the pre-Web 2.0 era. This
rich information usually brings about its own sub-structures (e.g., detailed menu of
a restaurant), and may contain hyperlinks to external documents (e.g., reviews of
the restaurant at various review web sites or social network sites). For example, a
restaurant object may be associated with attributes, such as number of Foursquare
check-ins, average rating on Yelp, and number of robberies within 5km gleaned
from Twitter. At online search for products (including services and other objects)
is widely accepted and continuously increasing in popularity. Selecting a set of
representative products is of great interests for online business, review web sites,
and recommender systems.

Here, a product can be represented by \( d \) numerical attributes. Representative
skyline queries [LYZZ07, TDLP09, MAM14] select a bounded set of the most signif-
icant products. However, although such products are superior, without considering
user preference, they may not be attractive to user. Hence, they are not suitable
to be advertised at online marketplaces.
A user preference can be described by a $d$ dimensional weighting vector $w[i]$, where $0 \leq w[i] \leq 1$ and $\sum_{i=0}^{d-1} w[i] = 1$, indicating the importance of each attribute to the user. This model is widely adopted by existing works related to top-$k$ queries [THPP07, XCH06] and reverse top-$k$ queries [VDKN10]. Top-$k$ queries select the $k$ products that are ranked the highest among all the products according to a query user preference. Since these products are more attractive to the user, top-$k$ queries are helpful in filtering the most promising products out of an overwhelmingly large collection of available products. In this sense, it is meaningful for online marketplaces to list products that are ranked in top-$k$ by as many user preferences as possible. Reverse top-$k$ queries are proposed for this purpose. A product with bigger set of reverse top-$k$ user preferences is more likely to be returned to the users search products via top-$k$ queries.

Koh et. al [KLC14] selects a set of representative products that collectively covers the maximum possible number of user preference. However, considering coverage is not enough. Fig. 1.3 shows an example where diversity shall be consid-
There are 4 user preferences (i.e. \(w_1, \cdots, w_4\)) and 5 products (i.e., \(p_1, \cdots, p_5\)) in a two dimensional space. Assume \(k = 3\) (top-3 products are attractive) and \(t = 2\) (a set of 2 representative products is going to be selected). Reverse top-\(k\) for the 5 products are shown in Table 1.1. \(p_2\) and \(p_4\) has the largest set of reverse top-\(k\), while \(p_1\) and \(p_5\) are the most diverse products based on reverse top-\(k\) set difference. In this case, selecting \(p_1\) and \(p_5\) as representative products is better than selecting \(p_2\) and \(p_4\) for the following reasons: 1) reverse top-\(k\) of \(p_2\) and \(p_4\) has overlap. This selection has a focus on the users lies in the overlap of their reverse top-\(k\). Such a focus is repealing for other users; 2) \(p_2\) and \(p_4\) are not top-1 product of any user, whilst \(p_1\) and \(p_5\) covers top-1 product of all users.

Gkorgkas et al [GVDN15] has taken diversity into consideration. Their work maximize the sum of all pairs dissimilarity, where dissimilarity between two products is the distance between the centroid of their reverse top-\(k\) user preferences. Since they only consider diversity, their work suffers in some cases. Fig. 1.4 shows an example. There are 6 user preferences (i.e. \(w_1, \cdots, w_6\)) and 4 products (i.e., \(p_1, \cdots, p_4\)) in a two dimensional space. Assume \(k = 1\) and \(t = 2\). Reverse top-\(k\) for the 4 products are shown in Table 1.2. We can see that there are two bigger clusters of user preferences, \(\{w_2, w_3\}\) and \(\{w_4, w_5\}\), representing two groups of majority users, and there are two smaller clusters of user preferences, \(\{w_1\}\) and \(\{w_6\}\), representing two groups of minority users. \(p_1\) and \(p_4\) will be selected by [GVDN15]. Although \(p_2\) and \(p_3\) cover more user preferences, as the centroid of their reverse top-\(k\) user preferences are too close to each other, they will not be selected by [GVDN15].

Notice that, in the example shown in Fig. 1.4, if set difference of reverse top-\(k\) is used other than distance between the centroid of reverse top-\(k\) preferences, then \(p_2\) and \(p_3\) will also be considered as a pair of diverse products. In real world, clustered
user preferences are ubiquitous. For example, users favor different genres of movies, users favor various types of cars, and we have computers designed for game players or business man. Hence, it is important to cover as much big user groups as possible. Inspired by this, we propose to model representative products with consideration of both coverage and diversity based on reverse top-$k$ user preferences. Ideally, selected products should have minimum possible overlaps on reverse top-$k$, and they should collectively cover the maximum possible user preferences.

1.1.3 Maximize Spatial Influence of Facility Bundle

Consider a two dimensional Euclidean space, let $F$ be a set of points representing facilities (e.g., gas stations, supermarkets) and $U$ be a set of points representing users (e.g., cars, residents). Given a facility $f \in F$, the spatial influence of a facility is the number of users who have this facility as one of their $k$ nearest facilities. These users are said to be influenced by the facility. The set of all such users is referred to as the influence set of the facility. Given a facility bundle of size $t$ (e.g., a set of $t$ facilities), the spatial influence of the facility bundle is the number of distinct users influenced by any one of them.

In the literature, a user’s $k$ nearest facilities are also referred to as its bichromatic $k$ nearest neighbors ($k$NN), and a facility’s reverse $k$ nearest users are also referred to as its bichromatic reverse $k$ nearest neighbors ($Rk$NN). The concept of influence is defined based on reverse nearest neighbors (RNN) queries [KM00], reverse $k$ nearest neighbors (R$k$NN) queries [CLZZ11], and reverse top-$k$ queries [VDNK10]. A great number of existing works study facility allocation problem [WÖY+09, WÖF+11, ZWL+11, HWQ+11, CLQ+17] and facility/product selection problem [XZKD05, ZZZL12, ZZZL15, LWHC14, VDNK10, KLC14, GVDN15, WCZL15] based on the concept of influence. In this paper, we focus on facility selection prob-
lem, and we want to maximize spatial influence.

Existing works on facility selection problem find out $t$ facilities such that no other facility has higher spatial influence than any one of the $t$ facilities, either based on RNN queries [XZKD05, ZZZL12, ZZZL15] or R$k$NN queries [LWHC14]. However, the literature lacks study on this problem when the $t$ facilities are selected as a facility bundle. The distinctive feature of facility bundle is that the combined spatial influence of the $t$ facilities is considered other than the spatial influence of the $t$ facilities individually. The scenario of facility bundling finds a lot of real world applications.

Consider an example of facility bundling, where the government wants to promote or do a survey on a new policy by campaign at a couple of supermarkets, so that the residents shopping at those supermarkets (e.g., influence sets) will be covered (e.g., influenced). For consideration of budget, only a limited number of supermarkets, let’s say $t$ supermarkets, can be selected, but the government wants to cover as much residents as possible (e.g., maximize spatial influence). Notice that in this example, a user may go shopping at two selected supermarkets, but should not be counted as two heads. As a result, we can not simply select the top-$t$ facilities with the largest spatial influence. Instead, we need to select a facility bundle with the highest spatial influence among all possible facility bundles of size $t$.

Consider another example, where we have a corporate group manages a large number of facilities. The corporate group wants to create a facility bundle by selecting $t$ facilities and promote a joint membership so that users of any one of them can also get discount at all the other selected facilities. The joint membership aims to increase the number of potential users for all the selected facilities without extra customer acquisition costs. In this example, to maximize the number of
potential users of every selected facility, we need to maximize the spatial influence of the selected facility bundle.

The example in Fig. 1.5 illustrates the reason why we cannot adapt existing works on facility selection problem to facility bundle selection problem. $f_1$, $f_2$ and $f_3$ are 3 facilities. Assume the green triangles inside the 3 circles, $C_1$, $C_2$ and $C_3$, are their R$k$NN users, then their spatial influence are 5, 4 and 3, respectively. When $t = 2$, existing works would select $f_1$ and $f_2$ because they have the largest spatial influence individually. However, the spatial influence of $\{f_1, f_2\}$ is 6, which is smaller than the spatial influence of $\{f_1, f_3\}$, which is 8. As a facility bundle of size 2, $\{f_1, f_3\}$ is better than $\{f_1, f_2\}$. Motivated by this, we study the problem of maximizing bundled reverse $k$ nearest neighbors (MB-R$k$NN), so that the spatial influence of a facility bundle of size $t$ is maximized.

It can be shown that the MB-R$k$NN problem is NP-hard by reducing an existing NP-complete problem called the Maximum Coverage Problem to MB-R$k$NN problem in polynomial time. Hence, we propose a greedy algorithm that iteratively select the currently best facility that increases spatial influence of the current facility bundle the most. We assume that the data sets, namely the facilities and users are indexed by R-tree, respectively. The greedy algorithm is a branch-and-bound
best first search algorithm on R-tree. Moreover, we introduce the concept of \( k \)NN region, which is an area shared by a group of users, such that the \( k \)NN facilities of them are all contained in the same area. \( k \)NN region shared between users allows us to avoid redundant calculation with dynamic programming technique.

1.2 Contributions

In this section, we summarize our contribution of this thesis. For each of the three problems we studied in this thesis, we are the first to study the problem, and we proposed efficient algorithms to address the problems.

1.2.1 Efficiently Computing Reverse \( k \) Furthest Neighbors

To the best of our knowledge, we are the first to study the problem of R\( k \)FN query for arbitrary value of \( k \). Based on several interesting observations and novel pruning techniques, we devise an efficient algorithm to process R\( k \)FN queries.

We present a rigorous theoretical analysis to analyse two different aspects of the problem, namely the expected area of \( k \)-depth contour and the area pruned by our algorithm. This helps in analysing the expected number of futile queries, number of facilities required for computing a R\( k \)FN query, number of unpruned users and I/O cost etc. Also, to the best of our knowledge, we are the first to analyse the expected area of \( k \)-depth contour which is of stand-alone interest.

We conduct extensive experimental study on both real and synthetic data sets, and demonstrate that our algorithm is up to several orders of magnitude faster than the existing algorithms in terms of both CPU and I/O cost. We also conduct experiments to evaluate the accuracy of our theoretical analysis.
1.2.2 Selecting Set of Representative Objects

We are the first to study the problem of selecting set of representative products while considering both coverage and diversity based on reverse top-$k$ queries.

We formulate the problem to be an optimization problem of an objective function. As this problem is NP-hard, we propose a $\epsilon$-approximate greedy algorithm.

To enhance the scalability and efficiency when the number of user preferences is huge, we propose to boost set operations by adopting MinHash and KMV Synopses.

1.2.3 Maximize Spatial Influence of Facility Bundle

We introduce the concept of spatial influence of a facility and a facility bundle. We are the first to study the problem of Maximizing Bundled Reverse $k$ Nearest Neighbor (MB-R$k$NN).

We prove that MB-R$k$NN problem is NP-hard and propose a branch-and-bound best first search algorithm that greedily select the currently best facility. We introduce the concept of $k$NN region that is shared by a group of users. This allows us to avoid redundant calculation with dynamic programming technique.

We conduct experiments on real data sets. Experiments show that our algorithm is orders of magnitudes better than our baseline algorithm both in terms of CPU time and IO cost.

1.3 Thesis Organization

We first survey the related work in Chapter 2. Then we study the three problems one in each of the three following chapters. In Chapter 3, we study the problem of efficiently computing reverse $k$ furthest neighbors (R$k$FN) queries. In Chapter 4, we study the problem of selecting set of representative products considering both
diversity and coverage based on reverse top-$k$ queries. In Chapter 5, we study the problem of maximizing spatial influence of facility bundle based on reverse $k$ nearest neighbors (R$k$NN) queries. Finally in Chapter 6, we conclude this thesis. More specifically, each chapter is structured as follows.

- Chapter 2 provides a literature review of the existing work related to the three problems. In Section 2.1, we review related works related to reverse $k$ furthest neighbors queries. In Section 2.2, we review related works on the problem of top-$k$ and reverse top-$k$ queries, and we review related works regarding representative products. In Section 2.3, we review related works on the problem of reverse $k$ nearest neighbors queries, and we review related works regarding facility allocation problem, product selection problem and facility selection problem.

- Chapter 3 studies the problem of efficiently computing reverse $k$ furthest neighbors. Section 3.1 present preliminaries. Specifically, we describe how to extend the state-of-the-art work on reverse furthest neighbors queries to reverse $k$ furthest neighbors queries. This extension would be used as our baseline algorithm. Section 3.2 presents an overview of our algorithm. Our techniques are presented in Section 3.3 followed by theoretical analysis in Section 3.4. An extensive experimental study is provided in Section 3.5.

- Chapter 4 studies the problem of selecting set of representative products considering both diversity and coverage based on reverse top-$k$ queries. This chapter is structured as follows. Section 4.1 provides the necessary preliminaries, including top-$k$ and reverse top-$k$ queries, MinHash and KMV synopsis. In Section 4.2 we formally define the $t$-representative problem. We present a greedy algorithm in Section 4.3 and prove it is $\epsilon$-approximate in Section 4.4.
Finally, we present experimental results in Section 4.5.

- Chapter 5 studies the problem of maximizing spatial influence of facility bundle based on reverse $k$ nearest neighbors (R$k$NN) queries. This chapter is structured as follows. Section 5.1 present a formal definition of MB-R$k$NN problem, prove its NP-hardness, and present a baseline algorithm. Section 5.2 present our algorithm. Section 5.3 present experimental results.

- Chapter 6 concludes our research by summarizing the contributions for each problem we studied in this thesis.
Chapter 2

Related Work

This chapter provides a non-exhaustive review of the literature related to the three problems studied in this thesis. Each one of them is closely related to one particular spatial query. We hence organize this chapter in three sections accordingly.

- **Reverse $k$ Furthest Neighbors Queries** (Section 2.1) : This section reviews all the existing works related to reverse $k$ furthest neighbors ($RkFN$) queries, although there are only a few of them. We study $RkFN$ queries in Chapter 3, and they are directly related to our work.

- **Top-$k$ and Reverse Top-$k$ Queries** (Section 2.2) : This section reviews existing works related to top-$k$ and reverse top-$k$ queries, and also reviews existing works related to the problem of selecting representative products. We study the problem of selecting set of representative products based on reverse top-$k$ queries in Chapter 4, and they are closely related to our work.

- **Reverse $k$ Nearest Neighbors Queries** (Section 2.3) : This section reviews existing works related to reverse $k$ nearest neighbors ($RkNN$) queries, and also reviews existing works related to the problem of facility allocation
problem, product selection problem and facility selection problem. We study the problem of maximizing spatial influence of facility bundle based on reverse $k$ nearest neighbors (R$k$NN) queries in Chapter 5, and they are closely related to our work.

2.1 Reverse $k$ Furthest Neighbors Queries

Reverse $k$ Furthest Neighbors Queries. Kumar et. al [KJG08] presented an approximate approach to compute RFNs. It requires pre-calculation of so called FN-ball of points. Tran et. al [TTS09] address RFN query on road network based on Network Voronoi Diagram and pre-computation of network distances. Liu et. al [LCFK10] proposed PIV algorithm that constructs metric indexes and employs triangle inequality to do pruning. Their follow up work [LCFK12] proposed PIV+, which identifies a safe area that confirms RFN results without invoking the PIV process. All these techniques require expensive pre-computation and, therefore, are not suitable for dynamic data sets or for arbitrary value of $k$.

The state-of-the-art algorithm for RFN query in 2D Euclidean space is CHFC (Convex Hull Furthest Cell) proposed by Yao et. al [YLK09]. CHFC computes the convex hull (CH) of the set of facilities $F$. They proved that the furthest neighbor (FN) of a user is a facility that lies on CH. Therefore, $q$ cannot have any RFN if lies inside CH. For example, in Fig. 2.1, $f_9$ cannot have any RFN because it lies inside CH. If $q$ lies on CH, CHFC proceeds to calculate furthest voronoi cell (FVC) of $q$, which is a convex polygon in the space such that a point is RFN of $q$ if and only if it is inside FVC.

R$k$NN Query. Most of the existing techniques adopt a pruning and verification framework. The two most notable pruning techniques are regions-based prun-
Figure 2.1: CHFC

The best known algorithm in terms of I/O cost is influence zone [CLZZ11] and the state-of-the-art algorithm in terms of overall running time is SLICE [YCLZ14]. A brief survey and empirical comparison of some of the most popular R$k$NN algorithms is provided in [YCLW15].

$k$-depth contour $k$-depth contour (also known as $k$-hull) is a well studied topic in the field of Computational Geometry and Statistics. Several efficient algorithms [CSY84, KMV02, MRR+01] have been proposed in the past. However, these algorithms assume that the data can fit in main-memory. Böhm et. al [BK01] proposed I/O optimal algorithms for $k = 1$. Cheema et. al [CSLZ14] proposed an I/O optimal algorithm for arbitrary value of $k$.

2.1.1 CHFC

Given a set of points $P$, a query set $Q$ and a query point $q \in Q$, a bichromatic reverse furthest neighbors (BRFN) query fetches the set of points $p \in P$ such that $q$ is the furthest neighbor of $p$ among all points in $Q$.

The convex hull $C_q$ for the set of points $Q$ is the smallest convex polygon defined
by points in $Q$ that fully contains $Q$. Given a point $q \in Q$, Yao et. al [YLK09] proved that the furthest point from $P$ to $q$ must be found in $C_q$, and only points in $C_q$ will have reverse furthest neighbors w.r.t the data set $P$.

Given a query point $q$, we could view it as an insertion to $Q$ and obtain a new data set $Q^* = Q \cup \{q\}$. It is critical to decide whether $q$ belongs to the set of vertices defining the convex hull of $Q^*$. straightforwardly, $C_{q^*}$ could be computed efficiently from $C_q$ and $q$ alone, without looking at the rest of points in $Q \setminus C_q$. That is, if $q$ is strictly contained by $C_q$, then $C_{Q \cup \{q\}} = C_q$. Otherwise, $C_{Q \cup \{q\}} = C_{C_q \cup \{q\}}$.

Algorithm 1: CHFC

**Input**: $q$; $Q$; $T$: an R-tree that indexes $P$

**Output**: Reverse furthest neighbors of $q$

1. Compute $C_q$;
2. if $q \subset C_q$ then
   3. return $\emptyset$;
4. else
   5. Compute $C_{q^*}$ using $C_q \cup \{q\}$;
   6. Set $fvc(q, Q^*)$ equal to $fvc(q, C_{q^*})$;
   7. Execute a range query using $fvc(q, Q^*)$ on $T$;

Algorithm 1 illustrates how CHFC algorithm works. The convex hull of $Q$, namely $C_q$ is first calculated. Given a query point $q$, we can then quickly return empty if $q \subset C_q$. Otherwise, we set $Q^* = Q \cup \{q\}$ and compute $C_{q^*}$ using only $C_q$ and $q$. Next, the furthest Voronoi cell of $q \in Q^*$ is calculated using only $C_{q^*}$. Finally, a range query using $fvc(q, Q^*)$ is executed on the R-tree of $P$ to retrieve the reverse furthest neighbors (RFNs) of $q$. This is referred as the convex hull furthest cell (CHFC) algorithm.
The efficiency of the CHFC algorithm is achieved in two folds. First of all, all query points that are enclosed by $C_q$ are extremely fast to deal with. Secondly, for the rest of query points, computing their furthest Voronoi cells becomes much more efficient by taking into account only the points in $C_q$. In practice, $|C_q| \ll |Q|$.  

2.1.2 SkyRider

A point $p = (u, v)$ in primal space is mapped to a line $p^* : y = ux + v$ in the dual space and a line $L : y = -ux + v$ in the primal space is mapped to a point $L^* = (u, v)$ in the dual space.

A weighting vector $\vec{w} = (w_1, w_2)$ in primal space is mapped to a vertical line $w^* : x = \frac{w_1}{w_2}$ in dual space. Let the $y$-coordinate value of the point where $a^*$ (resp. $b^*$) meets the vertical line $w^*$ be called $y_a$ (resp. $y_b$). Cheema et. al [CSLZ14] proved that the relative ranking of $a$ and $b$ w.r.t. the weighting vector $\vec{w}$ can be determined by comparing $y_a$ and $y_b$.

Consider two points $a = (a_1, a_2)$ and $b = (b_1, b_2)$ in primal space. Given a weighting vector $\vec{w} = (w_1, w_2)$, the score of $a$ is $a.score = w_1a_1 + w_2a_2$. Similarly, $b.score = w_1b_1 + w_2b_2$. When $w_2 > 0$, $a.score < b.score \iff y_a < y_b$. When $w_2 < 0$, $a.score < b.score \iff y_a > y_b$.

Consider a weighting vector $\vec{w} = (w_1, w_2)$. When $w_2 > 0$, top-$k$ objects correspond to the $k$-lowest lines intersecting $w^*$ in the dual space. When $w_2 < 0$, top-$k$ objects correspond to the $k$-highest lines intersecting $w^*$.

$k$-lower (resp. upper) envelope. Consider a set of lines $L$. Lower (resp. upper) score of a point $p$ is the number of lines below (resp. above) $p$. $k$-lower (resp. upper) envelope is the closure of the set of points that have lower (resp. upper) score equal to $k - 1$.

Upper (resp. lower) hull. Let $Z$ be the convex hull of a set of points $P$. The
upper (resp. lower) hull of $P$ is the set of edges of $Z$ that lie on or above (resp. on or below) every point $p \in P$. In Figure 2.3, the upper hull of the points in $k$-lower envelope is shown using dotted lines.

There are three steps to compute $k$-depth contour in the dual space. First, map all objects in primal space to dual space. Second, compute $k$-lower envelope and $k$-upper envelope of these lines and determine the convex vertices by computing upper and lower hulls. Third, map the convex vertices to lines in primal space and use these lines to obtain the $k$-depth contour.

**Rider algorithm for computing $k$-lower envelope.** Assume that all objects in primal space have been mapped to lines in dual space. Let origin line $L_o$ be the line with the $k$-th smallest slope. Let destination line $L_d$ be the line with the $k$-th largest slope. Assume that all lines are roads and a bike rider starts traveling from the right most point on the origin line (i.e., at $x = \infty$). The rider always travels towards his left (i.e., towards decreasing value of $x$). Whenever he reaches at an intersection of two lines, he makes a turn. The rider keeps traveling until it reaches the left most point of the destination line (i.e., at $x = -\infty$). It is easy to verify that the path that the rider travels on corresponds to the $k$-lower envelope.
SkyRider algorithm. Assume that the preference function \( f_1 \) prefers smaller values on both coordinates, and another preference function \( f_2 \) prefers smaller values on \( y \)-coordinate and larger values on \( x \)-coordinate. The objects that are \( k \)-skyband objects according to \( f_1 \) and \( f_2 \) are sufficient to be used to correctly compute the \( k \)-lower envelope. Hence, SkyRider algorithm first computes these two \( k \)-skybands using BBS [PTFS05] and stores \( k \)-skyband objects in a main-memory R-tree which is then used by the rider algorithm.

2.2 Top-\( k \) and Reverse Top-\( k \) Queries

Top-\( k \) and Reverse Top-\( k \) Queries. Tao et al [THPP07] process top-\( k \) queries based on branch-and-bound search on R-tree index. Xin et al [XCH06] propose index method to boost run time performance of top-\( k \) query processing.

Reverse top-\( k \) queries is first introduced by Vlachou et al [VDKN10]. The size of a product’s reverse top-\( k \) indicates its potential market and possible profit. Ge et al [GUMC13] evaluate batch of top-\( k \) queries by grouping similar queries, which in turn enables the computation of a large number of reverse top-\( k \) queries. Yu et al [YAY12] address the problem of processing a large number of continuous top-\( k \) queries. The proposed method relies on a pre-calculated index of the \( k \)-th ranked objects. Vlachou et al [VDNK13] propose a branch-and-bound algorithm for reverse top-\( k \) queries that based on R-tree index.

Representative Products. Lin et al [LYZZ07] select a set of skyline representatives, which collectively dominate as many distinct points as possible. This work maximize coverage. Tao et al [TDLP09] select a set of representative skyline points that best describe the skyline contour. This work maximize diversity. Magnani et al [MAM14] take both the significance (sigmoid significance) of all
the records and their diversity (skyline edit distance, namely the number of skyline records in between) into account, and select representative skyline points that maximize the sum of significance and minimum distance to other skyline points.

Top-\(m\) influential query [VDNK10] finds the \(m\) most influential products based on the influence score, which is the cardinality of a product’s reverse top-\(k\) user preference. The selected products have significant impact in the market individually. Lin et. al [LKC13] study the problem of determining a set of most demanding products. In this work, products are not ranked according to user preferences. Instead, a product is acceptable as long as the product is below a threshold defined by a user. They also assume all qualified products have equal probability to be selected by a user, and calculate the potential profit of a product by summing up the chances it is selected across all users. This work considers coverage only. Koh et. al [KLC14] select a set of products that collectively covers the maximum possible number of reverse top-\(k\). This work considers coverage only. Gkorgkas et. al [GVDN15] select a set of products that are most diverse to each other based on centroid of reverse top-\(k\). This work considers diversity only.

Other Related Works. Drosou et. al [DP12] study the problem of result diversification based on dissimilarity and coverage. In their definition, the neighbors of an object, i.e., the objects lying at distance no more than a threshold are considered as similar to the object. A diverse subset selected for a query result contains objects such that: 1) each object in the result is represented by at least one similar object in the diverse subset, and 2) none of the objects in the diverse subset are similar to each other. Indyk et. al [IMMM14] consider efficient construction of composable core-sets for diversity and coverage maximization problems. A core-set is representative in the sense that an approximate solution to the whole data set can be obtained from the core-set only.
2.2.1 Reverse top-k threshold algorithm

Given a point \( q \) and a positive number \( k \), as well as two data sets \( S \) and \( W \), where \( S \) represents data points and \( W \) is a data set containing different weighting vectors, a weighting vector \( w_i \in W \) belongs to the bichromatic reverse top-k (\( bRTOP_k(q) \)) result set of \( q \), if and only if \( \exists p \in TOP_k(w_i) \) such that \( f_{w_i}(q) \leq f_{w_i}(p) \).

\( W \) is stored sorted. The first weighting vector \( w_1 \) is the most similar vector to the diagonal vector of the space. Thereafter, the most similar weighting vector \( w_{i+1} \) to the previous vector \( w_i \) is examined.

Algorithm 2: RTA : Reverse top-k Threshold Algorithm

| Input : \( S; W; q; k \) |
| Output: \( bRTOP_k(q) \) |

1. \( W' \leftarrow \{\emptyset\} \);
2. \( buffer \leftarrow \{\emptyset\} \);
3. \( threshold \leftarrow \infty \);
4. for each \( w_i \in W \) do
   5. \[ \text{if } f_{w_i}(q) \leq threshold \text{ then} \]
   6. \[ buffer \leftarrow TOP_k(w_i); \]
   7. \[ \text{if } f_{w_i}(q) \leq \max\{f_{w_i}(buffer)\} \text{ then} \]
   8. \[ W' \leftarrow W' \cup \{w_i\}; \]
   9. \[ threshold \leftarrow \max\{f_{w_{i+1}}(buffer)\}; \]
10. return \( W' \);  

Algorithm 2 describes the RTA algorithm for processing a bichromatic reverse top-k query. Initially, RTA computes the top-k result \( TOP_k(w_i) \) for the first weighting vector (line 6). Notice that in the first iteration we cannot avoid evaluating a top-k query, as the threshold cannot be set yet. The \( k \) data points that belong to
the result set $TOP_k(w_i)$ are kept in a main memory buffer.

Given a set of points $P$, we denote as $\max\{f_{w_i}(P)\}$ the maximum value of all s-core values $f_{w_i}(p_j), p_j \in P$, which means that $\max\{f_{w_i}(P)\} \geq f_{w_i}(p_j), \forall p_j \in P$. The score $f_{w_i}(q)$ of query point $q$ based on vector $w_i$ is computed and compared against the maximum $f_{w_i}$ value of all points in the buffer, denoted as $\max\{f_{w_i}(buffer)\}$ (line 7). This maximum score defines the threshold value. If the score $f_{w_i}(q)$ is smaller than or equal to $\max\{f_{w_i}(buffer)\}$, then $w_i$ is added to the result set.

Before the next iteration of the algorithm, we take the next weighting vector $w_{i+1}$ and we set as threshold value the maximum score of any point in the buffer based on this new vector. Then the condition of line 5 is tested, so if the score $f_{w_i}(q)$ is larger than the threshold, then we can safely discard $w_i$. Otherwise, we have to evaluate the top-$k$ query for the vector $w_i$, in order to determine whether $w_i$ belongs to the reverse top-$k$ result. Therefore, we pose again a top-$k$ query on data set $S$ and we update the main memory buffer with the new result set $TOP_k(w_i)$.

In each iteration, the $k$ points of the previously processed top-$k$ query are kept in the buffer. Notice that the size of the buffer is limited, since queries with small $k$ values are commonly used in practice. The algorithm terminates when all weighting vectors have been evaluated or discarded.

### 2.2.2 Branch-and-Bound algorithm for reverse top-$k$

Given a set of weighting vectors $V$ and a data point $p$, the score-lower-bound of $p$ is $L_V(p) = \sum_{i=1}^{n} \min_{w \in V} (w[i]) \cdot p[i]$. And the score-upper-bound of $p$ is $U_V(p) = \sum_{i=1}^{n} \max_{w \in V} (w[i]) \cdot p[i]$.

A set of data points can be represented by a minimum bounding rectangle (MBR) $m$. We denote as $m.l$ the lower-left corner and as $m.u$ the upper-right corner. Given a set of weighting vectors $V$ and an MBR $m$ of data points. We can
define a lower and upper bound of the scores of all data points that are enclosed in $m$ based on any $w \in V$. The bounds are $L_V(m.l)$ and $U_V(m.u)$ respectively.

**INTOP$k$ algorithm.** Given a data set $S$, the INTOP$k$ algorithm takes as an input the MBR $m_V$, a query point $q$ and a value $k$, and returns whether $q$ belongs to the top-$k$ result set for all or none of the weighting vectors enclosed in $m_V$. If none of these cases holds, the INTOP$k$ algorithm returns that it cannot give a definite answer. The algorithm uses two counters to determine whether $m_V$ belongs to the result set or not while traversing the R-tree of data set $S$. The first counter counts the entries $e$ such that $e.m$ precede $q$ based on $V$. If the number of these entries is greater than or equal to $k$, then we can exclude $m_V$ from the result set. The second counter counts the data points that are either incomparable or precede $q$. If fewer than $k$ such entries exist, $m_V$ can be added to the result set.

**Branch-and-Bound algorithm (BBR).** The algorithm uses an R-tree to index the data set of weighting vectors $W$. Intuitively, as the algorithm traverses this index, it bounds the search space of results for the reverse top-$k$ query by discarding MBRs of weighting vectors that cannot contribute to the result set. Essentially, each time an entry of the R-tree is processed, the algorithm tests whether the weighting vectors that are enclosed by the MBR of the entry, forming a set $V$, may belong to the result set or whether it can be discarded. This test is accomplished in practice by using INTOP$k$ algorithm. If INTOP$k$ algorithm is inconclusive about $V$, the current entry of the R-tree needs to be expanded, and smaller subsets of $V$ need to be examined for inclusion or pruning. The goal of BBR is to expand as few entries as possible by either discarding entries of the R-tree or by adding them to the result set immediately.
2.2.3 Top-\(k\) most favorite products maximizing coverage

A reverse top-\(t\) query for a product returns a set of customers, named potential customers, who regard the product as one of their top-\(t\) favorites. Given a set of customers with different preferences on the features of the products, we want to select at most \(k\) products from a pool of candidate products such that their total number of potential customers is maximized.

After computing the set of reverse top-\(t\) for each candidate product, the problem is then equivalent to the maximum coverage problem which is one of the well-known NP-hard problems. Therefore, an incremental greedy algorithm is proposed for solving the problem.

First, the candidate products are generated by performing a natural join operation. The set of potential customers for each candidate product is computed. Then, according to the reverse top-\(t\) of each candidate product, the incremental greedy algorithm is used to select at most \(k\) candidate products.

Koh et. al [KLC14] proved that there exists one solution of the problem consisting of only the candidate products in the skyline of all candidate products, denoted as \(\text{Skyline}(CP)\). The search space can then be reduced to \(\text{Skyline}(CP)\). Besides, only the candidate products in the \(t\)-skyband of all candidate products, denoted as \(t\text{Skyband}(CP)\), can be the top-\(t\) favorites for a customer.

Let \(cp\) denote a candidate product in \(\text{Skyline}(CP)\). For each customer \(c\) in \(RTOP_t(cp, CP, C)\), \(c\) must belong to \(RTOP_t(cp, \text{Skyline}(CP), C)\) because \(\text{Skyline}(CP)\) is a subset of \(CP\) and \(cp\) is in \(\text{Skyline}(CP)\). Accordingly, \(RTOP_t(cp, \text{Skyline}(CP), C)\) is a superset of \(RTOP_t(cp, CP, C)\), which can be used to estimate an upper bound of \(RTOP_t(cp, CP, C)\).

Three main steps of the proposed algorithm for reducing the computation cost of the problem are: (1) reducing the search spaces of both the problem and the top-
Chapter 2. Related Work

2.2.4 Finding the most diverse products

The goal is to find a set of products that are attractive to a wide range of customers with different preferences.

The problem of selecting the \( r \) most diverse products from a given set \( S \) can be viewed as a dispersion problem, where the aim is to find \( r \) objects such that an objective function of their distance \( d \) is optimized. The dispersion sum problem maximizes the sum of pairwise distances between the \( r \) selected products and it has been proved that it is NP-hard by reduction from the clique problem.

Given a set of data objects \( S \), a set of weighting vectors \( W \), and an object \( p \in S \) such that \( RTOP_k(p) \neq \emptyset \), the centroid of \( p \) is defined as the vector \( c_p = \frac{1}{|RTOP_k(p)|} \sum_{w \in RTOP_k(p)} w \). Since each \( RTOP_k \) set corresponds to exactly one data point, the respective centroid corresponds to exactly one data point as well. Therefore, each data point can be mapped to exactly one centroid vector and vice-versa.

Given a set of data objects \( S \), a set of weighting vectors \( W \), two objects \( p, q \in S \), and their respective centroids \( c_p \) and \( c_q \), the distance of \( p \) and \( q \) is defined based on the cosine similarity of the centroids, namely \( 1 - \cos(c_p, c_q) \). Given a subset \( D \subseteq S \) of size \( r \), diversity of \( D \) is defined as \( \text{div}(D) = \frac{2}{r(r-1)} \sum_{p,q \in D, p \neq q} (1 - \cos(c_p, c_q)) \).

The most straightforward method is to perform a reverse top-\( k \) query for each product \( p \) in \( S \) and compute the centroid vector of each set \( RTOP_k(p) \). A better method exploits the observation that it may be more efficient to process all top-\( k \)
queries, instead of processing multiple reverse top-$k$ queries.

After having computed the centroid vectors of all non-empty reverse top-$k$ sets, the next step is to find the $r$ most diverse centroids and the products that they represent. More specifically, a greedy algorithm is used. The greedy algorithm is called Diverse Product Selection Algorithm (DPSA), that iteratively selects the next centroid that maximizes the value of the objective function.

The main drawback of the previous algorithm is that it requires the computation of all centroids, which has a significant processing cost regardless of the employed method for candidate centroid computation. In order to alleviate this shortcoming, Gkorgkas et. al [GVDN15] proposed a method that fuses the centroid computation with the selection of diverse objects. Their goal is to efficiently compute an approximation of the centroids (by evaluating only a handful of carefully selected top-$k$ queries), which is sufficient to produce a set of $r$ products with high diversity.

Conceptually, the proposed algorithm uses a series of iterations, where each iteration consists of three parts: (1) select a weighting vector $w_i$ in order to process the top-$k$ query it defines, (2) compute an approximation of the centroid-set based on the results of all already processed top-$k$ queries, and (3) select diverse products by invoking the DPSA algorithm with input the approximate centroid-set.

In each iteration, a top-$k$ query based on $w_i$ is executed. Some objects $p \in TOP_k(w_i)$ may not have been retrieved before and those are added to the centroid-set. For the remaining objects $p \in TOP_k(w_i)$ the approximate centroid is updated, since $w_i$ is added to their reverse top-$k$ sets. In fact, the reverse top-$k$ sets are not maintained, but the centroid of an object is computed progressively. Thus, in each iteration the centroid-set is only an approximation of the candidate centroids. However, in each iteration, the candidate-set is enriched with the results of the next
top-\(k\) query. Additionally, a set of \(r\) diverse products \(D_r(S)\) is computed based on the current set of centroids. Finally, the selection of the next weighting vector to be processed is based on maximizing the sum of distances to the set of centroids defined by \(D_r(S)\).

2.3 Reverse \(k\) Nearest Neighbors Queries

**Reverse \(k\) Nearest Neighbors Queries.** Reverse Nearest Neighbor (RNN) query is first introduced by Korn et. al in [KM00]. Given a set \(F\) of facilities and a set \(U\) of users, the reverse nearest neighbors of a facility \(f \in F\) are the users whose nearest neighbors in \(f\). The set of such users is the influence set of \(f\), and the influence of \(f\) can be measured by the number of users whose nearest neighbors is \(f\). A generalization of RNN query is reverse \(k\) nearest neighbor (R\(k\)NN) query, where users are assumed to prefer not just their nearest facility but their \(k\) nearest facilities. Most of the existing techniques adopt a pruning and verification framework to solve R\(k\)NN queries. The two most notable pruning techniques are regions-based pruning [SAE00, YCLZ14] and half-space pruning [TPL04, CLZZ11]. The best known algorithm in terms of IO cost is influence zone [CLZZ11] and the-state-of-the-art algorithm in terms of overall running time is SLICE [YCLZ14].

**Facility Allocation Problem.** Wong et. al [WÖY+09] introduce the MaxOverlap problem and proposed the MaxOverlap algorithm in the two-dimensional case when the metric space is the \(L_2\)-norm space. They then extend the problem definition to any Minkowski metric of order 1 or above for two- and three-dimensional spaces and propose appropriate algorithmic solutions [WÖF+11]. These two works maximize spatial influence based on RNN queries. Zhou et. al [ZWL+11] find an optimal region that maximize spatial influence based on R\(k\)NN queries. More gen-
erally, Huang et. al [HWQ+11] select a set of $t$ optimal locations from a given set of candidate locations such that no other candidate location is more influential than any one of them. Alternatively, Chen et. al [CLQ+17] finds out a set of $t$ regions such that setting up $t$ new facilities, one in each region, will collectively attract the maximum number of RNN users. The distinctive feature of this work is that it optimizes the combined influence of $t$ selected regions rather than the influence of any single region. However, facility allocation problem is intrinsically different from facility selection problem. Locations are represented by either points or regions, and are either given or not given, but can never be collocated with existing facilities. In other words, what is being selected can never be existing facilities.

**Product Selection Problem.** In product selection problem, top-$t$ most influential products are selected based on the influence, which is defined as the cardinality of a product’s reverse top-$k$ user preference [VDNK10]. The selected products have significant impact in the market individually. Koh et. al [KLC14] considers coverage, and select a set of products that collectively covers the maximum possible number of reverse top-$k$. Gkorgkas et. al [GVDN15] considers diversity, and select a set of products that are most diverse to each other based on centroid of reverse top-$k$. Wang et. al [WCZL15] considers a combination of coverage and diversity. Product selection problem is intrinsically different from facility selection problem as the influence of a product is defined based on reverse top-$k$ queries, but not R$k$NN queries.

**Facility Selection Problem.** Previous works tried to find out $t$ facilities such that no other facility is more spatially influential than any one of the $t$ selected facilities, either based on RNN queries [XZKD05, ZZZL12, ZZZL15] or R$k$NN queries [LWHC14]. Sun et. al [SZX+16] study the problem of constructing a heat (e.g., spatial influence) map, where points share the same set of RNN are grouped
together. However, the literature lacks study on facility selection problem when the \( t \) facilities have the highest spatial influence as a bundle.

### 2.3.1 Influence zone

Consider a set of facilities \( F = \{f_1, f_2, \cdots, f_n\} \) and a query \( q \in F \) in a Euclidean space. Given a point \( p \), \( C_p \) denotes a circle centered at \( p \) with radius equal to \( \text{dist}(p, q) \) where \( \text{dist}(p, q) \) is the distance between \( p \) and \( q \). \( |C_p| \) denotes the number of facilities that lie within the circle \( C_p \). The influence zone \( Z_k \) is the area such that for every point \( p \in Z_k \), \( |C_p| < k \) and for every point \( p' \notin Z_k \), \( |C_{p'}| \geq k \). Given a set of users \( U \), a bichromatic \( RkNN \) query is to retrieve every user \( u \in U \) for which \( |C_u| < k \).

Given two facility points \( a \) and \( q \), a perpendicular bisector \( B_{a,q} \) between these two points divides the space into two halves. The half plane that contains \( a \) is denoted as \( H_{a,q} \). We say that the point \( p \) is pruned by the bisector \( B_{a,q} \) if \( p \) lies in \( H_{a,q} \). In general, if a point \( p \) is pruned by at least \( k \) bisectors then \( C_p \) contains at least \( k \) facilities.

One straightforward approach to compute the influence zone is to consider the bisectors of \( q \) with every facility point \( f \). If the bisectors of \( q \) and all facilities are considered, then the unpruned area is the area that is pruned by less than \( k \) bisectors. However, this straightforward approach is too expensive because it requires computing the bisectors between \( q \) and all facility points. It is noted that for some facilities, there is no need to consider their bisectors.

Cheema et al. [CLZZ11] proved that a facility \( f \) can be ignored if for every point \( p \) of the unpruned polygon, the facility \( f \) lies outside \( C_p \). A facility \( f \) can be ignored if, for every point \( p \) on the boundary of the unpruned polygon, \( f \) lies outside \( C_p \). A facility \( f \) can be ignored if, for every vertex \( v \) of the unpruned
polygon, the facility \( f \) lies outside \( C_v \). A facility \( f \) can be ignored if it lies outside \( C_v \) for every convex vertex \( v \) of the unpruned polygon \( P \).

Algorithm 3: Influence Zone

\[
\text{Input} : \text{ a set of objects } O; \text{ a query } q \in O; k
\]

\[
\text{Output}: \text{ Influence zone } Z_k
\]

1. initialize \( Z_k \) to the boundary of data universe;
2. insert root of R-tree in a min-heap \( h \);
3. while \( h \) is not empty do
   4. deheap an entry \( e \);
   5. for each convex vertex \( v \) of \( Z_k \) do
      6. if \( \text{mindist}(v, e) < \text{dist}(v, q) \) then
         7. mark \( e \) as valid; break;
     8. if \( e \) is valid then
        9. if \( e \) is an intermediate node or leaf then
           10. insert every child \( c \) in \( h \) with key \( \text{mindist}(q, c) \);
        else if \( e \) is an object then
           12. update the influence zone \( Z_k \) using \( e \);

Algorithm 3 presents the details. Initially, the whole data space is considered as the influence zone and the root of the R-tree is inserted in a min-heap \( h \). The entries are iteratively de-heaped from the heap. The entries in the heap may be rectangles (e.g., intermediate nodes) or points. If a de-heaped entry \( e \) completely lies outside \( C_v \) of all convex vertices of the current influence zone (e.g., the current unpruned area), it can be ignored. Otherwise, it is considered valid. If the entry is valid and is an intermediate node or a leaf node, its children are inserted in the
heap. Otherwise, if the entry \( e \) is valid and is a data object, it is used to prune the space. The current influence zone is also updated accordingly. The algorithm stops when the heap becomes empty.

### 2.3.2 Slice

Given a query point \( q \), the subtended angle between two points \( x \) and \( y \) is the angle \( \angle xqy \) in the triangle \( \triangle xqy \). It is denoted as \( \text{angle}(x, y) \). If \( x, q \) and \( y \) lie on the same line then \( \text{angle}(x, y) = 0^\circ \) or \( \text{angle}(x, y) = 180^\circ \) depending on the relative positions of \( x, q \) and \( y \).

Maximum subtended angle between a point \( x \) and a partition \( P \) is the maximum subtended angle between \( x \) and any point \( p \) in the partition \( P \), i.e., \( \arg\max_{p \in P} \text{angle}(x, y) \). It is denoted as \( \text{maxAngle}(x, P) \). The minimum subtended angle is defined similarly and is denoted as \( \text{minAngle}(x, P) \).

A facility \( f \) prunes every point \( p \in P \) for which \( \text{dist}(p, q) > \frac{\text{dist}(f, q)}{2 \cos(\theta)} \) where \( \theta = \text{maxAngle}(f, P) \) and \( 0^\circ \leq \theta < 90^\circ \).

Upper arc of a facility \( f \) w.r.t. a partition \( P \) is the arc centered at \( q \) with radius \( \frac{\text{dist}(f, q)}{2 \cos(\theta)} \) where \( \theta = \text{maxAngle}(f, P) \) and \( 0^\circ \leq \theta < 90^\circ \). The radius of the upper arc is denoted as \( r^U_{f, P} \) (or simply \( r^U \)). If \( \theta \geq 90^\circ \), \( r^U_{f, P} = \infty \).

The \( k \)-th smallest upper arc of a partition \( P \) is called its bounding arc. The radius of the bounding arc of a partition \( P \) is denoted as \( r_{B,P} \) (or simply \( r_B \)). If the partition contains less than \( k \) upper arcs, \( r_{B,P} = \infty \). A facility \( f \) is called a significant facility of a partition \( P \) if it prunes at least one point \( p \in P \) lying inside the bounding arc of \( P \).

A facility \( f \) cannot prune a point \( p \in P \) for which \( \text{dist}(p, q) \leq \frac{\text{dist}(f, q)}{2 \cos(\theta)} \) where \( \theta = \text{minAngle}(f, P) \) and \( 0^\circ \leq \theta < 90^\circ \). A facility \( f \) cannot prune any point \( p \in P \) if \( \text{minAngle}(f, P) \geq 90^\circ \).
Chapter 2. Related Work

Lower arc of a facility $f$ w.r.t. a partition $P$ is the arc centered at $q$ with radius $\frac{\text{dist}(f, q)}{2 \cos(\theta)}$ where $\theta = \min\text{Angle}(f, P)$. The lower arc is denoted as $r^L_{f,P}$ (or simply $r^L$). A facility $f$ cannot prune any point $p \in P$ that lies inside the lower arc.

A facility $f \in P$ cannot be a significant facility of $P$ if $\text{dist}(f, q) > 2r_B$. A facility $f \notin P$ cannot be a significant facility if $\text{dist}(M, f) > r_B$ and $\text{dist}(N, f) > r_B$ where $M$ and $N$ are the points where the bounding arc of $P$ intersects the boundary of $P$.

In the pruning phase, the space around $q$ is divided into $t$ equally sized partitions. The algorithm utilizes a min-heap $h$ which is initialized by inserting the root of the R-tree that indexes the set of facilities. The entries are de-heaped iteratively from the heap. If the de-heaped entry cannot contain any significant facility for any partition, we ignore it because it cannot prune space in any partition.

If $e$ may contain a significant facility, it is processed as follows. If $e$ is an intermediate or leaf node, every child $c$ of $e$ is inserted in the heap with its key set to $\text{mindist}(q, c)$. The algorithm accesses the entries in ascending order of $\text{mindist}(q, c)$ because the facilities that are closer to the query are expected to prune more area. If $e$ is a data object, it is used to prune the space. The algorithm terminates when the heap becomes empty.

In the verification phase, the users that do not lie in the pruned area are short listed and are called candidate users. The verification algorithm iteratively accesses the nodes of the R-tree that indexes the users. If an entry $e$ lies completely in the pruned area, it is ignored. Otherwise, if $e$ is an intermediate or leaf node, its children are accessed iteratively. If $e$ is a data object and does not lie in the pruned area, it is called a candidate object and is verified.

A user $u$ is verified as follows. The algorithm accessed the $\text{sigList}$ of the parti-
tion $P$ in which the user $u$ lies. The facilities in $\text{sigList}$ are accessed in ascending order of the radius of their lower arcs. A counter is initialized to zero. This counter records the number of facilities that prune $u$ and is incremented whenever the accessed facility $f$ is found to prune $u$, i.e., $\text{dist}(u, f) < \text{dist}(u, q)$. If the counter is at least equal to $k$, the algorithm returns false because $u$ is not R$k$NN of $q$. At any stage, if $\text{dist}(u, q) \leq r_{L, P}^f$, for the accessed facility $f$, the user is confirmed as R$k$NN and the algorithm returns true. This is because counter is less than $k$ and none of the remaining facilities can prune $u$. Also, if the counter remains less than $k$ after processing all facilities in $\text{sigList}$, the algorithm confirms $u$ as R$k$NN and returns true.

### 2.3.3 R$k$NN based top-$n$

Given two types of data $G$ and $C$, the R$k$NN based top-$n$ query, retrieves $n$ data points from $g$, which have the largest number of R$k$NNs.

Li et. al [LWHC14] construct the Voronoi Diagram of $g$ as the index structure. Given a set of data points $G$ in a plane, the Voronoi Diagram of $g$ partitions the plane into cells (named VD-cells), each associated with an individual data point (named VD-node) in $G$.

Given a VD-node $VD_i$, the VD-cells adjacent to $VD_i$-cell are defined as the neighbors of $VD_i$-cell. Given a VD-node $VD_i$, each VD-cell in the Voronoi Diagram of $G$ is associated with a layer number regarding $VD_i$. $VD_i$-cell is defined as the only VD-cell at layer-0 regarding $VD_i$. The neighbors of $VD_i$-cell are defined as being at layer-1 regarding $VD_i$. Then, regarding the VD-cells at layer-0 and layer-1 if $VD_i$ as an aggregate cell, the neighbors of the aggregate cell are defined as being at layer-2 regarding $VD_i$. The layer numbers of other VD-cells are defined similarly.
Given a data point \( p \) belonging to \( C \) in \( VD_i \)-cell, the line segment connecting \( p \) to another VD-node, say \( VD_j \), where \( VD_i \neq VD_j \), may pass through some VD-cells. If \( VD_m \)-cell is one of such VD-cells, where \( VD_m \neq VD_i \) and \( VD_m \neq VD_j \), then \( \text{dist}(p, VD_m) < \text{dist}(p, VD_j) \).

Given a data point \( p \in C \), the line segment connecting \( p \) to another VD-node, say \( VD_j \), may pass through some VD-cells. If \( VD_j \) is in \( BkNN_{\text{ans}}(p, C, G) \), any of the VD-nodes corresponding to the VD-cells passed by the line segment must also be in \( BkNN_{\text{ans}}(p, C, G) \).

Given two VD-nodes \( VD_i \) and \( VD_j \in G \) and a data point \( p \) belonging to \( C \) in \( VD_i \)-cell. \( VD_j \notin BkNN_{\text{ans}}(p, C, G) \) if \( VD_j \)-cell is at layer-\( h \) regarding \( VD_i \), where \( k \leq h \). \( p \notin BRkNN_{\text{ans}}(VD_j, G, C) \) if \( VD_j \)-cell is at layer-\( h \) regarding \( VD_i \), where \( k \leq h \). \( BRkNN_{\text{ans}}(VD_i, G, C) \cap BRkNN_{\text{ans}}(VD_j, G, C) = \emptyset \) if \( VD_j \)-cell is at layer-\( h \) regarding \( VD_i \), where \( k \leq h \).

Let \( VD_i \) be a VD-node in \( G \) and \( U \) be the set of all the VD-nodes from layer-0 to layer-\( k \) regarding \( VD_i \). Given a data point \( p \in C \), if \( p \in BRkNN_{\text{ans}}(VD_i, U, C) \), then \( p \in BRkNN_{\text{ans}}(VD_i, G, C) \).

Pre-processing is required. The Voronoi Diagram of \( G \) is constructed. The VD-cell in the Voronoi Diagram of \( g \) for each data point in \( C \) is recognized. For each VD-node, say \( VD_i \), a list of the other VD-nodes are kept to identify the neighbors of the \( VD_i \)-cell, called the neighbor list of \( VD_i \). For each VD-node, say \( VD_i \), the Euclidean distance between \( VD_i \) and some other VD-nodes, say \( VD_j \), up to layer-\( m \) regarding \( VD_i \), are pre-computed and kept sorted in increasing order of distances.

Similar to FINCH, Li et. al [LWHC14] proposed an approach to dealing with the BRkNN query, which is also a filter-refinement based algorithm. Given a VD-node \( VD_i \), if a VD-node \( VD_j \) may influence \( BRkNN_{\text{ans}}(VD_i, G, C) \), we consider constructing \( \perp (VD_i, VD_j) \) to partition the plane for recognizing which data points
in $C$ are closer to $VD_i$ than $VD_j$. The filter phase stops when all of the VD-nodes contained in the current candidate region have been used to construct the bisectors regarding $VD_i$. If a data point $p \in C$ falls into the final candidate region of $BRkNN(VD_i, G, C)$, by computing the Euclidean distance between $p$ and each VD-node at the layer-$h$ regarding $VD_i$, where $h \leq k$, we can confirm whether $p$ is contained in $BRkNN_{and}(VD_i, G, C)$.

**Voronoi Diagram based algorithm.** First, we randomly select $n$ VD-nodes and compute their exact $BRkNN$ values. Then, a threshold $T$ is set to the smallest value among the $n$ computed $BRkNN$ values. For each VD-node $VD_i$ with an unknown $BRkNN$ value, the upper bound of its $BRkNN$ value is computed. If the upper bound is smaller than $T$, $VD_i$ cannot be one of the results, thus pruned. Otherwise, the exact $BRkNN$ value of $VD_i$ is obtained and then, the newly obtained $BRkNN$ value is checked to see whether it can replace the current results. If yes, the threshold $T$ is updated accordingly.
Chapter 3

Efficiently Computing Reverse $k$ Furthest Neighbors

This chapter studies the problem of efficiently computing reverse $k$ furthest neighbors. Given a set of facilities $F$, a set of users $U$ and a query facility $q$, a reverse $k$ furthest neighbors (R$k$FN) query retrieves every user $u \in U$ for which $q$ is one of its $k$-furthest facilities. R$k$FN query is the natural complement of reverse $k$-nearest neighbors (R$k$NN) query that returns every user $u$ for which $q$ is one of its $k$-nearest facilities. While R$k$NN query returns the users that are highly influenced by a query $q$, R$k$FN query aims at finding the users that are least influenced by a query $q$. R$k$FN query has many applications in location-based services, marketing, facility location, clustering, and recommendation systems etc. While there exist several algorithms that answer R$k$FN query for $k = 1$, we are the first to propose a solution for arbitrary value of $k$. Based on several interesting observations, we present an efficient algorithm to process the R$k$FN queries. We also present a rigorous theoretical analysis to study various important aspects of the problem and our algorithm. An extensive experimental study is conducted using both real and
synthetic data sets, demonstrating that our algorithm outperforms the state-of-the-art algorithm even for \( k = 1 \). The accuracy of our theoretical analysis is also verified by the experiments.

This chapter is structured as follows. Section 3.1 present preliminaries. Specifically, we describe how to extend the state-of-the-art work on reverse furthest neighbors queries to reverse \( k \) furthest neighbors queries. This extension would be used as our baseline algorithm. Section 3.2 presents an overview of our algorithm. Our techniques are presented in Section 3.3 followed by theoretical analysis in Section 3.4. An extensive experimental study is provided in Section 3.5.

3.1 Preliminaries

**Extending CHFC for arbitrary value of \( k \).** CHFC can be extended to answer R\( k \)FN queries for arbitrary value of \( k \). We define \( k \)-th **convex hull** as the convex hull of only the facilities that lie within \((k - 1)\)-th convex hull for \( k > 1 \), where \( k = 1 \) corresponds to the usual convex hull. Fig. 2.1 shows first two convex hulls, i.e., the outer polygon is the 1st convex hull and the inner polygon is the 2nd convex hull. It is easy to prove that a query \( q \) cannot have any R\( k \)FN if it lies within the \( k \)-th convex hull. For example, \( f_8 \) cannot have any R2FN because it lies within the 2nd convex hull. To answer a R\( k \)FN query, CHFC computes \( k \)-th convex hull, and it computes result in the same way of computing FVC if \( q \) lies on or outside \( k \)-th convex hull.

Note that CHFC can not identify that \( f_6 \) and \( f_7 \) are futile for \( k = 2 \) since they do not lie inside the 2nd convex hull. In contrast, we use a much stronger condition that is able to identify \( f_6 \) and \( f_7 \) as futile. Furthermore, computing \( k \)-th convex hull is quite expensive, but the condition used by us can be applied at a much lower
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

cost.

3.2 Overview

Our algorithm has three phases. In the first phase, we access the set of facilities only, and exploit certain problem characteristics to efficiently identify whether a query is futile or not. A query is futile if it is guaranteed not to have any R$k$FN regardless of users’ locations. The algorithm terminates if the query is futile. Furthermore, the proposed technique to identify futile queries is tight, i.e., we prove that a query is futile if and only if it satisfies the proposed criteria.

If the query is not futile, our algorithm enters into the pruning phase where search space that cannot contain R$k$FNs is pruned. This is done by realizing a novel application of $k$-depth contour [CSLZ14] which has been used earlier in Computational Geometry and Statistics to solve different problems. First, we prove that only shallow facilities, namely the facilities that lie on or outside $k$-depth contour are required to answer the query. The expected number of shallow facilities is $O(k \log |F|)$, where $|F|$ is the total number of facilities. This significantly reduces the I/O and CPU cost of our algorithm. Second, we propose two cheap yet effective pruning strategies. The first pruning strategy utilizes the vertices of $k$-depth contour in contrast to facilities used by previous techniques. The second pruning strategy adopts a brand new pruning paradigm and calculates sweeping regions based on ranges of angles in a polar coordinate system centered at the query.

Finally, in the verification phase, the users that lie in unpruned area are identified. These candidate users may or may not be the R$k$FNs of the query. We introduce the concept of shadow, where every user that lies in this area is guaranteed to be a R$k$FN of the query. The majority of the users are either pruned
during the pruning phase or are confirmed by shadow. The remaining candidate
users are verified one by one by checking whether $q$ is one of its $k$-furthest facilities
or not. This can be done easily using only the shallow facilities.

3.3 Techniques

Some queries cannot have any R$k$FN regardless of users’ locations. These are
futile queries. Efficiently identifying futile queries can avoid unnecessary pruning
and verification phases, and hence reduces computational cost significantly. We
present such techniques in Section 3.3.1. If a query is not futile, we use several
pruning strategies to effectively prune the search space that cannot contain any
R$k$FN. These pruning strategies are presented in Section 3.3.2. Users lie in the
unpruned space may be R$k$FNs. They are candidate users. We present techniques
to efficiently verify whether a candidate user is a R$k$FN or not in Section 3.3.3.

3.3.1 Determine futile query

Given a query $q$, a facility $f$, and a point $p$, if $\text{dist}(p, f) > \text{dist}(p, q)$, then $p$ is
disqualified from being RFN of $q$ due to $f$, and we say $f$ prunes $p$. When pruned
by at least $k$ facilities, $p$ cannot be R$k$FN. We divide the space around $q$ in four
quadrants $Q_1$ to $Q_4$ as shown in Fig. 3.1.

**Lemma 3.1.** A query $q$ cannot have any R$k$FN if every quadrant has at least $k$
facilities.

**Proof.** (Fig. 3.1) Without loss of generality, we prove that the quadrant $Q_3$ cannot
have any R$k$FN. Consider a point $p$ in $Q_3$ and a facility $f$ in $Q_1$. Let $p_x$ and $p_y$ be
$x$ and $y$ coordinates of $p$. For any facility $f$ in $Q_1$, we have $|p_x - f_x| > |p_x - q_x|$ and
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

Figure 3.1: Quadrant test

Figure 3.2: P-ray pruning ($k = 2$)

$$|p_y - f_y| > |p_y - q_y|.$$ Hence, $\text{dist}(p,f) > \text{dist}(p,q)$. If there are at least $k$ facilities in $Q_1$, $q$ cannot be among the $k$-furthest facilities of $p$. \hfill \Box$

Let $L_q$ be a line passing through $q$. It divides the space in two halves. Let $H_q$ denote one of the two half-spaces. We define the P-ray (perpendicular ray) of $H_q$ to be the ray that is perpendicular to $L_q$ and whose only point in $H_q$ is $q$ (see the dashed ray in Fig. 3.2(a)).

**Lemma 3.2.** Let $f$ be a facility in $H_q$ and $p$ be a point on the P-ray of $H_q$. $f$ prunes $p$ (i.e., $\text{dist}(p,f) > \text{dist}(p,q)$).
Chapter 3. Efficiently Computing Reverse \( k \) Furthest Neighbors

Proof. (Fig. 3.2(a)) Let \( x \) be the point where the line \( pf \) intersects \( L_q \). As either \( \triangle xqp \) is a right angle triangle or \( x \) lies on \( q \), we have \( \text{dist}(p, x) \geq \text{dist}(p, q) \). Hence, \( \text{dist}(p, f) > \text{dist}(p, q) \).

We define the **depth** of a half-space \( H_q \) as the number of facilities that lie in \( H_q \) and denote it by \( |H_q| \). In Fig. 3.2(a), \( H_q \) contains facilities \( q, f_1 \) and \( f_3 \) and its depth \( |H_q| = 3 \).

**Lemma 3.3.** A point \( p \) on the P-ray of \( H_q \) cannot be \( RkFN \) of \( q \) if \( |H_q| > k \).

Proof. Since \( |H_q| > k \), there are at least \( k \) facilities prunes \( p \) (Lemma 3.2). Therefore, \( p \) cannot be \( RkFN \) of \( q \).

Now assume \( L_q \) (i.e., \( H_q \)) is rotated anti-clockwise, and \( H_q \) always contains more than \( k \) facilities (at least \( k \) other facilities apart from \( q \)). During this process, P-ray of \( H_q \) also rotates and sweeps an area. According to Lemma 3.3, no point in this area can be \( RkFN \) of \( q \). If the P-ray sweeps the whole space (i.e., rotated \( 360^\circ \)), and \( H_q \) always contains more than \( k \) facilities, we know \( q \) is futile. Assume \( k = 2 \) in Fig. 3.2(b), \( H_q \) is rotated \( \theta^\circ \) anti-clockwise, and it always contains at least 3 facilities, namely \( \{q, f_1, f_3\} \), \( \{q, f_1, f_2, f_3\} \) or \( \{q, f_1, f_2\} \). The shaded area is swept by the P-ray and cannot contain any \( R2FN \). Now we adapt the concept of location depth in Computational Geometry to our problem, and define location depth of \( q \) as the minimum number of facilities in \( H_q \) when \( H_q \) is rotated \( 360^\circ \).

**Definition 3.1.** **Location depth** of a query point \( q \) (denoted as \( |q| \)) is the minimum depth of any half-space \( H_q \) defined using a line \( L_q \) passing through \( q \).

Fig. 3.3(a) shows nine facilities (\( q \) and \( f_1 \) to \( f_8 \)) and an arbitrary point \( p \). The location depth of \( q \) is \( |q| = 2 \) because the number of facilities in the half-space defined by the dotted line is 2 (\( q \) and \( f_3 \)) which is minimum. Similarly, location
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

(a) Location depth

(b) $q$ is 2FN of at least one point

Figure 3.3: Location depth and its relationship with R$k$FN

depth of $f_6$ is $|f_6| = 3$ (see the half-space defined by the solid line that contains $f_1$, $f_2$ and $f_6$). Note that the location depth of a facility is at least one because each half-space contains the facility itself. In contrast, the location depth of an arbitrary point $p$ may be zero. For example, in Fig. 3.3(a), $|p| = 0$ (see the half-space defined by the dashed line).

Lemma 3.4. A query facility $q$ cannot have any R$k$FN if $|q| > k$.

Proof. Let $p$ be an arbitrary point, and $H_q$ be a half-space whose P-ray passes through $p$. Since $|q| > k$, we have $|H_q| > k$. According to Lemma 3.3, $p$ cannot be R$k$FN of $q$.

Next we show the tightness of Lemma 3.4 by proving that $q$ cannot be futile if $|q| \leq k$.

Lemma 3.5. If $|q| \leq k$, there exists at least one point $p$ which is R$k$FN of $q$.

Proof. Since $|q| \leq k$, there is at least one half-space $H_q$ that contains at most $k$ facilities. Assume $k = 2$ in Fig. 3.3(b), $H_q$ (white area) contains only two facilities, $q$ and $f_3$. It can be proved that there exists a point $p$ for which $q$ is its furthest
facility among the facilities lying in the shaded area. Since the number of facilities in $H_q$ is at most $k$ including $q$, there are at most $(k - 1)$ facilities that prune $p$, i.e., $p$ is R$k$FN of $q$.

We construct a convex hull (the polygon) using $q$ and all the facilities that lie outside $H_q$. It is clear that $q$ is one of the vertices of the convex hull. We construct the furthest Voronoi cell of $q$ using the facilities that lie on or inside this convex hull. Since $q$ is a vertex on the convex hull, the furthest Voronoi cell of $q$ cannot be empty [OBSC99]. This implies that there exists at least one point $p$ for which $q$ is the furthest neighbors among the facilities that lie outside $H_q$.

Lemma 3.4 and 3.5 state that a query $q$ is futile if and only if $|q| > k$. We remark that although a query $q$ that is not futile has at least one R$k$FN point, $q$ may or may not have a R$k$FN user. In other words, the set that contains all R$k$FN users of a query $q$ may be empty even if $q$ is not futile. Next, we present our pruning strategies to prune the search space that cannot contain any R$k$FN of the query after we have determined that the query is not futile.

### 3.3.2 Pruning the search space

#### 3.3.2.1 Limitations of adopting half-space pruning

*Half-space pruning* [CLZZ11, TPL04] is a well known pruning strategy used for processing R$k$NN queries. Although it is possible to apply this technique for R$k$FN queries, it suffers from certain limitations.

The perpendicular bisector between a facility $f$ and a query $q$ divides the space into two halves. Each half is called a *bisection*. We use $B_{f,q}$ to denote the bisection that contains $f$ and $B_{q,f}$ to denote the bisection that contains $q$. For any point
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

$p \in B_{q,f}$, we have $dist(p, f) > dist(p, q)$, i.e., $f$ prunes $p$ and $p$ cannot be RFN of $q$. In other words, we say that the bisection $B_{q,f_1}$ is pruned. Clearly, the intersection of at least $k$ such bisections is pruned by at least $k$ facilities, and cannot contain R$k$FN of $q$. Fig. 3.4 shows an example. The point $p$ lies in the bisection $B_{q,f_1}$, and $f_1$ prunes $p$. Assuming $k = 2$, the shaded area can be pruned because it is the intersection of two bisections, namely $B_{q,f_1}$ and $B_{q,f_2}$.

Based on the above pruning strategy, we can consider every facility $f \in F$ and prune the space that is intersection of at least $k$ bisections. However, this strategy suffers from two serious limitations: 1) the number of facilities considered for pruning may be quite large and considering all bisections is prohibitively expensive; 2) even if the number of facilities (i.e., bisections) used for pruning is not large, the pruning becomes quite expensive especially when $k$ is not small. The reasons are similar to those mentioned in [YCLZ14], e.g., given $n$ bisections, it is quite expensive to determine the search space that is pruned by at least $k$ bisections.

3.3.2.2 Discarding un-necessary facilities

To address the first limitation, we discard un-necessary facilities, and keep only the minimum required to correctly compute the results.

**Definition 3.2.** A facility $f$ that has $|f| \leq k$ is called a shallow facility. In contrast, a facility $f'$ is called a deep facility if $|f'| > k$.

**Lemma 3.6.** R$k$FN of a query $q$ can be correctly computed using only the set of shallow facilities.

**Proof.** Let $S \subseteq F$ be the set that contains all the shallow facilities. We prove that for any point $p, q$ is one of $p$’s $k$-furthest facilities in $F$ (i.e., $p$ is R$k$FN of $q$) i) if and ii) only if $q$ is one of $p$’s $k$-furthest facilities in $S$. 

Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

i) Let $q$ be one of the $k$-furthest facilities of $p$ in $S$. If $q$ is not one of $p$’s $k$-furthest facilities in $F$, there is at least one deep facility $f$ which is among $p$’s $k$-furthest facilities in $F$. Since $f$ is a deep facility, we have $|f| > k$. According to Lemma 3.4, $f$ cannot have any R$k$FN, i.e., $f$ cannot be among $p$’s $k$-furthest facilities. This contradicts the assumption.

ii) Since $S \subseteq F$, it is clear that $q$ is one of the $k$-furthest facilities of $p$ in $F$ only if it is one of the $k$ furthest facilities of $p$ in $S$.\[ \square \]

Lemma 3.6 implies that we only need to consider the shallow facilities to compute R$k$FN of a query $q$. A major challenge in using Lemma 3.6 is how to identify the shallow facilities efficiently. A straightforward solution is to compute the location depth of each facility. However, this is quite expensive because it requires accessing the whole data set at least once for each facility. We observed that the set of shallow facilities can be efficiently identified with the help of $k$-depth contour [CSY84, MRR+01] (also known as $k$-hull) which has been used in the past for different problems in Computational Geometry and Statistics.

**Definition 3.3.** $k$-depth contour is a convex polygon such that, for every point $p$, 1) $|p| \geq k$ if $p$ lies on or inside this polygon and 2) $|p| < k$ if $p$ lies strictly outside this polygon.

Fig. 3.5 shows 9 facilities ($q$ and $f_1$ to $f_8$). 2-depth contour is the convex polygon. Note that the vertices of $k$-depth contour may or may not be the facility objects. The dotted lines define the shape of the 2-depth contour. Each dotted line defines a half-space with depth equal to $k + 1$ such that even a slight rotation reduces the depth to 2. We remark that $k$-depth contour is the same as convex hull only when $k = 1$.

**Lemma 3.7.** A facility $f$ that lies strictly inside the $k$-depth contour has $|f| > k$. 
Proof. We show that every half-space $H_f$ defined by a line $L_f$ passing through $f$ has $|H_f| > k$. Since $f$ lies strictly inside the $k$-depth contour, the line $L_f$ passes through the $k$-depth contour. As we show later in Lemma 3.10, $|H_f| > k$ for every such half-space. Hence, $|f| > k$. \qed

According to Lemma 3.7, to obtain the shallow facilities, we can compute $k$-depth contour and identify the facilities that lie on or outside $k$-depth contour. In Fig. 3.5, the facility $q$ lies on the 2-depth contour and is a shallow facility ($|q| = 2$). The set of shallow facilities consists of $q$ and $f_1$ to $f_5$. The facilities $f_6$, $f_7$ and $f_8$ are not shallow facilities and can be discarded when computing R$k$FN ($k = 2$) of any query.

Lemma 3.6 and 3.7 significantly reduce the number of facilities required for computing R$k$FN queries. It was shown [CSLZ14] that the number of facilities that lie on or outside $k$-depth contour is bounded by the cardinality of $k$-skyband [PTFS05]. Let $|F|$ be the total number of facilities. Assuming that the two location coordinates are independent of each other, the expected cardinality of $k$-skyband is $O(k \log |F|)$ [CLZZ11]. Hence, the expected number of shallow facilities is $O(k \log |F|)$. 

Figure 3.4: Half-space pruning

Figure 3.5: $k$-depth contour ($k = 2$)
3.3.2.3 Implementing the ideas presented so far

We use the state-of-the-art algorithms to compute $k$-depth contour [CSLZ14]. The algorithms utilize the index (e.g., R-tree) constructed on the set of facilities and efficiently return the $k$-depth contour. Since $k$-depth contour is always a convex polygon, shallow facilities can be efficiently identified using R-tree (i.e., open only the nodes intersect with the boundary of $k$-depth contour).

During the computation, we maintain a counter $c_i$ for the number of facilities seen in each quadrant $Q_i$. If $c_i \geq k$ for each of the four quadrants, we terminate the algorithm as the query is futile (Lemma 3.1). While the algorithm iteratively computes the $k$-depth contour, we terminate the algorithm whenever it implies that $|q| > k$ and $q$ is futile (Lemma 3.4).

Next, we present two cheap yet effective pruning strategies.

3.3.2.4 Pruning using $k$-depth contour

To address the second limitation (Section 3.3.2.1), we show that instead of using at least $k$ facilities (i.e., bisections) to prune a point $p$, we can use the vertices of the $k$-depth contour, and that a point $p$ is pruned if one vertex prunes it.

**Lemma 3.8.** Let $v$ be a vertex$^1$ of the $k$-depth contour, no point in $B_{q,v}$ can be $RkFN$ of $q$.

**Proof.** Consider the example in Fig. 3.6(a)$^2$. The shaded area illustrates the bi-

---

$^1$Note that $q$ may lie on a vertex of the $k$-depth contour as shown in Fig. 3.6(a). This lemma does not apply to the vertex on which $q$ lies.

$^2$Although Fig. 3.6(a) shows an example where $q$ lies on $k$-depth contour, Lemma 3.8 and its proof hold regardless of whether $q$ lies on $k$-depth contour or not.
section $B_{q;v_1}$. Let $p$ be a point in $B_{q;v_1}$, we prove that $p$ cannot be R$k$FN of $q$. Let $H_{v_1}$ be the half-space that has $p$ on its P-ray. Similar to Lemma 3.2, every facility $f$ in $H_{v_1}$ has $\text{dist}(p, f) > \text{dist}(p, v_1)$. Since $p$ lies in the bisection $B_{q;v_1}$, we have $\text{dist}(p, v_1) > \text{dist}(p, q)$. Hence, for every facility $f$ in $H_{v_1}$ we have $\text{dist}(p, f) > \text{dist}(p, q)$, i.e., $f$ prunes $p$. By the definition of $k$-depth contour, $|v_1| \geq k$. Hence, $|H_{v_1}| \geq k$, there are at least $k$ such facilities that prunes $p$, and $p$ cannot be R$k$FN of $q$. \hfill $\square$

We use the bisection $B_{q;v_i}$ for each vertex $v_i$ on the $k$-depth contour to iteratively prune the search space. The space that cannot be pruned is called **candidate polygon** because a user that lies in this space is a candidate user. In the example of Fig. 3.6(a), the candidate polygon is initialized to the data space (the square), and is updated to the white area after the vertex $v_1$ is used for pruning. Then, in Fig. 3.6(b), the candidate polygon is further updated to the white area after the vertex $v_2$ is used for pruning.

We observed that certain vertices of $k$-depth contour are not necessary to be considered for pruning. For example, in Fig. 3.6(b), the bisection $B_{q;v_4}$ (dashed
line) does not prune any point in the current candidate polygon. In other words, $v_4$ can be discarded after $v_1$ and $v_2$ have been used for updating the candidate polygon. The same can be shown for the vertex $v_3$.

**Lemma 3.9.** A vertex $v$ on $k$-depth contour cannot prune any point $p$ in the candidate polygon if, for every corner $c_i$ of the current candidate polygon, $\text{dist}(c_i, q) \geq \text{dist}(c_i, v)$.

**Proof.** (Fig. 3.6(b)) We prove this by contradiction. Assume $p$ can be pruned by $B_q$, i.e., $\text{dist}(p, q) < \text{dist}(p, v)$. Since candidate region is a polygon, $B_q$ prunes a point $p$ inside it only if it prunes at least one corner $c_i$ of the candidate region, i.e., $\text{dist}(c_i, q) < \text{dist}(c_i, v)$. This contradicts the condition: $\text{dist}(c_i, q) \geq \text{dist}(c_i, v)$ for every corner $c_i$. \hfill \square

Algorithm 4 presents the details of computing the candidate polygon using the $k$-depth contour. For efficiency, we index the vertices of the $k$-depth contour in a main-memory R-tree$^3$. The algorithm iteratively accesses the entries of this R-tree and ignores the entry if each vertex $v$ in an entry $e$ satisfies the condition specified in Lemma 3.9. Specifically, if $\text{dist}(c_i, q) \geq \text{maxdist}(c_i, e)$ for each corner $c_i$ of the candidate polygon then all vertex $v$ in the entry $e$ can be discarded. Otherwise, the entry $e$ may contain some vertices that are required to update the current candidate polygon. The rest of the algorithm is self-explanatory.

### 3.3.2.5 Pruning using shallow facilities

$^3$Although the expected number of vertices of $k$-depth contour is small, it can be much larger, e.g., the convex hull may contain $|F|$ vertices. In this case, the R-tree can be stored in secondary memory.
Algorithm 4: Candidate Polygon

**Input**: $q$: the query point; an R-tree that indexes the vertices of $k$-depth contour

**Output**: $CP$: candidate polygon

1. initialize $CP$ as the boundary of data space;
2. insert root of the R-tree in a queue $Q$;
3. **while** $Q$ is not empty **do**
   4. dequeue an entry $e$;
   5. **for** each corner $c_i$ of $CP$ **do**
      6. **if** $\text{dist}(c_i, q) < \text{maxdist}(c_i, e)$ **then**
         7. mark $e$ as unpruned; break;
   8. **if** $e$ is unpruned **then**
      9. **if** $e$ is an intermediate or leaf node **then**
         10. insert children of $e$ in $Q$;
      11. **else**
         12. use $B_{q,e}$ to update $CP$;
13. **return** $CP$;
According to Lemma 3.3, P-ray of a half-space $H_q$ cannot contain any $R_k$FN if $|H_q| > k$. In other words, a P-ray may contain $R_k$FN only if $|H_q| \leq k$. This property can be used to prune the search space that cannot contain any $R_k$FN.

**Lemma 3.10.** Let $L_q$ be a line passes through the $k$-depth contour and the query facility $q$. The half-space $H_q$ defined by $L_q$ has $|H_q| > k$.

*Proof.* (Fig. 3.7(a) illustrates an example of $k = 2$, where 2-depth contour is the polygon shown in thick line.) Let $L_q$ be a line (dotted line) that passes through $k$-depth contour and defines $H_q$. Let $p$ be a point (shown as a star) that is inside $k$-depth contour and lies on $L_q$. By the definition of $k$-depth contour, $|p| \geq k$. This implies that $(|H_q| = |H_p|) \geq k$. Now, assume $|H_p| = k$. Since $q$ lies on $H_p$, an infinitely small rotation can exclude $q$ from $H_p$ resulting in $|H_p| < k$. However, this is not possible because $|p| \geq k$ as per the definition of $k$-depth contour. Hence, we have $|H_p| > k$, and $|H_q| > k$. \hfill $\square$

According to Lemma 3.10, we only need to consider the half-spaces $H_q$ that do not pass through the $k$-depth contour. Hence, we draw the two lines that pass through $q$ and are tangent to the $k$-depth contour, and we consider only the half-spaces between these two lines. For example, assume $k = 2$ in Fig. 3.7(a), the two tangent lines are $qf_5$ and $qf_2$. Let $H_q$ be the half-space that is defined by the line passing through $q$ and $f_5$ and contains $f_4$. We rotate $H_q$ anti-clockwise until $H_q$ contains $qf_2$. During the rotation, we identify the space for which $|H_q| \leq k$ (shaded areas). For instance, the half-space defined by the broken line in the shaded area contains only two facilities $q$ and $f_4$.

As we rotate $H_q$, its P-ray also rotates and sweeps the search space. The area swept by P-ray while $|H_q| \leq k$ holds is called **sweeping region**. In the example of Fig. 3.7(b), when $H_q$ is rotated in the shaded area marked as $R1$, its P-ray sweeps...
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

(a) Pruning by rotating $H_q$

(b) Sweeping regions

Figure 3.7: Pruning using shallow facilities

the dotted area marked as $S_1$, and when $H_q$ is rotated in the shaded area marked as $R2$, its P-ray sweeps the dotted area marked as $S2$. $S1$ and $S2$ are the sweeping regions. Any point $p$ that does not lie in $S1$ or $S2$ cannot be R$k$FN of $q$. In other words, only the sweeping regions can contain R$k$FN.

To identify sweeping regions, we first compute the two tangent lines as aforementioned. Then, we pick one of the tangent lines and count the number of facilities in $H_q$. We maintain the counter for $|H_q|$ as we rotate it: $|H_q|$ is incremented when a facility enters $H_q$ and decremented when a facility leaves $H_q$. Let $x$ be a point on the horizontal line passing through $q$ and lies on the right of $q$, and $S$ be the set of shallow facilities, we sort all $f \in S$ by $\angle xqf$, and the next facility encountered during the rotation can be accessed sequentially.

Note that the candidate polygon (Section 3.3.2.4) and sweeping regions are inherently different. Fig. 3.8 gives an example of both candidate polygon (shaded area) and sweeping regions (dotted areas). R$k$FN of a query can be found only in the space that cannot be pruned by both pruning strategies (where dotted areas overlap with shaded area).

Next, we present techniques to further prune the area in sweeping regions.
Definition 3.4. (Fig. 3.9) Let $L$ and $L'$ be the bounding lines of a sweeping region, and $H_q$ ($H'_q$) be the half-space such that $L$ ($L'$) is its $P$-ray. The area contained by both $H_q$ and $H'_q$ (i.e., $H_q \cap H'_q$) is the pruner region of this sweeping region. The set of facilities that lie in the pruner region are the pruners of this sweeping region.

According to Lemma 3.2, every point $p$ in a sweeping region is pruned by each of its pruners. In Fig. 3.9, the pruner region of sweeping region $S1$ is the shaded area, and the pruners of $S1$ is $\{f_4\}$. $f_4$ prunes every point $p$ in the sweeping region.

Lemma 3.11. Let $P$ be the set of pruners of a sweeping region. If $P$ contains $k - 1$ facilities, a point $p$ in the sweeping region cannot be R$k$FN if it is also pruned by a facility $f \notin P$.

Proof. The point $p$ is pruned by every facility $f' \in P$ and a facility $f \notin P$. Since $P$ contains at least $k - 1$ facilities, $p$ is pruned by at least $k$ facilities and cannot be R$k$FN of $q$. 

Fig 3.10 shows the same example of Fig. 3.9, the set of pruners of the sweeping region contains one facility $f_4$. We draw the perpendicular bisector between $f_3$ and
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

$q$. Assuming $k = 2$, according to Lemma 3.11, no point $p$ in the bisection $B_{q;f_3}$ can be R$k$FN of $q$ because $p$ is pruned by both $f_4$ and $f_3$ (i.e., the dotted white area of the sweeping region can be pruned). Similarly, the bisection $B_{q;f_5}$ is pruned by $f_4$ and $f_5$. Hence, in this example, $f_3$ and $f_5$ together can prune the whole sweeping region.

![Figure 3.10: Pruning sweeping region](image)

![Figure 3.11: Verification phase](image)

Note that, whenever the data space (i.e., the square in our example) is pruned, or in other words, unpruned region can only exist outside the data space, our algorithm can be terminated as the query cannot contain any R$k$FN.

3.3.3 Verification

Users lie in the unpruned space can be R$k$FN and is called candidate users. Each candidate user is verified by checking whether $q$ is one of its $k$-furthest facilities or not.

According to Lemma 3.6, we only need the set of shallow facilities $S$ to correctly compute the results. We index shallow facilities in a main-memory R-tree and access only this R-tree to verify the users. This can be done by issuing boolean $k$-region query. Let $R$ be the space outside the circle centered at $u$ with radius
\( \text{dist}(u, q) \) (Fig. 3.11 shows part of such a circle), a boolean \( k \)-region query returns true if and only if there are at least \( k \) facilities in \( R \). A user \( u \) is RkFN if and only if the query returns false. Assuming \( k = 2 \), the user \( u \) in Fig. 3.11 is not a RkFN because there are two facilities outside the circle. An optimization of this is to use boolean \( m \)-region query \((m < k)\). This is an optimization since true is returned sooner, especially when the difference between \( m \) and \( k \) is large.

**Lemma 3.12.** Let \( S \) be the set of shallow facilities, and \( P \) be the set of pruners of a sweeping region. If \( P \) contains \( k - m \) \((m \leq k)\) facilities, a point \( p \) in the sweeping region cannot be RkFN if it is also pruned by \( m \) facilities in \( S \setminus P \).

The proof is similar to Lemma 3.11 and is omitted. During the rotation of \( H_q \), we store for each sweeping region its pruner region and the number of its pruners. To verify an user \( u \), we identify the sweeping region that contains \( u \). Let \( R \) be the region that outside both the corresponding pruner region and the circle centered at \( u \) with radius \( \text{dist}(u, q) \), we then issue a boolean \( m \)-region query with region \( R \). Assuming \( k = 2 \) in the example of Fig. 3.11, the pruner region of \( u \) is the shaded area and it contains one facility \( f_4 \) (i.e., \( m = 1 \)). In this example, \( R \) is the space outside the circle excluding the shaded area. As \( k - m = 1 \), we issue a boolean 1-region query with region \( R \), and it returns true as \( f_5 \) lies in \( R \). Hence, \( u \) is not R2FN.

Due to the large number of candidate users, the verification phase could still be very time consuming. Next, we present techniques that help us to identify an area such that candidate users lie in this area are guaranteed to be RkFN and, therefore, do not require verification.

**Lemma 3.13.** (Fig. 3.12(a)) Let \( p \) be RkFN of \( q \), \( \overrightarrow{qp} \) be the ray whose endpoint is \( q \) and passes through \( p \), and \( p' \) be a point on \( \overrightarrow{qp} \) that has \( \text{dist}(p', q) > \text{dist}(p, q) \). \( p' \) is RkFN of \( q \).
Chapter 3. Efficiently Computing Reverse k Furthest Neighbors

Proof. (Fig. 3.12(a)) Let \(\text{Circ}(x)\) denote the circle centered at \(x\) with radius \(\text{dist}(x, q)\). We draw two circles \(\text{Circ}(p)\) and \(\text{Circ}(p')\). Since both \(p\) and \(p'\) lie on \(\overrightarrow{qp}\) and \(\text{dist}(p', q) > \text{dist}(p, q)\), we know that \(\text{Circ}(p')\) contains \(\text{Circ}(p)\). Since \(p\) is RkFN of \(q\) only if there are less than \(k\) facilities outside \(\text{Circ}(p)\), we know that there are less than \(k\) facilities outside \(\text{Circ}(p')\), which implies \(p'\) is RkFN of \(q\). □

![Diagram](image)

(a) Lemma 3.13  
(b) Lemma 3.14

Figure 3.12: Improving verification using shadows

**Definition 3.5.** Let \(e\) be a node entry of the R-tree that indexes users. Let \(p\) be any point in \(e\). The **shadow** of \(e\) consists of every point \(p'\) that lies on the ray \(\overrightarrow{qp}\) and has \(\text{dist}(p', q) \geq \text{dist}(p, q)\). A node \(e\) is a **RkFN node** if every point \(p\) in \(e\) is RkFN of \(q\).

In Fig. 3.12(b), the shadow of the entry \(e\) is shown shaded. Assuming that \(q\) is the light source and \(e\) is an item in its way, the shaded area is the shadow of \(e\).

**Lemma 3.14.** A user \(u\) lies in the shadow of a RkFN node is RkFN.

Proof. Since \(u\) lies in the shadow of \(e\), the line \(\overrightarrow{uq}\) passes through a point \(p\) in \(e\). According to Lemma 3.13, since \(p\) is RkFN of \(q\), \(u\) is also RkFN of \(q\). □
Algorithm 5: Verification

1 insert root of the R-tree in a queue $Q$;

2 while $H_a$ is not empty do

3 dequeue an entry $e$ from $Q$;

4 if $e$ is an intermediate or leaf node then

5 if $e$ is partially inside the unpruned space then

6 insert children of $e$ in $Q$;

7 if $e$ is completely inside the unpruned space then

8 if $e$ is covered by shadows or is R$k$FN node then

9 add users in $e$ to result;

10 update the list of shadows using $e$;

11 else

12 insert children of $e$ in $Q$;

13 else // $e$ is a data object

14 if $e$ lies in one of the shadows then

15 add $e$ to result;

16 else

17 if verified by boolean $m$-region query then

18 add $e$ to result;

19 return result;
Recall that a facility $f$ prunes a point $p$ if $\text{dist}(p, f) > \text{dist}(p, q)$. There exists a point in a R-tree node $e$ that is pruned by a facility $f$ if $\text{maxdist}(e, f) > \text{mindist}(e, q)$. We count the number of such facilities. This represents the maximum number of facilities that can possibly prune a point $p$ in $e$. If the number is less than $k$, then $e$ is a $Rk$FN node. Note that $e$ can be a $Rk$FN node only if it lies entirely in the unpruned space. Otherwise, we do not need to do this check. Algorithm 5 presents the details of our verification algorithm.

### 3.4 Theoretical Analysis

Assume that the data space is a circle with radius $r = 1$ and that facilities and users are uniformly distributed (Note that this is more challenging for $k$-depth contour algorithms [CSLZ14]). Let $|F|$ ($|U|$) be the total number of facilities (users). We analyse the expected area of $k$-depth contour, and the expected area that cannot be pruned by our algorithm.

#### 3.4.1 Expected area of $k$-depth contour

To the best of our knowledge, we are the first to analyse the expected area of $k$-depth contour. This is of stand-alone interest and also helps in analysing various important aspects of our algorithm. For instance, the expected number of shallow facilities and the probability of a random query to be futile. Besides, the I/O cost of our pruning phase corresponds to the cost of computing $k$-depth contour. Since KnightRider [CSLZ14] does not access any node lies completely inside the $k$-depth contour, the expected area of $k$-depth contour also helps in analysing the I/O cost of our pruning phase.

Let $c$ be the center of the circular data space, and $p$ be any point in it. Let $L_p$
be a chord passing through \( p \), and \( H_p \) be the corresponding segment that do not contain \( c \). The area of \( H_p \) is inversely proportional to the minimum distance from \( c \) to \( L_p \), and \( H_p \) takes the minimum area when \( L_p \) is perpendicular to the line \( cp \). As we assume uniform distribution, \(|p|\) is the number of facilities in \( H_p \) when \( H_p \) takes the minimum area. In the example of Fig. 3.13(a), the area of \( H_p \) (shaded segment) is smaller than the area of \( H_p' \) (white segment).

\( \text{Figure 3.13: Area of } k \)-depth contour \)

Recall that every point \( p \) lies on or inside \( k \)-depth contour has \(|p| \geq k \). Consider the point \( p \) in Fig. 3.13(b), let \( H_p \) be the minimum segment that has \( p \) on its chord (shaded segment). \(|p| = k \) if there are \( k \) facilities in \( H_p \). Let \( \theta \) be the central angle of \( H_p \), \( r' \) be the distance from \( c \) to \( p \). \( r' = r \cos \frac{\theta}{2} \) is the radius of \( k \)-depth contour (circle in broken line) because every point \( p' \) farther to \( c \) than \( p \) has \(|p'| < k \) and every point \( p' \) closer to \( c \) than \( p \) has \(|p'| > k \).

First, we compute \( \theta \). In Fig. 3.13(b), let \( A_s \) be the area of \( H_p \), it is equal to the area of the circular sector minus the area of the triangular portion, that is \( A_s = \frac{1}{2}r^2\theta - \frac{1}{2}r^2\sin \theta \). Since \( r = 1 \), we have \( A_s = \frac{1}{2}(\theta - \sin \theta) \). We know that the area of data space is \( \pi r^2 = \pi \) which contains \(|F|\) facilities. Since \( H_p \) contains \( k \) facilities, we expect \( A_s = \frac{kr}{|F|} \). Hence, we have \( \frac{1}{2}(\theta - \sin \theta) = \frac{kr}{|F|} \). We approximate
\( \theta - \sin \theta \) by Eq. (3.1) obtained using nonlinear curve fitting\(^4\).

\[
\theta - \sin \theta \approx e^{a-b/(\theta + c)} \tag{3.1}
\]

where \( a = 3.87031, b = 11.09062 \) and \( c = 0.93407 \). Given \( k \), we can derive Eq. (3.2) from \( \frac{1}{2}(\theta - \sin \theta) = \frac{k\pi}{|F|} \).

\[
\theta = \frac{b}{a - \log_e \frac{2k\pi}{|F|}} - c \tag{3.2}
\]

Now, the area of \( k \)-depth contour denoted by \( A_{kdc} \) can be computed using Eq. (3.3).

\[
A_{kdc} = \pi (r')^2 = \pi \cos^2 \frac{\theta}{2} \tag{3.3}
\]

### 3.4.2 Expected area of sweeping region

Given a query facility \( q \), we analyse the expected area of sweeping region (Section 3.3.2.5). This is used to estimate the expected number of candidate users and the I/O cost of verification phase.

In the example of Fig. 3.14(a), assume the circle in dashed line is the \( k \)-depth contour. The two lines \( qt_1 \) and \( qt_2 \) are tangent to the \( k \)-depth contour. \( t_1 \) and \( t_2 \) are their points of tangency, respectively. The line \( qb_1 \) (\( qb_2 \)) is the P-ray of the half-space defined by \( qt_1 \) (\( qt_2 \)). The sweeping region is shown shaded. This sweeping region consists of two parts (Fig. 3.14(b)): i) triangle portion \( \triangle qb_1b_2 \) and ii) the segment whose chord is \( b_1b_2 \) (thick dashed line). Let \( A_{tri} \) be the area of \( \triangle qb_1b_2 \), \( A_{seg} \) be the area of the segment, and \( A_{SR} \) be the area of sweeping region, \( A_{SR} = A_{tri} + A_{seg} \).

\(^4\)We obtain non-linear curve fitting using Origin downloaded from www.originlab.com
We first compute $A_{tri}$ (Fig. 3.14(b)). Let $\delta = \angle b_2 cb_1$. Let $b$ be the length of the base $b_1 b_2$. Consider $\triangle b_2 cb_1$, we can obtain $b = 2r \sin(\frac{\delta}{2}) = 2 \sin(\frac{\delta}{2})$. Let $h$ be the height of $\triangle q b_1 b_2$, and $d$ be the distance from $c$ to $q$. We have $h = d + r \cos(\frac{\delta}{2}) = d + \cos(\frac{\delta}{2})$. Hence we can get $A_{tri} = \frac{1}{2} bh = \sin(\frac{\delta}{2})(d + \cos(\frac{\delta}{2}))$. We will show how to compute $\delta$ later.

We then compute $A_{seg}$ (Fig. 3.14(b)). As this can be easily obtained from $A_{seg} = \frac{1}{2} r^2 \delta - \frac{1}{2} r^2 \sin \delta = \frac{1}{2} (\delta - \sin \delta)$, the area of sweeping region $A_{SR} = A_{tri} + A_{seg}$ can be calculated by Eq. (3.4).

$$A_{SR} = \sin(\frac{\delta}{2})(d + \cos(\frac{\delta}{2})) + \frac{1}{2}(\delta - \sin \delta)$$

(3.4)

Figure 3.14: Expected area of the sweeping region

Now, we compute $\delta$ (Fig. 3.14(b)). Let $e$ be a point on $q b_2$ such that the line $ce$ is perpendicular to $q b_2$. Let $\mu = \angle qce$, $\gamma = \angle ec b_2$ and $\eta = \mu + \gamma$. Since $\triangle q c b_2$ and $\triangle q c b_1$ are symmetric, we have $2\eta + \delta = 2\pi$. Therefore, we have $\delta = 2\pi - 2\eta = 2\pi - 2\mu - 2\gamma$, and we need to compute $\mu$ and $\gamma$.

Now, we compute $\mu$ (Fig. 3.14(b)). Let $\alpha = \angle t_1 c t_2$ and $\beta = \angle b_1 q b_2$. Considering the right angle triangle $\triangle q e c$, we have $\mu = \frac{\pi}{2} - \frac{\beta}{2}$. Note that $ct_1$ is parallel to $qb_1$.
because they are both perpendicular to $\vec{q}_1$. Similarly, $\vec{c}_2$ is parallel to $\vec{q}_b$. Hence, $\beta = \alpha$. Considering the right angle triangle $\triangle ct_1q$ or $\triangle ct_2q$, we have $\cos(\alpha) = \frac{r'}{d}$. Hence, we have $\alpha = \beta = 2 \arccos(\frac{r'}{d})$, and $\mu = \frac{\pi}{2} - \arccos(\frac{r'}{d})$.

Now, we compute $\gamma$ (Fig. 3.14(b)). Let $\text{dist}(c, e)$ be the distance from $c$ to $e$. Considering the right angle triangle $\triangle qec$, we have $\text{dist}(c, e) = d \cdot \sin(\frac{\beta}{2})$. Considering the right angle triangle $\triangle ceb_2$, we have $\cos(\gamma) = \frac{\text{dist}(c, e)}{r} = \text{dist}(c, e) = d \cdot \sin(\frac{\beta}{2})$. Therefore, we have $\gamma = \arccos(d \cdot \sin(\frac{\beta}{2}))$. Since $\beta = 2 \arccos(\frac{r'}{d})$, we have $\gamma = \arccos(d \sqrt{1 - (\frac{r'}{d})^2})$.

Now, we can compute $\delta = 2\pi - 2\mu - 2\gamma$ using Eq. (3.5).

$$\delta = \pi - 2 \arccos(d \sqrt{1 - (\frac{r'}{d})^2}) + 2 \arccos(\frac{r'}{d})$$  \hspace{1cm} (3.5)

### 3.5 Experiments

#### 3.5.1 Experimental settings

We compare our algorithm (labelled as Our) with the state-of-the-art algorithm CHFC [YLK09] (extended for $k > 1$ as described in Section 3.1). Both algorithms are implemented in C++, compiled by g++ with flag -O3. The experiments are run on a 32-bit PC with Intel Xeon 2.40GHz dual CPU and 4GB memory running Debian Linux. As stated earlier, our algorithm adopts existing algorithms [CSLZ14] to compute the $k$-depth contour. Two algorithms were proposed in [CSLZ14], namely SkyRider and KnightRider. KnightRider is an I/O optimal algorithm, while SkyRider consumes slightly more I/Os but is more efficient in terms of CPU cost. Although any of the two algorithms can be embedded in our framework, in our implementation, we prefer better CPU cost and use SkyRider.

The experimental settings are similar to those used in [YLK09]. Specifically, we
use both synthetic and real data sets. The real data set consists of 476,587 points, and is a combination of 4 data sets\(^5\), namely nodes in California road network (CA), San Francisco road network (SF), road network of North American (NA) and points of interest in CA. Longitude and latitude are normalized to the range \([0,100000]\). We also removed duplicated points which reduced the data size to 474,655. The synthetic data set consists of 3,000,000 points following uniform distribution.

We vary \(k\) from 1 to 50 with default value 10. We vary \(|F|\) and \(|U|\) from 100,000 to 1,000,000 on synthetic data set, and from 50,000 to 200,000 on real data set (the facilities and users are randomly selected points from the respective data sets). Similar to existing works, we set \(|F| = |U|\) in each experiment unless mentioned otherwise.

Each of the set of facilities and users is indexed by a R*-tree. Similar to existing works, the construction cost of these indices is not included in our evaluation. Everything else is calculated on-the-fly (e.g., \(k\)-depth contour, the main-memory R*-tree containing shallow facilities) and the relevant cost is included. The page size is set to 4096 bytes. Since the I/O cost is highly system specific [YCLW15], instead of reporting total time (i.e., sum of CPU time and I/O time), we report the CPU cost and the number of I/Os separately (which is a better evaluation approach as suggested in [YCLW15]). For each experiments, 1000 points are randomly selected from the facilities data set and correspond to the queries (unless mentioned otherwise), and we report the average cost per query.

### 3.5.2 Shallow facilities

Our algorithm requires only the set of shallow facilities to answer R\(k\)FN queries, while CHFC requires access to the facilities that lie on or outside \(k\)-th convex

\(^5\)http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm
hull (Section 3.1). We compare the number of shallow facilities with the number of facilities on or outside $k$-th convex hull. This is important because the CPU cost and I/O cost of each algorithm is significantly affected by the number of such facilities.

![Figure 3.15: Number of shallow facilities](image)

Figure 3.15 compares the number of shallow facilities with the number of facilities on or outside of $k$-th convex hull. Figure 3.15(a) shows the effect of $k$. For $k = 1$, our algorithm must access the same number of facilities as CHFC since 1-depth contour is exactly the same as convex hull. However, for $k > 1$, the number of shallow facilities is much smaller than the number facilities on or outside of $k$-th convex hull, e.g., for $k = 50$, the number of facilities CHFC must access is approximately 5 times more than that of our algorithm. Figure 3.15(b) shows the effect of $|F|$. In this case, the number of facilities CHFC must access is approximately 3 times more than our algorithm for various $|F|$. Also, note that our algorithm scales better for increasing $k$ and $|F|$. 
3.5.3 Performance evaluation

3.5.3.1 Effect of $k$

Figure 3.16 and 3.17 study the effect of $k$ on real and synthetic data sets, respectively. Specifically, Figure 3.16(a) and 3.17(a) show the I/O cost and Figure 3.16(b) and 3.17(b) show the CPU time. Although the results on both synthetic and real data sets demonstrate similar trend, we use linear scale in Figure 3.16 to clearly show how each algorithm scales with $k$, and log-scale in Figure 3.17 to better compare the performance for each value of $k$. Clearly on both data sets, our algorithm is several orders of magnitude better than CHFC in terms of both I/O and CPU cost and scales better.

Although for $k = 1$ $k$-th convex hull and $k$-depth contour are the same, our algorithm is significantly better than CHFC on both data sets. This is mainly because our algorithm can terminate in case of futile query even before the computation of $k$-depth contour has been completed in contrast to CHFC that can terminate only after the convex hull has been computed.
Chapter 3. Efficiently Computing Reverse $k$ Furthest Neighbors

Figure 3.17: Synthetic data set

Figure 3.18: Real data set
3.5.3.2 Effect of data size

Figure 3.18 and 3.19 study the effect of data size. Figure 3.18(a) and 3.19(a) show the I/O cost whereas Figure 3.18(b) and 3.19(b) show the CPU cost. For the similar reasons mentioned earlier, linear scale is used in Figure 3.18 and log scale is used in Figure 3.19. Our algorithm is several orders of magnitude better than CHFC in terms of both CPU and I/O cost. Furthermore, the cost of our algorithm is not affected by the data set size in contrast to the cost of CHFC that increases significantly. This is mainly because the verification cost of CHFC is linear to $|U|$, whereas our algorithm is not significantly affected by $|U|$ due to more sophisticated filtering and verification strategies adopted.

![Figure 3.19: Synthetic data set](image)

3.5.3.3 Non-futile queries

In the previous section, we reported the results where each query corresponds to a randomly selected facility from the data set. Since the number of shallow facilities is much smaller as compared to the total number of facilities, a majority of the queries are futile queries (a query that is not a shallow facility is a futile query). Is it possible that our algorithm is better than CHFC for futile queries but worse
for non-futile queries? In this section, we address this question and select only the queries that are not futile. Specifically, we randomly generate 1000 points outside $k$-depth contour and each point corresponds to one query (which is guaranteed not to be futile).

![Graph](image)

Figure 3.20: Non-futile queries (effect of $k$)

Figure 3.20 shows the effect of $k$. Note that the I/O cost of our algorithm is larger than CHFC for $k = 1$. This is because CHFC uses an I/O optimal algorithm to compute convex hull, whereas in our implementation, we use SkyRider which is not I/O optimal. Our algorithm can easily adopt an I/O optimal algorithm (e.g., KnightRider) to reduce the I/O cost. For $k > 1$, our algorithm is significantly better than CHFC for non-futile queries (e.g., for $k = 50$, the number of I/Os for CHFC is more than ten times of that of our algorithm). In terms of CPU cost, our algorithm is faster than CHFC for all values of $k$ including $k = 1$. Figure 3.20 also demonstrates that pruning cost is the dominant cost for both algorithms.

Figure 3.21 shows the effect of data set. Clearly, our algorithm performs significantly better in terms of both I/O and CPU cost, e.g., for the largest data set, CHFC consumes around 5 times more I/O and is around 7 times slower.
3.5.3.4 Effectiveness of proposed techniques

Since our algorithm requires pruning and verification only for the non-futile queries, we randomly generate 1000 queries that lie outside the convex hull, and report the average cost per these non-futile queries.

Effectiveness of pruning strategies. Figure 3.22 evaluates the efficiency of our pruning techniques by showing the number of users pruned by the candidate polygon (Section 3.3.2.4) and the number of users pruned by the sweeping regions (Section 3.3.2.4). To show the effectiveness of the pruning strategies, we also show the total number of users that are not RkFNs. Since only the users that are not
RkFN can be pruned by a pruning strategy, this corresponds to the maximum possible number of users that any pruning technique can prune.

Figure 3.22(a) evaluates the effectiveness of pruning techniques for varying values of $k$ with $|U|$ and $|F|$ set to 100,000. Figure 3.22(b) evaluates the effect of varying data set size with $k$ set to 10. Note that our pruning techniques can prune almost all users that are not RkFNs, more than 99.5% of which are pruned by our pruning techniques. Furthermore, both strategies are quite effective and prune a significant number of users.

![Figure 3.22(a) and Figure 3.22(b)](image)

**Effectiveness of shadows.** Recall that a candidate user $u$ can be confirmed as RkFN and does not require verification if it lies in a shadow. We evaluate the effectiveness of the shadows by showing the number of users that are not required to be verified. Figure 3.23 shows that around $25 - 35\%$ of the users lie in the shadows and are not required to be verified. This results in significant saving because the boolean $k$-region queries are not required for such users.
3.5.4 Theoretical analysis evaluation

This section evaluates the accuracy of our theoretical analysis. In each experiment, we generate 100,000 facilities and 100,000 users following uniform distribution in a circle of unit radius.

3.5.4.1 Area of $k$-depth contour

Recall that, in Section 3.4.1, we obtain the value of $\theta$ using Equation (3.1) that approximates $\theta - \sin(\theta) = \frac{2k\pi}{|F|}$ using non-linear curve fitting. We remark that the value of $\theta$ can also be approximated to desired accuracy from $\theta - \sin(\theta) = \frac{2k\pi}{|F|}$ by using a binary search, e.g., continue a binary search on $\theta$ until $\theta - \sin(\theta)$ is almost equal to $\frac{2k\pi}{|F|}$. In our experiments, we use both of these methods to approximate $\theta$, and in turn compute the expected area of $k$-depth contour. The result of the two methods are labeled *Estimation by Approximation* and *Estimation by Binary Search*, respectively. The so-called actual area of the $k$-depth contour (shown as *Experimental*) is estimated by generating 1 Million points uniformly at random in the data universe and finding how many of these points lie inside the $k$-depth contour. The area is shown relative to the total area of the data space (e.g., $x\%$ means the area of $k$-depth contour is $x\%$ of the total space).

Figure 3.24(a) shows the result for $k$ from 1 to 50. It can be observed that the estimation by binary search is almost identical to the experimental result. However, although estimation by approximation follows the trend, it is not as accurate as the estimation by binary search. This is because the nonlinear curve fitting is optimized for the value of $\theta$ for the range from 0 to $\pi$ which is useful when $k$ varies a lot. To confirm this, in Figure 3.24(b), we vary $k$ from 1 to 10,000 and evaluate the theoretical analysis. As it can be seen, the estimation is quite close to the experimental value.
We remark that the analysis of $k$-depth contour is not only useful for R$k$FN queries but also of stand-alone interest due to its applications in computational geometry and statistics [CSLZ14].

Figure 3.25: Area of sweeping region (varying $k$)

3.5.4.2 Area of sweeping regions

We randomly generate 1,000 non-futile queries and report the average area of sweeping regions for different values of $k$ (Figure 3.25). The experimental results are quite close to the theoretical results. Recall that the area outside the sweeping regions can be pruned. Figure 3.25 shows that the area of sweeping region is quite small (less than 12% of the total space for $k = 50$), hence, a large area can be
pruned using the sweeping regions strategy.
Chapter 4

Selecting Set of Representative Objects

This chapter studies the problem of selecting set of representative products considering both diversity and coverage based on reverse top-k queries. Given a set of products and a set of user preferences, we then study the problem of selecting a bounded set of t representative products that individually favoured by various types of users and collectively favoured by as much users as possible. This problem has numerous applications in electronic marketplaces, e.g., for selecting a number of representative products to be displayed on the home page of online business or review web sites. The majority of existing works ignore user preferences, and fail to capture user interests. A few works that did consider user preferences either concentrate on coverage or diversity, and there is a lack of balanced work. We model this problem as a combination of diversity problem, where each product has its focus of user preferences, and coverage problem, where the set of selected products collectively cover as much user preferences as possible. Since this problem is NP-hard, we employ a greedy algorithm that takes as input the sets of reverse top-k
user preferences of $k$ skyband products. For the sake of time and space efficiency, we adopt MinHash and KMV Synopses to assist set operations. We prove that the proposed greedy algorithm is $\epsilon$-approximate. Our experimental study demonstrates the performance of the algorithms in terms of time, space consumption and quality of the selected set of products.

This chapter is structured as follows. Section 4.1 provides the necessary preliminaries, including top-$k$ and reverse top-$k$ queries, MinHash and KMV synopsis. In Section 4.2 we formally define the $t$-representative problem. We present a greedy algorithm in Section 4.3 and prove it is $\epsilon$-approximate in Section 4.4. Finally, we present experimental results in Section 4.5.

4.1 Preliminaries

4.1.1 Top-$k$ and reverse top-$k$ queries

Given a $d$-dimensional space $\mathcal{R}^d$, we have a set of data objects $\mathcal{P}$, where each object $p \in \mathcal{P}$ is described by a $d$-dimensional point $p = \{p[0], \ldots, p[d-1]\}$ in $\mathcal{R}^d$. Here, $p[i] \geq 0$ for $0 \leq i \leq d$ describes an attribute of the data object $p$.

In addition, we have a set of weighting vectors $\mathcal{W}$, where each weighting vector $w \in \mathcal{W}$ is described by a $d$-dimensional vector $w = \{w[0], \ldots, w[d-1]\}$ in $\mathcal{R}^d$. Here, $w[i] \geq 0$ for $0 \leq i \leq d$ describes the relative weight of the $i$-th attribute of data objects in $\mathcal{P}$. As a widely accepted convention, we assume $\sum_{i=0}^{d-1} w[i] = 1$. A linear scoring function is of the form $s(p, w) = \sum_{i=0}^{d-1} p[i] \cdot w[i]$.

**Definition 4.1.** Top-$k$ query. Given a set of data objects $\mathcal{P}$, and a query weighting vector $w$, a top-$k$ query retrieves a set of data objects $\mathcal{S}_k(w)$ such that $\mathcal{S}_k(w) \subseteq \mathcal{P}$, $|\mathcal{S}_k(w)| = k$, and $\forall p_1, p_2 : p_1 \in \mathcal{S}_k(w), p_2 \in \mathcal{P} - \mathcal{S}_k(w)$ it holds that $s(p_1, w) \leq s(p_2, w)$. 
Definition 4.2. **Reverse top-\(k\) query.** Given a set of data objects \(P\), a set of weighting vectors \(W\), and a query object \(p \in P\), a reverse top-\(k\) query retrieves a set of weighting vectors \(S_k(p) \subseteq W\) such that a weighting vector \(w \in S_k(p)\) if and only if \(\exists p' \in S_k(w)\) for which \(s(p, w) \leq s(p', w)\).

### 4.1.2 MinHash

MinHash is a technique for estimating Jaccard similarity, firstly introduced by Border [Bro97]. Given a set of elements \(V\) and a hash function \(F : V \rightarrow V\) that defines a random ordering over \(V\). Let \(F^{\text{min}}(V)\) be the minimum element in \(V\) with respect to \(F\). Given two randomly chosen sets \(V_1\) and \(V_2\), and let \(J(V_1, V_2)\) be the Jaccard similarity of \(V_1\) and \(V_2\). The probability that \(F^{\text{min}}(V_1) = F^{\text{min}}(V_2)\) is equal to \(J(V_1, V_2)\). Let \(r\) be a random variable that is 1 when \(F^{\text{min}}(V_1) = F^{\text{min}}(V_2)\), and 0 otherwise, then \(r\) is an unbiased estimator of \(J(V_1, V_2)\). However, \(r\) has too high a variance to be useful for estimating \(J(V_1, V_2)\). The idea of MinHash scheme is to reduce the variance by averaging several variables constructed in the same way. In other words, given \(N\) hash functions \(F_i\), where \(i \in [1, N]\), and let \(n\) be the number of \(F_i\)s that has \(F_i^{\text{min}}(V_1) = F_i^{\text{min}}(V_2)\), then \(J(V_1, V_2)\) can be estimated by \(\frac{n}{N}\).

### 4.1.3 KMV synopsis

The \(k\) minimal values (KMV) technique is first proposed by Bar-Yossef et al. [BJK+02] to estimate the number of distinct values in a data stream. Suppose \(h\) is a pair-wise independent hash function which randomly maps the values onto the range \([0, 1]\) and \(h(v_i) \neq h(v_j)\) for any two different values \(v_i\) and \(v_j\). A KMV synopsis of a set \(D\) of values, denoted by \(L_D\), keeps \(k\) smallest hash values of the elements in \(D\). Then the number of distinct values in \(D\), denoted by \(\hat{D}\), can be estimated by \(\hat{D} = \frac{k}{L^{(k)}}\), where \(L^{(k)}\) is \(k\)-th smallest hash value. Bayer et
[BHR+07] systematically investigate the problem of distinct value estimation under multi-set operations. They show that $\hat{D} = \frac{k-1}{\binom{k}{2}}$ is an unbiased estimator.

**Multi-set Union.** Consider two sets $A$ and $B$, with their KMV synopses $L_A$ and $L_B$ of size $k_A$ and $k_B$, respectively. Let $L_A \oplus L_B$ be the set comprising the $k$ smallest values in $L_A \cup L_B$, where $k = \min(k_A, k_B)$. Then the set $L = L_A \oplus L_B$ is the KMV synopses of $A \cup B$. The number of distinct values in $A \cup B$, denoted by $\hat{D}_U$, can be estimated as follows:

$$\hat{D}_U = \frac{k - 1}{\binom{k}{2}} \quad (4.1)$$

### 4.2 Problem Definition

Given a set of products represented by data objects $P$. Given a set of user preference represented by weighting vectors $W$. Let $p_1, p_2 \in P$ be two products. Given a positive integer $k$, let $S_k(p_1), S_k(p_2) \subseteq W$ be the sets of reverse top-$k$ user preference of $p_1$ and $p_2$, respectively. We define the distance $d(p_1, p_2)$ of $p_1$ and $p_2$ as:

$$d(p_1, p_2) = 1 - \frac{|S_k(p_1) \cap S_k(p_2)|}{|S_k(p_1) \cup S_k(p_2)|} \quad (4.2)$$

Equation (4.2) is defined based on Jaccard similarity, we know that $d(p_1, p_2) \in [0, 1]$. When $S_k(p_1)$ and $S_k(p_2)$ has no common elements (i.e., user preferences), $d(p_1, p_2) = 1$, and when they are identical, $d(p_1, p_2) = 0$.

Note that, we chose Jaccard similarity over other distance metrics is to distinguish products based on their focus on markets. As long as two products attract different sets of users, they are considered as totally different products. This is independent on the dissimilarity of products based on their own properties or attributes. For example, a laptop targeting business man may differ a laptop targeting gamer
only by the screen size. If their similarity is measured by their attributes, they
will be considered as similar products. In contrast, as their reverse top-$k$ are quite
different, they will be considered as dissimilar products by our criteria.

**Problem 4.1. $t$-diversity problem.** Given a set of data objects $\mathcal{P}$, a distance
function $d$ that measures the dissimilarity of two data objects, $t$-diversity problem
is to select a size-$t$ subset $\mathcal{S}_d \subseteq \mathcal{P}$ such that:

$$
\mathcal{S}_d = \arg \max_{\mathcal{S}_d \subseteq \mathcal{P}, |\mathcal{S}_d| = t} d_{\min}(\mathcal{S}_d) \quad (4.3)
$$

where $d_{\min}(\mathcal{S}_d)$ is the minimum pair wise distance between any two data objects in
$\mathcal{S}_d$, defined as:

$$
d_{\min}(\mathcal{S}_d) = \min_{p_i, p_j \in \mathcal{S}_d, i \neq j} d(p_i, p_j) \quad (4.4)
$$

An optimal solution of $t$-diversity problem selects $t$ data objects $\mathcal{S}_d$ while max-
imizes $d_{\min}(\mathcal{S}_d)$. Intuitively, $t$ maximum diverse products should serve $t$ groups of
users with their unique preferences. $t$-diverse problem can be seen as a dispersion
problem [Erk90]. Which is to select a number of points out of a set of given candidate
points, such that the minimum distance between pairs of the selected points
is maximized. It is proved that $p$-dispersion problem is NP-complete [Erk90] by
transformation to the clique problem.

**Problem 4.2. $t$-coverage problem.** Given a set of data objects $\mathcal{P}$, a set of
weighting vectors $\mathcal{W}$, a positive integer $k$, $t$-coverage problem is to select a size-$t$
subset $\mathcal{S}_c \subseteq \mathcal{P}$ such that:

$$
\mathcal{S}_c = \arg \max_{\mathcal{S}_c \subseteq \mathcal{P}, |\mathcal{S}_c| = t} c(\mathcal{S}_c) \quad (4.5)
$$

where $c(\mathcal{S}_c)$ is the proportion of $\mathcal{W}$ covered by $\mathcal{S}_c$ (e.g., belong to some $\mathcal{S}_k(p)$ for
$p \in \mathcal{S}_c$), formally defined as:

$$
c(\mathcal{S}_c) = \frac{\left| \bigcup_{p_i \in \mathcal{S}_c} \mathcal{S}_k(p_i) \right|}{|\mathcal{W}|} \quad (4.6)
$$
An optimal solution of $t$-coverage problem selects $t$ data objects $S_c$ while maximizes $c(S_c)$. Intuitively, $t$ maximum coverage products should collectively be favoured by as many users as possible. $t$-coverage problem can be seen as a set cover problem. As each data object $p$ is associated with a set of weighting vectors $S_k(p)$, which is a subset of $W$, selecting $S_c$ is equivalent to selecting a set of subsets of $W$ such that as many weighting vectors as possible should belong to at least one $S_k(p)$ for $p \in S_c$. Hence, $t$-coverage problem is NP-hard.

**Problem 4.3. $t$-representative problem.** Given a set of data objects $P$, a set of weighting vectors $W$, a positive integer $k$, a non-negative weight $\alpha$, $t$-representative problem is to select a size-$t$ subset $S_r \subseteq P$ such that:

$$
S_r = \arg \max_{S_r \subseteq P \mid |S_r| = t} (\alpha \cdot d_{\min}(S_r) + (1 - \alpha) \cdot c(S_r))
$$

Consider a special case of $t$-representative problem where $\alpha = 0$. This is the case when $t$-representative problem becomes $t$-coverage problem. Since $t$-coverage problem, a special case of $t$-representative problem, is NP-hard, $t$-representative problem is NP-hard.

### 4.3 Algorithms and Techniques

A fundamental element for representative products selection is reverse top-$k$ sets. Hence we propose a two phases approach: 1) compute reverse top-$k$ sets for all products, excluding products with empty reverse top-$k$ sets (e.g., $S_k(p)$ for $p \in P$) (Section 4.3.1). 2) select $t$ products (e.g., $S_r \subseteq P$) via a greedy algorithm (Section 4.3.2).
4.3.1 Compute reverse top-$k$

We use R*-tree to index $\mathcal{P}$ and $\mathcal{W}$. As widely acknowledged, that only $k$-skyband points can have reverse top-$k$, hence we first use BBS [PTFS05] algorithm to filter $k$-skyband points out from $\mathcal{P}$, and store these points in a main memory R*-tree. Next, we present 3 algorithms to compute reverse top-$k$ sets.

**Reverse top-$k$ query based algorithm**

A straightforward algorithm is that iteratively calculate reverse top-$k$ for each $k$-skyband points. This can be improved. Given a point $p$, let $\mathcal{D}(p)$ denote the set of points that dominate $p$. We know that if the intersection of $\mathcal{S}_k(p')$ for $p' \in \mathcal{D}(p)$ is empty, then $\mathcal{S}_k(p)$ is empty (see Theorem 1 in [VDNK10]). Therefore, we calculate reverse top-$k$ $k$-skyband points $p$ in the order that they are determined by BBS, and we proceed to reverse top-$k$ calculation via BBRA [VDNK13] algorithm only if $\mathcal{D}(p)$ is empty or the intersection of $\mathcal{S}_k(p')$ for $p' \in \mathcal{D}(p)$ is not empty.

**Top-$k$ query based algorithm**

Since BBRA unavoidably have to calculate top-$k$ for many preferences to get $\mathcal{S}_k(p)$ for a single point. When BBRA is invoked for multiple points, there are redundant computations. Hence, a better way is to calculate top-$k$ $\mathcal{S}_k(w)$ for all the preferences $w \in \mathcal{W}$, and add $w$ to $\mathcal{S}_k(p)$ if $p \in \mathcal{S}_k(w)$. Here, we process top-$k$ query via a branch-and-bound search on R*-tree.

**Near brute force algorithm**

Since we have already obtained $k$-skyband, that means many of them will anyway need to be verified for top-$k$ computation. In this case, the overhead of branch-and-bound R*-tree traversing becomes a significant portion. To remove this overhead, we propose a near brute force algorithm. That is, for each $w$, we iterate through all $k$-skyband points $p$, and calculate the score $s(p, w)$. We maintain a size $k$ heap, and keep only the best $k$ points. At last, we add $w$ to $\mathcal{S}_k(p)$ if $p$ remains
in the heap. This algorithm is near brute force for that only two widely know
tricks are employed, that is: 1) calculation of $k$-skyband. 2) use heap instead of
$O(n \log n)$ sorting.

### 4.3.2 Select representative products

Since $t$-representative problem is NP-hard, we propose a greedy algorithm that
iteratively select currently optimal point that produce the highest value of $\alpha \cdot d_{\min}(S_r) + (1 - \alpha) \cdot c(S_r)$ among all the candidates. Notice that both $d_{\min}(S_r)$ and
$c(S_r)$ requires set operations heavily. When $|S_k(p)|$ for candidate point $p$ is large,
these set operations would become the bottleneck of $S_r$ computation. To speed
up these set operations, we employ MinHash for $d_{\min}(S_r)$ calculation, and employ
KMV Synopses for $c(S_r)$ calculation.

### 4.4 Theoretical Analysis

In this section, we analyse the approximation ratio of the proposed greedy algorithm
teoretically.

**Theorem 4.1.** The proposed greedy algorithm is $\epsilon$-approximate for some $\epsilon \in [\min(\frac{1}{t}, \frac{1}{|S_k(p)|}), 1]$.

**Proof.** When $t = 1$, $t$-representative problem becomes $t$-coverage problem. Since
the proposed greedy algorithm select the product $p$ that has $|S_k(p)| \geq |S_k(p')|$ for all $p' \in \mathcal{P}$, the solution is optimal (i.e., $\epsilon = 1$).

For $t > 1$, let $S_r = \{p_1, \cdots, p_t\}$ be the set of $t$ products selected by the proposed
greedy algorithm, and let $S^*_r = \{p^*_1, \cdots, p^*_t\}$ be the optimal solution to the $t$-
representative problem. The proximity ratio $\epsilon$ can be obtained by:

$$
\epsilon = \frac{\alpha \cdot d_{\min}(S_r) + (1 - \alpha) \cdot c(S_r)}{\alpha \cdot d_{\min}(S'_r) + (1 - \alpha) \cdot c(S'_r)}
$$

(4.8)

$$
= \frac{\alpha \cdot d_{\min}(S_r) + \frac{(1-\alpha)}{|W|} \cdot |\bigcup_{i \in S_r} S_k(p_i)|}{\alpha \cdot d_{\min}(S'_r) + \frac{(1-\alpha)}{|W|} \cdot |\bigcup_{i \in S'_r} S_k(p'_i)|}
$$

(4.9)

If we can find a pair $p_i$ and $p_j$ in $S_r$ such that $S_k(p_i)$ and $S_k(p_j)$ are identical, we know that $c(S_r) = c(S'_r) = 1$, and $d_{\min}(S_r) = d_{\min}(S'_r) = 0$. Hence we have $\epsilon = 1$.

If $S_k(p_i)$ differ each other by at least one user preference for all $p_i \in S_r$, $S_r$ may or may not be the optimal solution. Without lose of generality, let $|S_k(p_i)| \geq |S_k(p_j)|$ for $i \in [1, t]$, and $|S_k(p_i^2)| \geq |S_k(p_j^2)|$ for $i \in [1, t]$. We can get:

$$
\epsilon \geq \frac{\alpha \cdot (1 - \frac{|S_k(p_i)|}{|S_k(p_i)|}) + \frac{(1-\alpha)}{|W|} \cdot |S_k(p_i)|}{\alpha \cdot 1 + \frac{(1-\alpha)}{|W|} \cdot (t \cdot |S_k(p_i)|)}
$$

(4.10)

$$
\geq \frac{\alpha \cdot (1 - \frac{|S_k(p_i)|}{|S_k(p_i)|}) + \frac{(1-\alpha)}{|W|} \cdot |S_k(p_i)|}{\alpha \cdot 1 + \frac{(1-\alpha)}{|W|} \cdot (t \cdot |S_k(p_i)|)}
$$

(4.11)

$$
= \frac{|S_k(p_i)|^2 + \alpha \cdot (|W| - |S_k(p_i)|)^2}{t \cdot |S_k(p_i)|^2 + \alpha \cdot (|W| - |S_k(p_i)|) \cdot (\alpha)^2}
$$

(4.12)

The derivation\(^1\) is:

$$
\frac{|W| \cdot (t - |S_k(p_i)|)}{(t \cdot |S_k(p_i)| + (|W| - t \cdot |S_k(p_i)|) \cdot \alpha)^2}
$$

(4.13)

Now we have 3 cases:

1. When $t = |S_k(p_i)|$, $\epsilon$ is lower bounded by $\frac{1}{t}$.

2. When $t > |S_k(p_i)|$, this is a monotonically increasing function, and $\epsilon$ is lower bounded by a value $\in \left[\frac{1}{t}, \frac{1}{|S_k(p_i)|}\right]$, depending on the value of $\alpha$.

3. When $t < |S_k(p_i)|$, this is a monotonically decreasing function, and $\epsilon$ is lower bounded by a value $\in \left[\frac{1}{|S_k(p_i)|}, \frac{1}{t}\right]$, depending on the value of $\alpha$.

\(^{1}\)Obtained from quotient rule, $\frac{d}{d\alpha} \left( \frac{u}{v} \right) = \frac{v \cdot \frac{du}{d\alpha} - u \cdot \frac{dv}{d\alpha}}{v^2}$. 
Note that MinHash and KMV Synopses are not used when we derive Theorem 4.1.

### 4.5 Experiments

In this section, we present our experimental evaluation. All algorithms are implemented in C++, compiled by g++ with flag -O3. The experiments are run on a 32-bit PC with Intel Xeon 2.40GHz dual CPU and 4GB memory running Debian Linux. As moderate data sets may be completely loaded in main memory, we use main memory R*-tree with 4kB pagesize, and we evaluate CPU time mainly. To cooperate cases where disk-based index is used, we also report the number of page requests.

**Algorithms** We compare Reverse top-$k$ query based algorithm (RTK), Top-$k$ query based algorithm (TK) and Near brute force algorithm (NBF) for phase one. We also compare greedy selection using exact set operation (ESO) and MinHash and KMV Synopses based set operation (MK) for phase two. Note that the overhead of creating MinHash and KMV Synopses for use by the process of MK is also evaluated, and denoted as OH.

**Data sets** We use both synthetic and real data similar to those used in [GVDN15]. Specifically, we use two synthetic data sets, namely uniform and anti-correlated. For the uniform data set, the data object values for all dimensions are generated independently using a uniform distribution. The anti-correlated data set is generated in the same way as in [GVDN15]. We also use the same real data set \textsc{HOUSE} that consists of more than 147,043 6-dimensional tuples, representing the percentage of an American family’s annual income spent on 6 types of expenditure: gas,
electricity, water, heating, insurance, and property tax. For the data set \( \mathcal{W} \), we use a clustered distribution to simulate different types of user preferences.

**Parameters** Main parameters and values used are specified in Table 4.1, default settings are in bold. We have 100 randomly generated hash functions for MinHash. We use Golden ratio multiplication as the hash function for KMV Synopses and we keep 17 smallest values.

### 4.5.1 Effect of \( |\mathcal{W}| \)

Figure 4.1 shows the effect of \( |\mathcal{W}| \). This experiment is conducted on small uniform data set. In phase 1 (Figure 4.1(a)), NBF is several times faster than RTK, the fastest among the 3 algorithms. It also scales the best compared with the other
Table 4.2: Number of page requests

<table>
<thead>
<tr>
<th></th>
<th>WT</th>
<th>RTK</th>
<th>TK</th>
<th>NBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>327123</td>
<td>642</td>
<td>642</td>
<td></td>
</tr>
<tr>
<td>20,000</td>
<td>586712</td>
<td>954</td>
<td>954</td>
<td></td>
</tr>
<tr>
<td>30,000</td>
<td>908755</td>
<td>1272</td>
<td>1272</td>
<td></td>
</tr>
<tr>
<td>40,000</td>
<td>918044</td>
<td>1603</td>
<td>1603</td>
<td></td>
</tr>
<tr>
<td>50,000</td>
<td>1492885</td>
<td>1930</td>
<td>1930</td>
<td></td>
</tr>
</tbody>
</table>

two. Table 4.2 shows the number of page requests in phase 1. We can see that the IO cost demonstrates similar trends as in CPU time. We omit report of IO cost in later experiments as they are always similar to the CPU time of phase 1. In phase 2 (Figure 4.1(b)), despite the overhead of computing MinHash and KMV Synopses (OH), MK runs much faster than ESO. Although OH demonstrate linear growth with \( |W| \), it scales much better than ESO. MK process demonstrates almost no growth. The score of the results selected by MK is very close to ESO, and ESO scores nearly a full mark. As our default setting for \( \alpha \) is 0.5, the diversity part (e.g., \( \alpha \cdot d_{min}(S_r) \)) is around half of the weighted score. In this experiment, the use of MinHash and KMV Synopses does not affect the quality of selected results much.

Figure 4.2: Effect of \( |W| \) (in 1000) (large anti-correlated)

Figure 4.2 shows the effect of \( W \). This experiment is conducted on big anti-correlated data set. We can see that this group of figures demonstrate similar trend as the ones in Figure 4.1. The reason that NBF and MK scales even better in this experiment is not because of the cardinality, but the data distribution. We
remark that $\sum_{p \in \mathcal{P}} |S_k(p)| = k \cdot |\mathcal{W}|$ is the same for any data distribution. For anti-correlated data sets, the number of points in $k$-skyband is much larger compared with uniform data sets. Although $|S_k(p)|$ is smaller on average, the dominating factor is still the number of $k$-skyband points. Therefore, both in phase 1 and 2, the scalability of RTK and ESO are even worse compared with the experiments on uniform data set.

Due to space limitation, we omit experiments on effect of $|\mathcal{P}|$ as increasing $|\mathcal{P}|$ with fixed $|\mathcal{W}|$ is equivalent as decreasing $|\mathcal{W}|$ with fixed $|\mathcal{P}|$.

4.5.1.1 Effect of $d$

![Figure 4.3: Effect of d (small anti-correlated)](image)

Figure 4.3 shows the effect of $d$. We can see that the cost of RTK grows significantly compared with the other two. ESO slightly grows as the number of non-empty $S_k(p)$ grows with the number of $k$-skyband points when $d$ increases. Due to the inaccuracy of KMV Synopses, the coverage score of MK results are sometimes only half of the ones selected by ESO. In contrast, the diversity scores of MK looks identical to the ones of ESO. This is because, as $d$ grows, diversity is easier to achieve, especially on anti-correlated data sets, while coverage is harder to achieve.
4.5.1.2 Effect of $k$

Figure 4.4 is conducted on the real data set, HOUSE, and shows the effect of $k$. Both NBF and TK are not much influenced by $k$. However, RTK grows fast with $k$. In phase 2, MK including overhead scales much better than ESO. In Figure 4.4(c), there is a slight growth of coverage score along with $k$ for both ESO and MK. This is because with larger $k$, the size of $S_k(p)$ is larger on average, and hence easier to achieve coverage.

4.5.1.3 Effect of $t$

Figure 4.5 (conducted on small uniform data set) and Figure 4.6 (conducted on small anti-correlated data set) show the effect of $t$. ESO is faster on uniform data set is because the smaller number of $k$-skyband points, despite the larger size of $S_k(p)$ on average. We can also notice that it is easier to achieve coverage but harder...
to achieve diversity on uniform data sets, and that it is easier to achieve diversity but harder to achieve coverage on anti-correlated data sets. Hence, with increasing $t$, diversity score drops while coverage score remains high on uniform data set, and coverage score increases while diversity score remains high on anti-correlated data set.

### 4.5.1.4 Effect of $\alpha$

Figure 4.7 shows the effect of $\alpha$ on uniform data set, anti-correlated data set, and real data set. The figures clearly demonstrate the effectiveness of $\alpha$ on controlling the weight of diversity score and coverage score. It is clear that coverage scores were sacrificed for diversity scores with large $\alpha$. 
4.5.1.5 Significance of phase 2 time

Although we introduced MinHash and KMV Synopses to speed up set operations and save space, we also encourage exact set operation in some cases. For example, space limitation may not be an issue with nowadays hardware when the data set is of moderate size. Also, when the proportion of CPU time on set operation is not dominant, exact set operation may be considered as it has theoretical guarantees (Section 4.4).

Figure 4.8 shows the percentage of phase 2 time of all the 6 algorithms\(^2\) on uniform data set (Figure 4.8(a)), anti-correlated data set (Figure 4.8(b)), and real data set (Figure 4.8(c)). Note that, selection time is identical for RTK-ESO, TK-ESO and NBF-ESO, overhead is identical for RTK-MK, TK-MK and NBF-MK. Selection time for RTK-MK, TK-MK and NBF-MK are also identical. However, their proportions are too small to be visible. We distinguish the overhead time for computing MinHash and KMV Synopses is because this part is critical when considering reduce time and space cost by sacrificing accuracy.

Within the range of our experiments, it is clear that NBF is faster than both RTK and TK. The fancy pruning techniques of RTK and TK does not seem to justify their performance, and this is because they are designed for single query pro-

\(^2\)(3 for phase 1) × (2 for phase 2)
cessing, but here when applied in our problem, they actually introduce redundant computations compared with NBF.
Chapter 5

Maximize Spatial Influence of Facility Bundle

This chapter studies the problem of maximizing spatial influence of facility bundle based on reverse $k$ nearest neighbors (R$k$NN) queries. Consider a two dimensional Euclidean space, let $F$ be a set of points representing facilities and $U$ be a set of points representing users. Spatial influence of a facility is the number of users who have this facility as one of their $k$ nearest facilities. This is because that users normally prefer to go to their nearby facilities and are naturally influenced the most by their nearest facilities. Given a facility bundle of size $t$, spatial influence of the bundle is the number of distinct users influenced by any one of them. Existing works on facility selection problem find out top-$t$ facilities with the highest spatial influence. However, the literature lacks study on this problem when the $t$ facilities have the highest spatial influence as a bundle. We are the first to study the problem of Maximizing Bundled Reverse $k$ Nearest Neighbors (MB-R$k$NN), where the spatial influence of a facility bundle of size $t$ is maximized. We prove its NP-hardness, and propose a branch-and-bound best first search algorithm that
greedily select the currently best facility until we get $t$ facilities. We introduce the concept of $k$NN region such that a group of users have their $k$NN facilities all belong to the same $k$NN region. This sharing property of $k$NN region allows us to avoid redundant calculation with dynamic programming technique. We conduct experiments on real data sets and show that our algorithm is orders of magnitudes better than our baseline algorithm both in terms of CPU time and IO cost.

This chapter is structured as follows. Section 5.1 present a formal definition of MB-R$k$NN problem, prove its NP-hardness, and present a baseline algorithm. Section 5.2 present our algorithm. Section 5.3 present experimental results.

## 5.1 Preliminaries

In this section, we present a formal definition of our problem, prove it is NP-hard, and present our baseline algorithm.

### 5.1.1 Problem Definition

Consider a two dimensional Euclidean space, let $F$ be a set of points representing facilities and $U$ be a set of points representing users.

**Definition 5.1.** $(k$NN$)$ Given a query user $u \in U$ and a number $k$, a $k$NN query of $u$ retrieves every facility $f \in F$ such that $f$ is one of the $k$ nearest facilities from $u$. A facility like this is a $k$NN (facility) of $u$. The set of all such facilities, denoted as $kNN(u)$, is $k$NN set of $u$.

**Definition 5.2.** $(Rk$NN$)$ Given a query facility $f \in F$ and a number $k$, an $Rk$NN query of $f$ retrieves every user $u \in U$ such that $f \in kNN(u)$. A user like this is a $Rk$NN (user) of $f$. The set of all such users, denoted as $RkNN(f)$, is $Rk$NN set of $f$, and is also called the influence set of $f$.
Definition 5.3. **(Facility Bundle)** Let $S$ be a facility bundle, and $|S|$ be the size of $S$. Given a number $t$, a facility bundle of size $t$ is a set of $t$ facilities. The influence set of $S$, denoted as $R_k\text{NN}(S)$ is the union of the influence sets of all included facilities.

Definition 5.4. **(Spatial Influence)** Spatial influence of a facility $f$ or a facility bundle $S$ is the number of users in its influence set, denoted as $|R_k\text{NN}(f)|$ and $|R_k\text{NN}(S)|$, respectively.

Definition 5.5. **(Influence Zone [CLZZ11])** Influence zone of $f$, denoted as $Z(f)$, is a region such that for every user $u$, if $u \in Z(f)$ we have $u \in R_k\text{NN}(f)$, and if $u \notin Z(f)$ we have $u \notin R_k\text{NN}(f)$.

Influence zone is a star-shaped polygon such that for every point $p \in Z(f)$, the line segment $fp$ lies entirely in the polygon.

Definition 5.6. **(MB-R$_k$NN)** Given a number $t$, Maximizing Bundled Reverse $k$ Nearest Neighbors (MB-R$_k$NN) query retrieves a facility bundle $S$ such that $|S| = t$, and $|R_k\text{NN}(S)|$ is maximized.

### 5.1.2 NP-hardness

Next, we show that the MB-R$_k$NN problem is NP-hard by reducing an existing NP-complete problem called the Maximum Coverage Problem to MB-R$_k$NN in polynomial time.

**Maximum Coverage Problem:** Given a number $t$ and a collection of sets $S = \{S_1, S_2, S_3, \cdots, S_m\}$, the objective is to find a subset $S' \subset S$ such that $|S'| \leq t$ and the number of covered elements $|\bigcup_{S_i \in S'} S_i|$ is maximum.

We reduce the Maximum Coverage Problem to MB-R$_k$NN as follows. Sets $S_1, S_2, S_3, \cdots, S_m$ correspond to the influence sets of the facilities. The collection
Chapter 5. Maximize Spatial Influence of Facility Bundle

of sets $S$ is equal to the collection of the influence sets of all facilities. It is easy
to see that this transformation can be constructed in polynomial time, and it
can be easily verified that when the problem is solved in the transformed MB-
$R^kNN$ problem, the original Maximum Coverage Problem is also solved. Since the
Maximum Coverage Problem is an NP-complete problem, MB-$R^kNN$ is NP-hard.

5.1.3 Baseline Algorithm

Since MB-$R^kNN$ problem is NP-hard, it is straightforward to employ a greedy
algorithm. Iteratively, the currently best facility is selected until we have got a
facility bundle of desired size. To determine the currently best facility, we first
define the contribution of a facility.

**Definition 5.7. (Contribution)** Let $S$ be the current facility bundle. The con-
tribution of a facility $f$, denoted as $N(f)$, is $|R^kNN(S \cup f)| - |R^kNN(S)|$, or in
other words $|R^kNN(f) \setminus R^kNN(S)|$.

A facility $f$ is said to be the currently best facility if $f \notin S$ and for every other
facility $f' \in F$ we have $N(f) \geq N(f')$.

To determine the currently best facility in each iteration, it is necessary to cal-
culate influence set for every facility. As widely adopted, we assume that facilities
and users are each indexed by an R-tree, and we employ the state-of-the-art al-
gorithm $SLICE$ [YCLZ14] to do this. Then, we can calculate contribution of all
facilities, and the currently best facility can be identified by doing set minus as
described in Definition 5.7.

Notice that, since $S$ changes after each iteration, the contribution of the same
facility might also change after each iteration. Therefore, the contribution of all
facilities need to be calculated again after each iteration. We noticed that there
are duplicated calculations. Let $S_1 = \{f_1\}$ and $S_2 = \{f_1, f_2\}$ be the current facility bundle after the 1st and the 2nd iteration, respectively. To calculate $N(f)$ with $S_1$, we do $R_kNN(f) \setminus R_kNN(f_1)$. And then, to calculate $N(f)$ with $S_2$, we do $R_kNN(f) \setminus (R_kNN(f_1) \cup R_kNN(f_2))$, which is equivalent to $R_kNN(f) \setminus R_kNN(f_1) \setminus R_kNN(f_2)$. That is, we can exclude $R_kNN(f_1)$ from $R_kNN(f)$ after we have got $S_1$, and then exclude $R_kNN(f_2)$ from $R_kNN(f)$ after we have got $S_2$.

Inspired by this, we store $kNN$ sets for every user in addition to influence sets for every facility. Every time when a facility $f$ is to be inserted into $S$, we can for each user $u \in R_kNN(f) \setminus R_kNN(S)$, easily retrieve the facilities $f' \in kNN(u)$ and remove $u$ from $R_kNN(f')$. Note that this can actually be done by maintaining a counter for each influence set. As a result, the currently best facility is simply the facility with the highest counter for its influence set.

5.2 Techniques

5.2.1 Solution Overview

Similar as the baseline, we employ a greedy algorithm that iteratively selects the currently best facility and inserts it into $S$ until $|S| = t$. We give you an overview of our algorithm in this section.

Let $F^R$ be the R-tree indexing facilities, and $U^R$ be the R-tree indexing users. Let $F_e$ and $U_e$ be a node or an entry on $F^R$ and $U^R$, respectively. We first give you a brief introduction to the concept of maximum possible contribution and candidate $R_kNN$ set, to assist your understanding of the overview of our algorithm. More detailed information will be given in Section 5.2.3 and Section 5.2.4.

**Definition 5.8. (Maximum Possible Contribution)** The Maximum Possible Contribution (MPC) of $F_e$, denoted as $N_e(F_e)$, is an upper bound of the contribution...
of $F_e$. Thus, $N_e(F_e) \geq N(F_e)$.

Since MPC is no smaller than the contribution of $F_e$, higher value of MPC indicates higher chance of containing or being the next currently best facility. Therefore, we can access $F_e$s in decreasing order of their MPC. And this allows us to design a branch-and-bound best first search algorithm. At this stage, we initialize one’s MPC by the number of facilities belong to its candidate R$k$NN set, which is a super set of its R$k$NN set. However, $N_e(F_e)$ might take a tighter estimation, and we will elaborate on this further in Section 5.2.4.

**Definition 5.9. (Candidate R$k$NN Set)** The Candidate R$k$NN Set of $F_e$, denoted as $R_kNN_e(F_e)$, is a set of $U_e$ such that for every facility $f \in F_e$ we have $R_kNN(f) \subseteq R_kNN_e(F_e)$. The cardinality of the candidate R$k$NN set of $F_e$, denoted as $|R_kNN_e(F_e)|$ is the number of facilities indexed by $U_e \in R_kNN_e(F_e)$.

Recall the concept of influence zone, R$k$NN users of a facility occupies an area around it. Facilities that are geographically close to each other have their influence zone largely overlapped. This is the intuition behind this concept of candidate R$k$NN set. That is to group nearby facilities together, so that they can share the same candidate R$k$NN set without including too much false positives, and we can calculate one candidate R$k$NN set once for a few facilities.

Algorithm 6 illustrates the overview of our algorithm. $S$ is initialized as an empty set in line 1. Let $F^R.root$ and $U^R.root$ be the root node of $F^R$ and $U^R$, respectively. $R_kNN_e(F^R.root)$ is initialized to $U^R.root$. $N_e(F^R.root)$ is initialized to $|R_kNN_e(F^R.root)|$, which is equal to $|U|$ (line 2). Let $H$ be a max heap, it is initialized by $< N(F^R.root), F^R.root >$. Line 3 iterate through $F_e$ in descending order of $N_e(F_e)$. As long as $H$ is not empty and $|S| < t$, a heap entry $F_e$ is dequeued from $H$ in line 4. We check if $F_e$ is a node or a facility. Let $F^c_e$ be a child of $F_e$. If $F_e$
Algorithm 6: \textit{MB-RkNN}(\textit{FR}, \textit{UR}, \textit{t})

\textbf{Input}: \textit{FR}: an R-tree indexing facilities, \textit{UR}: an R-tree indexing users
\textit{t}: number of facilities to be selected, \textit{H}: a max heap

\textbf{Output}: \textit{S}: \textit{S} \subseteq \textit{F}, |\textit{S}| = \textit{t}

1. \textit{S} := \emptyset; \textit{RkNN}_e(\textit{FR}.\text{root}) := \textit{UR}.\text{root};
2. \textit{N}_e(\textit{FR}.\text{root}) := |\textit{U}|; \textit{H} \leftarrow < \textit{N}_e(\textit{FR}.\text{root}), \textit{FR}.\text{root}>;
3. \textbf{while} \textit{H} \neq \emptyset and \textit{|S| < t} \textbf{do}
4. \hspace{1em} dequeue an entry \textit{F}_e from \textit{H};
5. \hspace{1em} \textbf{if} \textit{F}_e is a node \textbf{then}
6. \hspace{2em} \textbf{for} child \textit{F}^c_e of \textit{F}_e \textbf{do}
7. \hspace{3em} calculate \textit{RkNN}_e(F^c_e);
8. \hspace{3em} \textit{N}_e(F^c_e) := |\textit{RkNN}_e(F^c_e)|; \textit{H} \leftarrow < \textit{N}_e(F^c_e), F^c_e>;
9. \hspace{1em} \textbf{else} \hspace{2em} /* \textit{F}_e is a facility */
10. \hspace{2em} \textbf{if} \textit{N}_e(\textit{F}_e) \neq \textit{N}(\textit{F}_e) \textbf{then} /* \textit{N}(\textit{F}_e) is not calculated */
11. \hspace{3em} \textit{N}(\textit{F}_e) := |\textit{RkNN}(\textit{F}_e) \setminus \textit{RkNN}(\textit{S})|;
12. \hspace{3em} \textit{N}_e(\textit{F}_e) := \textit{N}(\textit{F}_e); \textit{H} \leftarrow < \textit{N}(\textit{F}_e), \textit{F}_e>;
13. \hspace{1em} \textbf{else}
14. \hspace{2em} \textit{S} \leftarrow \textit{F}_e; clear \textit{N}(\textit{F}_e) for all \textit{F}_e;
15. \textbf{return} \textit{S};
is a node (line 5), then for every \( F^e_c \), we calculate \( RkNN_e(F^e_c) \), initialize \( N_e(F^e_c) \) and insert \(< N_e(F^e_c), F^e_c >\) into \( H \). If \( F^e_c \) is a facility, then we calculate \( RkNN(F^e_c) \), and initialize \( N_e(F^e_c) \) to \( |RkNN(F^e_c)| \). If \( F_e \) is a facility (line 9), we check if \( N_e(F_e) \) is set to \( N(F_e) \). Note that when \( N(F_e) \) is not calculated, \( N_e(F_e) \) is considered as not set to \( N(F_e) \). If it is not (line 10), we calculate \( N(F_e) \), update \( N_e(F_e) \), and insert \(< N_e(F_e), F_e >\) into \( H \). If it is (line 13), we add it to \( S \), and clear all calculated values of \( N(F_e) \) as they are invalidated by the facility newly added to \( S \).

### 5.2.2 \( k \)NN Region

Before we can explain how to calculate candidate \( RkNN \) set, we need the concept of \( k \)NN region. Recall that \( k \)NN facilities of a user occupies an area around it. Users that are geographically close to each other have their \( k \)NN sets largely overlapped. This is the intuition behind the concept of \( k \)NN region. That is to group nearby users together, so that they can share the same \( k \)NN region without including too much false positives, and we can calculate one \( k \)NN region once for a few users. And this is also the reason why we use \( U_e \) instead of individual users in candidate \( RkNN \) set.

**Definition 5.10. (\( k \)NN Region of a Point)** Let \( p \) be a point, the \( k \)NN region of \( p \), denoted as \( R(p) \), is the circle centered at \( p \) with radius equal to the distance from \( p \) to its \( k \)th nearest facility.

**Lemma 5.1.** Let \( p \) be a point, \( kNN(p) \in R(p) \).

**Proof.** We omit the proof of Lemma 5.1 as it is obvious from Definition 5.10. \( \square \)

**Definition 5.11. (\( k \)NN Region of a Line Segment)** (As illustrated in Fig. 5.1) Let \( v_1 \) and \( v_2 \) be two points that defines a line segment \( v_1v_2 \). The \( k \)NN
region of $v_1v_2$, denoted as $R(v_1v_2)$, is the minimum circle that contains $R(v_1)$ and $R(v_2)$.

**Lemma 5.2.** Let $v_1$ and $v_2$ be two points that defines a line segment $v_1v_2$. Let $p$ be a point on $v_1v_2$. $kNN(p) \in R(p) \subseteq R(v_1v_2)$.

**Proof.** As shown in Fig. 5.1. $kNN(v_1) \in R(v_1)$ and $kNN(v_2) \in R(v_2)$. Since $R(v_1v_2)$ is the smallest circle that contains both $R(v_1)$ and $R(v_2)$, let $o$ be the center of $R(v_1v_2)$, $o$ must lie on $v_1v_2$, and divides $v_1v_2$ into two segments, $v_1o$ and $ov_2$. Without lose of generality, we assume that $p$ lies on $v_1o$. This implies that we can find a circle $C_p$ that contains $R(v_1)$ and is contained by $R(v_1v_2)$. Since $R(v_1)$ has at least $k$ facilities, $C_p$ must have at least $k$ facilities. Thus, $kNN(p) \in R(p) \subseteq C_p \subseteq R(v_1v_2)$. 

**Definition 5.12.** (*kNN Region of a Rectangle*) (As illustrated in Fig. 5.2)

Let $r$ be a rectangle, $v_1, v_2, v_3, v_4$ be the four vertex of $r$. The $kNN$ region of $r$, denoted as $R(r)$, is the union of $R(v_1v_2), R(v_2v_3), R(v_3v_4), R(v_1v_4)$ and $r$.

**Lemma 5.3.** Given a rectangle $r$, and a point $p \in r$. $kNN(p) \in R(p) \subseteq R(r)$. 

---

Figure 5.1: $kNN$ Region of a Line Segment
Chapter 5. Maximize Spatial Influence of Facility Bundle

Figure 5.2: kNN region of rectangle

Figure 5.3: Lemma 5.3

Proof. As shown in Fig. 5.3. A point $p \in r$ either lies strictly inside $r$ (e.g., $p_1$) or on the boundary (e.g., $p_2$). Cases like $p_2$ is clear from Lemma 5.2, we prove cases like $p_1$ by contradiction.

Let $f$ be a facility such that $f \notin R(r)$ and $f \in kNN(p_1)$. Since $f \notin R(r)$, $f$ must be outside of $r$, and the line segment $p_1f$ intersects the boundary of $r$. Without lose of generality, let $p_2$ be the intersection point. Then we have $kNN(p_2) \subseteq R(p_2) \subseteq R(r)$.

Since $f \in kNN(p_1)$, we have $p_1 \in RkNN(f)$, and $p_1 \in Z(f)$. Since influence zone is a star-shaped polygon, and $p_2$ lies on $p_1f$, we have $p_2 \in Z(f)$, and $p_2 \in RkNN(f)$. Thus, $f \in kNN(p_2) \subseteq R(p_2) \subseteq R(r)$. This contradicts with the assumption $f \notin R(r)$.

5.2.3 Inherit Candidate R$k$NN Set

From the definition of candidate R$k$NN set, it is valid to set it as coarse as simply the root node of $U^R$, or as fine as exactly the R$k$NN set. The former would cover too much false positives. Recall that the cardinality of candidate R$k$NN set is used as the key to access $F_e$s in a best first manner, with too much false positives, the ordering will lose effectiveness. The later cannot effectively group users together.
To strike a balance here, we can initialize one’s candidate R$k$NN set as $U^R.root$, gradually open up $U_e \in RkNN_e(F_e)$ until all of them are of the right size, in other words, the right level on the tree. When we open up $U_e$, we need to know whether $U_e^c$, a child of $U_e$ should be pruned from $RkNN_e(F_e)$. This can be determined with the help of $R(U_e)$.

**Lemma 5.4. (Pruning)** $U_e$ can be pruned from $RkNN_e(F_e)$ when $R(U_e)$ and $F_e$ do not overlap with each other.

*Proof.* Let $r$ be the Minimum Bounding Rectangle (MBR) of $U_e$. From Lemma 5.3, we have $kNN(u) \in R(U_e)$ for every user $u \in U_e$. Since $F_e$ do not overlap with $R(U_e)$, we have $f \notin R(U_e)$ and $f \notin kNN(u)$ for every facility $f \in F_e$. In other words, we have $u \notin RkNN(f)$ for every pair of $u \in U_e$ and $f \in F_e$. Thus, $U_e$ can be pruned from $RkNN_e(F_e)$.

Next, we show that calculation of candidate R$k$NN set do not always have to start from $U^R.root$.

**Lemma 5.5. (Inheritance Property)** $RkNN_e(F_e^c) \subseteq RkNN_e(F_e)$.

*Proof.* Since $F_e^c$ is a child of $F_e$, $F_e$ contains $F_e^c$. Thus, if $F_e^c$ overlaps with $R(U_e)$, $F_e$ must also overlaps with $R(U_e)$. In other words, if $U_e \in RkNN_e(F_e^c)$, we must have $U_e \in RkNN_e(F_e)$ (e.g., $RkNN_e(F_e^c) \subseteq RkNN_e(F_e)$).

Lemma 5.5 implies that $RkNN_e(F_e^c)$ can be obtained by filtering on $U_e \in RkNN_e(F_e)$. This is referred to as the inheritance property of candidate R$k$NN set.

Algorithm 7 illustrates the process of inheritance. Line 1 loop through all $U_e \in RkNN_e(F_e)$. The ones who have their kNN region overlap with $F_e^c$ are selected in line 2. If $U_e$ is an user (line 3), $U_e$ is added to $RkNN_e(F_e^c)$. If $U_e$ is a
Algorithm 7: $\text{inherit}(F_e, F^c_e)$

**Input**: $F_e$: a node on facility R-tree, $F^c_e$: a child node/facility of $F_e$

**Output**: $RkNN_e(F^c_e)$

1. for $U_e$ in $RkNN_e(F_e)$ do
2.  if $F^c_e$ overlaps $R(U_e)$ then
3.    if $U_e$ is a user then
4.      $RkNN_e(F^c_e) \leftarrow U_e$;
5.  else /* $U_e$ is a node */
6.    for child $U^c_{e'}$ of $U_e$ do
7.      if $R(U^c_{e'})$ is not determined then
8.        calculate $R(U^c_{e'})$;
9.      if $F^c_e$ overlaps $R(U^c_{e'})$ then
10.     $RkNN_e(F^c_e) \leftarrow U^c_{e'}$;
node (line 5), all children $U^e_c$ of $U_e$ is examined. If $R(U^e_c)$ is not determined (line 7), then $R(U^e_c)$ is calculated. Note that we only calculate the same kNN region once, although it may be used several times. If $F^e_c$ overlaps $R(U^e_c)$ (line 9), it is added to $RkNN_e(F^e_c)$.

Note that in this algorithm, when we descend one level down on $F^R$, namely from $F_e$ to $F^e_c$, we also descend one level down on $U^R$, namely from $U_e$ to $U^e_c$. This is to say, whenever it is applicable, the distance from $U_e \in RkNN_e(F_e)$ to $U^R.root$ is no shorter than the distance from $F_e$ to $F^R.root$.

5.2.4 Maximum Possible Contribution and Contribution

Recall that in Algorithm 6, maximum possible contribution is initialized in line 8. However, it might be initialized to a tighter value.

**Lemma 5.6.** Let $f$ be a facility such that $f \in S$, and $F^e_c$ be a common ancestor of $F_e$ and $f$, then we have $N(F^e_c) \leq |RkNN_e(F^e_c)| - N(f)$.

**Proof.** We omit the proof of Lemma 5.6 as it is obvious.

Let $f^a (F^a_e)$ be an ancestor of $f$ ($F_e$). Inspired by Lemma 5.6, when we add a facility $f$ to $S$ (Algorithm 6 line 14), we decrease $N_e(f^a)$ by $N(f)$ for all $f^a$s. And when we initialize $N_e(F_e)$ (Algorithm 6 line 8), we assign it the minimum value among $N_e(F^a_c)$ and $|RkNN_e(F_e)|$.

Recall that in Algorithm 6 we have $N(F_e) := |RkNN(F_e) \setminus RkNN(S)|$ in line 11. However, there are cases when $|RkNN(F_e) \setminus RkNN(S)| = |RkNN(F_e)|$. This happens when the influence zone of $F_e$ is disjoint with every influence zone of facilities in $S$. When this happens, we set $N(F_e)$ to $|RkNN(F_e)|$ directly. The following Lemmas specifies the criteria when two influence zones are disjoint.
Lemma 5.7. Let $f_1$ and $f_2$ be two facilities. If there is a point $p_1$ that has both $f_1 \in kNN(p_1)$ and $f_2 \in kNN(p_1)$, then there is a point $p_2$ on the perpendicular bisector of $f_1$ and $f_2$ that has both $f_1 \in kNN(p_2)$ and $f_2 \in kNN(p_2)$.

Proof. As shown in Fig. 5.4. Without lose of generality, we assume that the line segment $f_1p_1$ intersects the perpendicular bisector of $f_1$ and $f_2$ at $p_2$. Let $C_1$ be a circle that centered at $p_1$ and passes through $f_1$. Since we have $f_1 \in kNN(p_1)$ and $f_2 \in kNN(p_1)$, there are at most $k$ facilities in $C_1$. Let $C_2$ be a circle that centered at $p_2$ and passes through $f_1$ and $f_2$. It is clear that $C_1$ contains $C_2$. Therefore, there are at most $k$ facilities in $C_2$. Thus, we have $f_1 \in kNN(p_2)$ and $f_2 \in kNN(p_2)$. \hfill \qed

Lemma 5.8. Let $f_1$ and $f_2$ be two facilities. If $Z(f_1)$ do not intersect with the perpendicular bisector between $f_1$ and $f_2$, then $Z(f_1)$ and $Z(f_2)$ are disjoint.

Proof. We prove this by contradiction. Assume $Z(f_1)$ and $Z(f_2)$ are not disjoint, and there is a point $p_1$ such that $f_1 \in kNN(p_1)$ and $f_2 \in kNN(p_1)$. From Lemma 5.7, we can find a point $p_2$ on the perpendicular bisector between $f_1$ and $f_2$ that has both $f_1 \in kNN(p_2)$ and $f_2 \in kNN(p_2)$. Thus $p_2 \in Z(f_1)$. This conflict with the assumption that $Z(f_1)$ do not intersect with the perpendicular bisector between $f_1$ and $f_2$. \hfill \qed
Lemma 5.9. Let $f_1$ and $f_2$ be two facilities, $d(f_1, v)$ be the distance from $f_1$ to the furthest point $v$ on $Z(f_1)$. If $f_2$ is more than twice the distance $d(f_1, v)$ away from $f_1$, then $Z(f_1)$ and $Z(f_2)$ are disjoint.

Proof. Since $f_2$ is more than twice the distance of $d(f_1, v)$ away from $f_1$, $Z(f_1)$ do not intersect with the perpendicular bisector between $f_1$ and $f_2$. From Lemma 5.8, $Z(f_1)$ and $Z(f_2)$ are disjoint. \qed

Inspired by Lemma 5.9, when we add a facility to $S$ in algorithm 6 line 14, we calculate its influence zone, and find out the distance from the facility to the furthest point on its influence zone. Then, in line 11, we check the distance from $F_e$ to all facilities in $S$ against Lemma 5.9. If Lemma 5.9 always holds, then we can set $N(F_e)$ to $|RkNN(F_e)|$ directly.

5.3 Experiments

In this section, we present our experimental evaluation. All algorithms are implemented in C++, compiled by g++ with flag -O3. The experiments are run on a 32-bit PC with Intel Xeon 2.40GHz dual CPU and 4GB memory running Debian Linux. As moderate data sets may be completely loaded in main memory, we use main memory R*-tree with 4kB pagesize, and we evaluate CPU time mainly. To cooperate cases where disk-based index is used, we also report the number of page requests.

Algorithms We compare our algorithm (DP) with our baseline algorithm (RKNN). Both algorithms calculate exact RkNN sets, and are approximate algorithms using the same criteria to select the currently best facility. Their results are expected to be the same with an assumption that no two facilities have equal spatial influence (e.g., same number of RkNN users). Hence, we compare their
Table 5.1: Experimental settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>F</td>
</tr>
<tr>
<td>$</td>
<td>U</td>
</tr>
<tr>
<td>$k$</td>
<td>5, 10, 15, 20, 30, 50</td>
</tr>
<tr>
<td>$t$</td>
<td>1, 3, 5, 8, 10</td>
</tr>
</tbody>
</table>

performance only on CPU time and I/O cost.

Data sets We use the real data set obtained from Yelp Dataset Challenge\(^1\). We extract 144072 points as facilities from business file\(^2\). A small random variable is added so that no two facilities share the same latitude and longitude. We extract 946600 points as users from tip file\(^3\). We treat each record in this file as a user. The location of a user is the latitude and longitude of the associated facility. A small random variable is added so that no two users share the same latitude and longitude. We than pick the required number of facilities and users out of the extracted points uniformly at random.

Parameters Parameters and values used are specified in Table 5.1, default settings are in bold.

![Figure 5.5: Effect of $|F|$ ($|U| = |F| \times 2$)](https://www.yelp.com/dataset/challenge)

\(^1\)https://www.yelp.com/dataset/challenge
\(^2\)yelp_academic_dataset_business.json
\(^3\)yelp_academic_dataset_tip.json
Effect of $|F|$. Fig. 5.5 show the effect of the size of the data sets. Specifically, Fig. 5.5(a) show the CPU time and Fig. 5.5(b) show the I/O cost. Since in real world scenario, it does not make sense to have $|U| < |F|$, when we vary the value of $|F|$, we also vary the value of $|U|$, and we keep the ratio $\frac{|U|}{|F|} = 2$ unchanged. Note that, log scale are used for $y$-axis in both Fig. 5.5(a) and Fig. 5.5(b). We can see that DP is orders of magnitudes better than RKNN both in terms of CPU time and I/O cost, and also scales better.

![Figure 5.6: Effect of $|U|$](image1)

Effect of $|U|$. Fig. 5.6 show the effect of the size of the data sets. Different from Fig. 5.5, we do not keep the ratio of $\frac{|U|}{|F|}$ unchanged. Instead, we keep the value of $|F|$ fixed, and vary only $|U|$. Without log scale for $y$-axis, it is clear that DP is orders of magnitudes better than RKNN both in terms of CPU time and I/O cost, and also scales much better.

Effect of $k$. Fig. 5.7 show the effect of $k$. It is clear that DP is orders of magnitudes better than RKNN both in terms of CPU time and I/O cost. Specifically, in Fig. 5.7(a), RKNN demonstrates linear growth in terms of CPU time while DP demonstrates no visible growth. In Fig. 5.7(b), they both demonstrate no visible growth in terms of I/O cost.

Effect of $t$. Fig. 5.8 show the effect of $t$. It is clear that DP is orders of magni-
Figure 5.7: Effect of $k$

Figure 5.8: Effect of $t$
tudes better than RKNN both in terms of CPU time and I/O cost. None of them demonstrate visible growth.
Chapter 6

Conclusion

In this Chapter, we conclude this thesis by summarizing the major contributions of our work. For each of the three problems we studied in this thesis, we are the first to study the problem, and we proposed efficient algorithms to address the problems.

Chapter 3 studies the problem of efficiently computing reverse $k$ furthest neighbors. To the best of our knowledge, we are the first to study the problem of R$k$FN query for arbitrary value of $k$. Based on several interesting observations and novel pruning techniques, we devise an efficient algorithm to process R$k$FN queries. We present a rigorous theoretical analysis to analyse two different aspects of the problem, namely the expected area of $k$-depth contour and the area pruned by our algorithm. This helps in analysing the expected number of futile queries, number of facilities required for computing a R$k$FN query, number of unpruned users and I/O cost etc. Also, to the best of our knowledge, we are the first to analyse the expected area of k-depth contour which is of stand-alone interest. We conduct extensive experimental study on both real and synthetic data sets, and demonstrate that our algorithm is up to several orders of magnitude faster than the existing
algorithms in terms of both CPU and I/O cost. We also conduct experiments to evaluate the accuracy of our theoretical analysis.

Chapter 4 studies the problem of selecting set of representative products considering both diversity and coverage based on reverse top-$k$ queries. We are the first to study the problem of selecting set of representative products while considering both coverage and diversity based on reverse top-$k$ queries. We formulate the problem to be an optimization problem of an objective function. As this problem is NP-hard, we propose a $\epsilon$-approximate greedy algorithm. To enhance the scalability and efficiency when the number of user preferences is huge, we propose to boost set operations by adopting MinHash and KMV Synopses.

Chapter 5 studies the problem of maximizing spatial influence of facility bundle based on reverse $k$ nearest neighbors (R$k$NN) queries. We introduce the concept of spatial influence of a facility and a facility bundle. We are the first to study the problem of Maximizing Bundled Reverse $k$ Nearest Neighbor (MB-R$k$NN). We prove that MB-R$k$NN problem is NP-hard and propose a branch-and-bound best first search algorithm that greedily select the currently best facility. We introduce the concept of $k$NN region that is shared by a group of users. This allows us to avoid redundant calculation with dynamic programming technique. We conduct experiments on real data sets. Experiments show that our algorithm is orders of magnitudes better than our baseline algorithm both in terms of CPU time and I/O cost.
Bibliography


[YL01] Congjun Yang and King-Ip Lin. An index structure for efficient reverse


