Efficient Influence Related Queries

by

Jianye Yang

B.E. Xidian University P.R.C, 2010

M.E. Xidian University P.R.C, 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

Monday 12th June, 2017

PLEASE TYPE

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Yang**

First name: **Jianye** Other name/s:

Abbreviation for degree as given in the University calendar: **PhD**

School: **School of Computer Science and Engineering** Faculty: **Faculty of Engineering**

Title: Efficient influence related queries.

Abstract 350 words maximum: (PLEASE TYPE)

Recently, there is a surge of interest on mining valuable information from the given datasets. As one of the most important information mining tasks, influence analysis has drawn tremendous attention in both industry and academic communities. Due to the large scale of dataset, there is an emerging call for efficient processing influence related queries. In this thesis, we study three important influence related problems regarding three types of data, i.e., product and user preference data, spatio-textual objects, and set-valued data.

Firstly, for product and user preference data, we formulate the problem of influence-based cost optimization on user preference functions, which is critical to unlock the great scientific and social-economic value of these data. By utilizing the classical k-level computation techniques, we show the solution space of our problem can be reduced to a finite number of possible positions (points). To efficient process this problem, we propose a traverse-based 2-dimensional algorithm with linear time complexity. For general multi-dimensional spaces, we develop a space partition based exact algorithm. To accelerate the computation, we further devise a randomized sampling method with high accuracy.

Secondly, for spatio-textual objects, we present a novel definition of object influence in applications where objects are of different categories. Under this definition, we investigate the problem of finding the top-k most influential objects. We first show that this problem is NP-hard with respect to the number of object categories. To tackle the computational hardness, we then develop efficient nearest neighbor set based exact as well as approximate algorithms. In particular, our polynomial approximate algorithm has a 2-factor performance guarantee.

Finally, for set-valued data, we investigate the problem of set containment join which is an essential and fundamental tool for set-valued data analysis. Based on the computing paradigms, we classify the existing algorithms into two categories, namely intersection-oriented and union-oriented, both of which have their limits. By utilizing the property of skewness in real-life data, we propose a new union-oriented method, namely TT-Join, which not only enhances the advantage of the previous union-oriented methods but also integrates the goodness of intersection-oriented methods by imposing a variant of prefix tree structure.

FOR OFFICE USE ONLY Date of completion of requirements for Award:

Copyright Statement

Signed .....

Date .....

Authenticity Statement

'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Signed .....

Date .....

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.'

Signed .....

Date .....

Abstract

Recently, there is a surge of interest on mining valuable information from the given datasets. As one of the most important information mining tasks, influence analysis has drawn tremendous attention in both industry and academic communities. Due to the large scale of dataset, there is an emerging call for efficient processing influence related queries. In this thesis, we study three important influence related problems regarding three types of data, i.e., product and user preference data, spatio-textual objects, and set-valued data.

Firstly, for product and user preference data, we formulate the problem of influence-based cost optimization on user preference functions, which is critical to unlock the great scientific and social-economic value of these data. By utilizing the classical k-level computation techniques, we show the solution space of our problem can be reduced to a finite number of possible positions (points). To efficient process this problem, we propose a traverse-based 2-dimensional algorithm with linear time complexity. For general multi-dimensional spaces, we develop a space partition based exact algorithm. To accelerate the computation, we further devise a randomized sampling method which can efficiently generate solutions with high accuracy.

Secondly, for spatio-textual objects, we present a novel definition of object influence in applications where objects are of different categories. Under this definition,

we investigate the problem of finding the top-k most influential objects. We first show that this problem is NP-hard with respect to the number of object categories. To tackle the computational hardness, we then develop efficient nearest neighbor set based exact as well as approximate algorithms. In particular, our polynomial approximate algorithm has a 2-factor performance guarantee.

Finally, for set-valued data, we investigate the problem of set containment join which is an essential and fundamental tool for set-valued data analysis. Based on the computing paradigms, we classify the existing algorithms into two categories, namely intersection-oriented and union-oriented, both of which have their limits. By utilizing the property of skewness in real-life data, we propose a new union-oriented method, namely TT-Join, which not only enhances the advantage of the previous union-oriented methods but also integrates the goodness of intersection-oriented methods by imposing a variant of prefix tree structure.

Publications

- **Jianye Yang**, Ying Zhang, Wenjie Zhang, Xuemin Lin. Influence based cost optimization on user preference. ICDE 2016. (Chapter 3)
- **Jianye Yang**, Wenjie Zhang, Ying Zhang, Xiaoyang Wang, Xuemin Lin. Categorical top-k spatial influence query. WWWJ 2016. (Chapter 4)
- **Jianye Yang**, Wenjie Zhang, Shiyu Yang, Ying Zhang, Xuemin Lin. TT-Join: Efficient set containment join. ICDE 2017. (Chapter 5)
- **Jianye Yang**, Wenjie Zhang, Shiyu Yang, Ying Zhang, Xuemin Lin. Efficient set containment join. Under review by VLDBJ. (Chapter 5)

Dedication

To my family

my parents

my relatives

my friends

For their love and support

Acknowledgements

First of all, I would like to deliver my sincere gratitude to my supervisors Prof. Xuemin Lin and Dr. Wenjie Zhang for their constant support and guidance. I thank Prof. Xuemin Lin for providing me the great opportunity to study in such a competitive research group and guiding me constantly through the road of my research. Also, I learnt the characteristics of an excellent researcher from Prof. Xuemin Lin - passion, diligence and earnestness in the research. I thank Dr. Wenjie Zhang for her constant encouragement and helpful guidance in my research work.

Secondly, I would like to express my great gratitude to my co-supervisor Dr. Ying Zhang for his constant encouragement and guidance. He has walked me through all the stages of the writing of this thesis. I also learnt many research skills and valuable research understandings from Dr. Ying Zhang. 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 Dr. Xiaoyang Wang and Dr. Shiyu Yang. I thank them for supporting the work presented in this thesis.

Besides, special thanks go to the former group members:Dr. Xiang Zhao, Dr. Wenren Yu, Dr. Liming Zhan, Dr. Chengyuan Zhang as well as fellow group members: Dr. Lijun Chang, Dr. Lu Qin, Dr. Zengfeng Huang, Dr. Xin Cao, Dr. Yang Wang, Mr. Long Yuan, Mr. Xing Feng, Ms. Shenlu Wang, Mr. Xiang Wang,

Mr. Longbin Lai, Mr. Bi Fei, Ms. Chen Zhang, Mr. Xubo Wang, Mr. Fan Zhang, Mr. Wei Li, Mr. Haida Zhang, and Mr. Yang Yang. The time we spent together will be memorized forever.

Last but not least, I am very thankful to my beloved family, parents and relatives for their love, support and encouragement during my PhD study.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this information age, data is enriched at a very rapid rate. Efficient mining valuable information from the given data system is of great interest, which not only opens a window to inspect the current state of the system, but also provides useful tools for future decision-making. As one of the most important information mining tasks, influence analysis has drawn tremendous attention in both industry and academic communities. For instance, in the e-business, analyzing the influence of a product regarding the user preferences and other competing products is fundamental in a wide range of applications such as marketing and advertising; in the physical world, analyzing the influence of a facility (e.g., gas station, supermarket) provides great value on applications such as decision support and resource allocation; in social media (e.g., Facebook, Twitter), analyzing the influence of a post is useful for tracking trending topics.

In this thesis, we study three important influence related problems on three types of datasets, namely product and user preference data, spatial object data, and set-valued data. Specifically, for product and user preference data, we formulate and investigate the problem of influence based cost optimization on user

preference, where each object is described by a point in a d-dimensional space and each preference function is a linear function represented by a weight vector with d components; for spatial object data, we study the problem of categorical top-k spatial influence query, where each object is modeled as a point in 2-dimensional Euclidean space with a specific type; for set-valued data, we investigate the problem of efficient set containment join, where each data record is described by a set of terms.

In Section 1.1, we first briefly introduce the background and motivations of investigating the above problems, and then explain the challenges encountered by these problems. Section 1.2 summarizes the contributions of this thesis for each studied problem. Thesis organization is presented in Section 1.3.

1.1 Motivations

1.1.1 Influence based Cost Optimization on User Preference

With the proliferation of e-business and preference learning techniques, there is a rapidly growing amount of product and preference data. Usually, an object (e.g., product) is described with multiple attributes (e.g., features). For instance, in a typical laptop commercial website, each laptop is described by a collection of features (e.g., CPU, RAM, Weight). On the other hand, the interest of a user towards objects with multiple attributes may be quantitatively expressed by a preference function. Given a user preference function $\mathbf{w}$ and an object p, we can compute the score of p with respect to the user preference function. By doing this, objects are ranked for a user according to her preference function. Generally, we say an object is *attractive/appealing* to the user if it is ranked the first by her. Given

a set of preference functions which are learned or collected from users [QGJ15], we can evaluate the influence/impact of an object by the number of users it can attract. Analyzing the influence of an object regarding the user preference functions and other competing objects is fundamental in a wide spectrum of applications such as marketing and advertising. In recent year, there is a surge of interest on the study of influence analysis on preference functions such as identifying the most influential objects [VDNK10, ADV12, LKC13, PW15] and supporting why-not questions [GLC+15].

In this thesis, we formulate the problem of influence-based cost optimization on user preference functions. Given a set of n user preferences, a set of objects in a d-dimensional space, and a cost function, we aim to find the cost optimal position for a new object in the space such that it attracts at least k users competing with the existing objects. This problem is of great practical value. For example, marketers may often conduct marketing research studies to decide the new product positioning by analyzing consumer preferences, the competitors' products and the manufacturing cost. In particular, a company wants to design a new type of laptop to attract at least a certain amount of users (e.g., 40% of the market share), and meanwhile, the manufacturing cost is lowest. Therefore, in this thesis, we aim to efficiently process the problem of influence-based cost optimization on user preference functions.

Challenges. The main challenges of influence based cost optimization on user preference lie in the following three aspects. Firstly, the number of user preferences n would be large. Given a target attracting number k, it is computational cost-prohibitive to explicitly enumerate $\binom{n}{k}$ possible groups of k out of n user preference functions. Secondly, the position of new object can be anywhere in the space. It is infeasible to enumerate the infinite possible positions of the new object. Thirdly, we

aim to support the general monotonic and convex cost function, which is challenging because the existing methods can only support quadratic cost function.

Our Approach. In this thesis, we aim to develop efficient query processing techniques to tackle the computational challenges. By taking advantage of the k-level computing techniques (e.g., [Mul91, ADBMS98]) and the monotonicity and convexity of the cost function, we reduce the solution space to a finite number of points on the k-level of n hyperplanes derived from n preference functions and objects. Although we can immediately come up with two baseline algorithms by applying the on-the-shelf k-level computing techniques: traverse based algorithm [AACS98, CSLZ14] for the 2-dimensional space and randomized incremental algorithm [Mul91] for the general multi-dimensional space, the computational cost is still expensive due to the high combinatorial complexity of k-level, $O(nk^{1/3})$ and $O(n^{\lfloor d/2 \rfloor}k^{\lceil d/2 \rceil})$ for 2 and d-dimensional spaces, respectively.

In this thesis, we develop an efficient traverse based algorithm for 2-dimensional spaces with time complexity $O(n)$ by exploiting the nice geometric properties of our problem in 2-dimensional spaces. It significantly improves the time complexity of the baseline approach, which is $O(n^2k^{1/3})$. To deal with the general multi-dimensional spaces, we develop a space partition based algorithm. Although we cannot improve the theoretical time complexity of the baseline solution, a variety of pruning techniques are developed to significantly enhance the performance of the algorithm by utilizing the cost bounds, influence bounds and local dominance relationship. Observe that it is cost expensive to maintain the k-level of hyperplanes in a partition. We further devise a novel sampling based approach such that the problem can be reduced to the classical *half-space intersection* problem in each round of samples. Empirical study shows that our sampling technique can significantly outperform the exact algorithm with high accuracy for space with $d \geq 3$.

1.1.2 Categorical Top-k Spatial Influence Query

With the advances in geo-positioning technologies, there is a rapidly growing number of spatial objects in many applications such as location based services (e.g., Google Map), where an object is described by its spatial location and a specific type. For instance, in the GPS navigation system, a POI (point of interest) is a geographically anchored pushpin that a user may find useful or interesting, which is annotated with function information (e.g., gas station, supermarket). Analyzing the influence of objects is an important spatial operator, which has been widely studied ever since it was introduced in [KM00] due to a wide spectrum of applications such as decision support, profile-based marketing, resource allocation, etc. Existing techniques define the influence of a facility as the number of users that consider it as nearest neighbors, namely, the bichromatic reverse nearest neighbor query. However, in practice several types of facilities exist and play different roles in satisfying uses' needs. Therefore, a more sophisticated way to evaluate the influence of facilities is preferable.

Instead of considering all facilities as the same type (e.g., finding the most influential supermarket among all supermarkets) as mentioned above, in this thesis, we first present a novel definition to evaluate the influence of a facility by its capacity of forming a *functional unit* together with facilities of other categories. Consider the CBD area in Sydney. There are various types of business/facilities such as restaurants, cafes, bookshops, supermarkets, etc. A customer may want to spend a leisure afternoon by drinking coffee, reading/purchasing books, and enjoying some sushi. Here, facilities of types *book store*, *cafe*, and *Sushi shop* form a functional unit for this customer where one functional unit consists exactly one facility from each category of desired services/facilities from the user.

After computing the optimal functional unit for each of the facilities, in this

thesis, we define the influence of a facility as the total number of optimal functional units that the facility participants in. Such an influence value clearly represents the importance of one facility in forming optimal functional units with other types of facilities and thus offers a new angle to analyze the potential of one facility in business strategies, urban planing, etc. In an extreme case, if a book store is the only one of this type and is surrounded by many other types of facilities, the influence value of this book store is high representing huge business potential.

Challenges. A straightforward approach is to compute the optimal functional unit for each object and then sum up the number of optimal functional units that each facility participates in. However, this approach is computational cost-prohibitive due to the following two aspects. Firstly, as we shall show that finding the optimal functional unit for a facility is an NP-hard problem regarding the number of types in the query. Secondly, the number of spatial facilities is usually massive involving a large number of categories.

Our Approach. To address the above challenges, it is critical to devise efficient spatial indexing and query processing techniques to support categorical top-k spatial influence query against a massive number of spatial objects. In this thesis, we follow the filtering-and-refinement framework based on R-tree style spatial indexes. Efficient and effective pruning techniques are developed to avoid the costly verification as much as possible. In particular, our approach consists of two main steps, namely possible participants finding and optimal feasible set computation. The major challenge in the first step is to derive a tight pruning distance. We find that the pruning distances derived by the common strategies are substantially larger than the optimal pruning distance. Motivated by this, we propose a neighbor set (NNS) based approach which leads to a performance acceleration by several orders of magnitude. In the second step, we develop two algorithms: one is an effi-

cient exact algorithm and the other is an approximate algorithm with performance guarantee.

1.1.3 Efficient Set Containment Join

Set-valued attributes play an important role in modeling database systems ranging from commercial applications to scientific studies. For instance, a set-valued attribute may correspond to the profile of a person, the tags of a post, the links or domain information of a webpage, and the tokens or q-grams of a document. To evaluate the influence of an object with a set of terms, one way is to compute the number of other objects that are a subset of that object. For instance, in the social media (e.g., Facebook, Twitter), a post that contains a large set of popular tags may be important to the service users who express interests by a set of tags. Since the number of posts and users is usually massive, we need to devise efficient query processing techniques to identify the influential objects.

In this thesis, we focus on the problem of *set containment join*. Given two collections Given two collections $\mathcal{R}$ and $\mathcal{S}$ of records, each of which contains a set of elements, the set containment join, denoted by $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$, retrieves all pairs $\{(r, s)\}$ where $r \in \mathcal{R}$, $s \in \mathcal{S}$, and $r \subseteq s$. As a fundamental operation on massive collections of set values, the set containment join has been extensively studied in the literature. According to the computing paradigms, the existing algorithms can be classified into two categories, namely *intersection-oriented* methods [Mam03, LFHDB15, BMGT15, KRS+16] and *union-oriented* methods [RPNK00, MGM02, MGM03, Mam03, JP05]. Nevertheless, we observe that two computing paradigms have their limits due to the nature of the intersection and union operators. Particularly, *intersection-oriented* method relies on the intersection of the relevant inverted lists built on the elements of $\mathcal{S}$. A nice prop-

erty of the *intersection-oriented* method is that the join computation is verification free. However, the number of records explored during the join process may be large because there are multiple replicas for each record in $\mathcal{S}$. On the other hand, the *union-oriented* method generates a signature for each record in $\mathcal{R}$ and the candidate pairs are obtained by the union of the inverted lists of the relevant signatures. The candidate size of *union-oriented* method is usually small because each record contributes only one replica in the index. Unfortunately, *union-oriented* method needs to verify the candidate pairs, which may be cost expensive especially when the join result size is large. As a matter of fact, the state-of-the-art *union-oriented* solution is not competitive compared to the *intersection-oriented* ones.

Furthermore, due to the limited computational resources (e.g., memory or CPU), it is often difficult to process large scale real-life datasets in a single machine. To alleviate this issue, a promising way is to extend the set containment join algorithms on top of MapReduce framework (e.g., Hadoop and Spark), which has attracted lots of interests in both academia and industry communities due to its high efficiency and scalability for batch processing tasks.

Challenges. There are two main challenges to efficiently process set containment join. Firstly, the existing methods have their limits due to the nature of intersection and union operators. It is not easy to develop new techniques to take advantage of their good properties, and meanwhile, avoid their limits as much as possible. Secondly, as far as we know, there is no existing work on computing set containment join using MapReduce framework. It is challenging on how to partition the two record collections such that good load-balance on cluster nodes can be achieved at a small communication cost.

Our Approach. In this thesis, we re-visit and design a new *union-oriented* method, namely *TT-Join*, where an efficient set containment join algorithm is de-

veloped based on two different prefix trees built on $\mathcal{R}$ and $\mathcal{S}$, respectively. Through comprehensive cost analysis on simple *intersection-oriented* and *union-oriented* methods, we show that the above two problems suffered by the *intersection-oriented* methods can be easily addressed by a new simple *union-oriented* method which uses the least frequent element as the signature. Not surprisingly, the new simple *union-oriented* method needs to verify candidates due to the inherent limit of *union-oriented* computing paradigm. Moreover, its pruning capability is limited by using only one element as the signature. To circumvent these limits, we propose a new prefix tree structure based on the k least frequent elements of the records within $\mathcal{R}$ such that we can (i) enhance the pruning power with a reasonable overhead, and (ii) integrate the *intersection* semantics to directly *validate* a significant number of join results without invoking the verification. To share the computational cost among records within $\mathcal{S}$, we also build a regular prefix tree on $\mathcal{S}$. Then we develop an efficient *TT-Join* algorithm to perform set containment join against two prefix trees.

To extend our techniques on top of MapReduce framework, we propose a novel signature-based distribution scheme, which dispatch records based on the above mentioned record signature (i.e., the least frequent element). Specifically, we first partition the element domain into N disjoin intervals, each related to a reduce node. Then, for a record $r \in \mathcal{R}$, we find the interval where its signature falls and dispatch r to the corresponding reduce node. For a record $s \in \mathcal{S}$, it will be dispatched to all reduce nodes whose corresponding intervals cover at least one element of s. With the help of careful designed element domain partition approaches that are guided by the join cost estimation on reduce nodes, our signature-based distribution mechanism can achieve good load-balance, low communication cost, and no duplicate in join results.

1.2 Contributions

In this section, we summarize our contributions in this thesis. We propose efficient techniques to process three important influence related queries. For each of these queries, we briefly describe our contributions.

Influence based Cost Optimization on User Preference. This is the first work to systematically study the problem of influence-based cost optimization for user preference functions, which is fundamental in many real applications such as marketing and advertising. Although we show the solution space can be reduced to a finite number of points by utilizing the classical k-level computation techniques, the computation cost is still very expensive due to the nature of the high combinatorial complexity of the k-level problem. To alleviate this issue, we develop a novel traverse based exact algorithm with time complexity $O(n)$ where n is the number of users by exploiting nice geometric properties of the problem in 2-dimensional spaces. To tackle the problem in general dimensional spaces ($d \geq 2$), we develop a space partition based exact algorithm following the random incremental paradigm, where efficient pruning techniques are devised to significantly enhance the performance. Then a sampling based approximate algorithm is devised to further speed up the computation (by up to orders of magnitude) with high accuracy. The extensive performance evaluation on real and synthetic data demonstrates the efficiency of our new techniques proposed in this thesis.

Categorical Top-k Spatial Influence Query. This is the first work to evaluate the influence of a facility regarding different categories of facilities. We show that the problem is NP-hard with respect to the number of object categories in the functional unit. To tackle the computational hardness, we develop an efficient framework following two main steps, possible participants finding and optimal functional

unit computation. Based on this framework, for the first step, novel and efficient pruning techniques are developed based on the nearest neighbor set approach. To find the optimal functional unit, we propose two algorithms, an exact algorithm and and efficient approximate algorithm with a factor of $2 - \frac{2}{m}$ performance guarantee, where m is the number of object categories in the query. Comprehensive experiments on both real and synthetic datasets demonstrate the effectiveness and efficiency of our techniques.

Efficient Set Containment Join. In this thesis, we study the problem of set containment join, which is fundamental and has many important applications in commercial and scientific fields. After comprehensive analysis on the advantages and limits of the existing solution, we propose a novel method, namely *TT-Join*. Particularly, we design a k least frequent elements based prefix tree structure, namely *kLFP-Tree*, to organize the records within $\mathcal{R}$. Together with a regular prefix tree constructed on records from $\mathcal{S}$, we develop an efficient set containment join algorithm. We conduct extensive experiments on 20 real-life datasets by comparing our method with 7 existing methods. The experiment results demonstrate that *TT-Join* significantly outperforms the existing algorithms on most of the datasets, and can achieve up to two orders of magnitude speedup. Furthermore, to support large scale of datasets, we extend our techniques to distributed systems on top of MapReduce framework. As far as we know, this is the first work to extend containment join system to a distributed environment.

1.3 Organization

This dissertation is organized as follows.

- Chapter 2 provides a survey of the related work.

- Chapter 3 presents our algorithms for the problem of influence based cost optimization on user preference.
- Chapter 4 describes our techniques for the problem of categorical top-k spatial influence query.
- Chapter 5 presents our techniques for efficient set containment join problem.
- Chapter 6 concludes our research and provides several possible directions for future work.

Chapter 2

Related Work

In this section, we provide an overview of the related work for each type of problems we studied in this thesis. In specific, we introduce the related work about influence-based queries on product and user preference data in Section 2.1. In Section 2.2, we introduce existing work related to categorical spatial influence query. In Section 2.3, we describe the existing techniques on set containment join related problems.

2.1 Influence based Cost Optimization on User Preference

Firstly, in Section 2.1.1, we review existing techniques for feature based top-k queries. Then, in Section 2.1.2, we introduce existing work on preference based influence queries. Last, Section 2.1.3 covers the dominance based influence queries.

2.1.1 Feature Based Top-k Queries

Given a set of interesting objects $\mathcal{O}$, Yiu et al. [YDMV07] propose the top-k spatial preference queries, which retrieve the k objects in $\mathcal{O}$ with the highest scores. The

score of an object is defined by the quality of *features* in its spatial neighborhood, where a feature is a type of facilities (e.g., restaurant or cafe). To efficiently process the query, the propose several algorithms. Among them, the branch and bound (BB) algorithm and feature joint (FJ) algorithm outperform others. BB achieves the best performance when the objects dataset is small whereas FJ is the best method when there are few and small feature datasets. Recently, Rocha-Junior et al. [RJVDN10] propose novel technique to speed up the performance of top-k spatial preference queries. To this end, they map the pairs of data and feature objects to a distance-score space, which in turn allows us to identify and materialize the minimal subset of pairs that is sufficient to answer a spatial preference query. Experimental results show that their methods perform better than BB and FJ [YDMV07] in both I/Os and execution time.

In many other applications (e.g., product data), an interesting object itself may be described by a set of attributes/features. Given an object p and a user preference function represented by a weight vector $\mathbf{w}$, the score of an object p regarding $\mathbf{w}$ is usually defined by a linear additive function as shown in the following equation:

$$\mathbf{w}(p) = \mathbf{w} \cdot p = \sum_{i=1}^{d} \mathbf{w}[i] \cdot p[i], \tag{2.1}$$

where d is the number of attributes for an object. Then given a user preference function $\mathbf{w}$, a top-k query retrieves k most interesting objects that give the best scores. As a crucial and fundamental tool in many applications, top-k query has been extensively studied in the literature [FLN03, APV07, IBS08, TXP07, THPP07, CBC$^+$00, HKP01, HP04, YYY$^+$03, MBP06].

Based on top-k query, several work are devoted to process interesting problems. Mouratidis *e*t al. [MP13] study the problem of computing immutable regions for subspace top-k queries. Specifically, given a query preference function $\mathbf{w}$, let $\mathcal{R}(\mathbf{w})$ denote the result set of a top-k query. An update to any query weight $\mathbf{w}[i]$ may

induce a change for $\mathcal{R}(\mathbf{w})$ or the ordering among its members, in which case we say that $\mathcal{R}(\mathbf{w})$ is perturbed; otherwise $\mathcal{R}(\mathbf{w})$ is preserved. Then for each dimension $j \in [1, d]$, the *immutable region* is defined at the widest range of $\mathbf{w}[j]$ values that preserve $\mathcal{R}(\mathbf{w})$, assuming that all other weights $\mathbf{w}[i]$ for $i \neq j$ remain constant. The immutable regions can serve to profile the robustness of a recommendation system to deviations in the query user preferences. A major limit of the above defined immutable region is that the validity interval is computed for each dimension separately (i.e., we assume all other weights remain constant while computing for a dimension). To get rid of this limitation, Zhang *e*t al. [ZMP14] propose *global immutable region*, which aims to compute all possible query weight setting that can preserve $\mathcal{R}(\mathbf{w})$. By exploiting the geometric properties of the problem, they develop efficient algorithms for global immutable region computation. Recently, Mouratidis *e*t al. [MZP15] consider the *maximum rank query*. Given a query object p, they compute the maximum rank it may achieve with respect to any possible preference function. In addition, they also report all the regions in the preference function domain where that rank is attained.

2.1.2 Preference Based Influence Queries

Given an object/product p, a reverse top-k query [VDKN10] returns a set of customers that find p appealing (i.e., for each customer in the query result, p is in the top-k query result set of the corresponding preference function). From the definition of reverse top-k query, it is naturally to consider it as a way to evaluate the influence/impact of an object against the preference functions and objects. That is, if the query result size is large, large amount of customers find p is attractive, which implies that p is popular. There are several follow-up work (e.g., [VDNK13, GMC$^+$13, CSLZ14]) aiming to improve the efficiency and scala-

bility of the reverse top-k query. To analyze the influence of the object for the given user preference functions and objects, some variants of the reverse top-k query has been studied in different scenarios. Vlachou *et* al. [VDNK10] introduce the problem of identifying the top-m most influential data objects where the influence of an object is defined by the cardinality of its reverse top-k result set. Koh *et* al. [KLC14] aim to find a set of products such that their total number of distinct attracted customers is maximized, which is essentially to maximize the coverage of the result set. Instead of coverage, Gkorgkas *et* al. [GVDN15] consider to select a set of products such that their diversity to each other is maximized. Recently, Wang *et* al. [WCZL15] combine those two criteria together and use a parameter to trade-off the importance of each criterion. Instead of analyzing exact preference function, Peng *et* al. [PW15] consider probabilistic preference functions and propose the k-Hit query which aims at finding a set of objects such that the sum of preference function probability in the result set is greatest. So far, all studies focus on the "hot" products that can be returned to some customers via reverse top-k query, while a large proportion of "unpopular" products cannot find any matching customers. Inspired by this observation, Zhang *et* al. [ZJK14] propose reverse k-ranks query, which finds for a given product, the top-k customers whose rank for the product is highest among all customers, to ensure 100% coverage for any given product. A common property of above studies is that they aim at finding a set of existing objects. Besides, none of them considers the cost function.

Recently, Gao *et* al. [GLC+15] investigate how to update a product at minimal penalty cost based on the quadratic functions such that it can attract a specific group of k users. It is worth mentioning that this is closest work to the problem studied in this thesis. Nevertheless, their techniques cannot be applied to our problem because we may attract arbitrary groups of k users and we aim to support

the general monotonic and convex cost function.

2.1.3 Dominance Based Influence Queries

Some existing work evaluate the influence of the objects based on the dominance relationship. In [LJ12], Lu *e*t. al study the problem of upgrading uncompetitive products economically, such that a set of k products will not be dominated by others. Ge *e*t al. [GMC+15] introduce a new dominance function to evaluate the goodness of a product and aim to find the best new product position under the given budget. Wan *e*t al. [WWI+09] introduce the problem of creating competitive products where a newly created product is competitive if it is not dominated by any existing products in the database. In order to maximize the profit, Wan *e*t al. [WWP11] consider price as one of the attribute of a product. In this work, a set of k products are selected from the candidate set of products such that they are not dominated by the existing products and meanwhile the profit of these k products is maximized. In [LOTW06], Li *e*t al. utilize the dominance relationship between products and customers to help manufactures position their products in the market. To this end, three types of queries called *dominant relationship queries (DRQs)* are proposed. In [ADV12, LKC13], each user preference is modeled as a point, and dominance relationships between products and user preferences are used to evaluate the goodness of a product.

It is evident that the problem studied in this thesis is different from all the dominance based influence queries because they either failed to consider user preference [LJ12, GMC+15, WWI+09, WWP11] or regard user preference as a point in the product space [LOTW06, ADV12, LKC13]. In our problem, a user preference is model as linear function represented by a weight vector. We argue that our model is more reasonable in a real world application because a user may find it hard to give

specific preferred values for the attributes of a product. In this case, a normalize weight vector to describe the preference on each attribute is recommended.

2.2 Categorical Top-k Spatial Influence Query

Our categorical top-k spatial influence query is closely related to several well studied research directions in spatial database area. We give a brief review for related work in this section.

2.2.1 NN Search and RNN Search

Nearest neighor search. In spatial database, an important and fundamental operator is nearest neighbor search (NN search). Given a set of objects $\mathcal{O}$ and a query object q, the NN search returns q's nearest object in $\mathcal{O}$, where the distance between two objects is defined by the Euclidean distance between them. Formally:

$$NN(q) = \{o \in \mathcal{O} | \forall p \in \mathcal{O} : d(q, o) \leq d(q, p)\}. \tag{2.2}$$

Two of the most popular algorithms are depth-first [RKV95] and best-first [HS99]. A direct work based on NN search is all nearest neighbor (ANN) query [CP07]. Given two datasets R and S, the ANN query finds a nearest neighbor in S for each object $r \in R$. In this thesis, we shall use NN search as a building brick to compute the optimal functional unit in our problem.

Reverse nearest neighor search. Given a set of object $\mathcal{O}$ and a query object q, the reverse nearest neighor search (RNN search) aims to find all objects in $\mathcal{O}$ that consider q as their nearest neighbor, which is defined as follows:

$$RNN(q) = \{o \in \mathcal{O} | \forall p \in \mathcal{O} : d(o, q) \leq d(o, p)\}. \tag{2.3}$$

In some applications, the objects in $\mathcal{O}$ are of two different categories, such as services facilities and customers. Based on this scenario, Korn *et* al. [KM00] propose the bichromatic reverse nearest search (BRNN search). In specific, given a set of facilities $\mathcal{F}$, a set of customers $\mathcal{C}$, and a query facility $q \in \mathcal{F}$, the BRNN query aims to find a set of customers whose nearest facility is q. Formally:

$$BRNN(q) = \{c \in \mathcal{C} | \forall p \in \mathcal{F} : d(c, q) \leq d(c, p)\}. \tag{2.4}$$

As a primary operator in spatial databases, BRNN has been widely used to discover the most "influential" objects. Existing studies following this direction are introduced in next section.

2.2.2 Maximum Influence Problem

Given a set of facilities/locations $\mathcal{F}$ and a set of customers $\mathcal{C}$, the maximum influence (Max-Inf) problem aims to find the most influential facilities where the influence of a facility f is defined by the number of customers in the BRNN of f. Xia *et* al. [XZKD05] propose the top-k most influential sites problem. In particular, given a query region Q, they aim to find the k most influential facilities inside region Q. A branch and bound method is developed, which takes advantage of a novel R-tree based metric called *minExistDNN*. By browsing both R-trees that are constructed on $\mathcal{F}$ and $\mathcal{C}$, respectively, the method can find the top-k most influential sites. Under the same problem settings, Du *et* al. [DZX05] try to find an optimal location. However, the work is limited to the L_1-norm space. Instead of inside a region, Gao *et* al. [GZCL09] aim to find k locations outside a region Q with largest optimality. Here, the optimality of a location p is defined to be the number of clients in Q whose distance to p is within a given threshold d_c. Specifically, each of the k retrieved locations has an optimality no less than that of any of the

remaining locations which are not in the result set. Instead of finding the most influential locations in the existing facilities, Cabello *et* al. [CDbL$^+$05] propose a similar problem called MAXCOV, which is to find a region Q in the space such that any location in Q has maximum influence. They give a theoretical analysis on the complexity of computing this problem, but no efficient algorithm is presented. In particular, they only give an algorithm whose running time is exponential to the size of $\mathcal{C}$. Later, Wong *et* al. [WÖY$^+$09] revisit this problem. By utilizing the principle of *region-to-point* transformation, which transforms the optimal *region* search problem to an optimal *point* search problem, they propose a polynomial algorithm to resolve the problem. Alternatively, Huang *et* al. [HWQ$^+$11] consider the case where there are a set of new candidate locations $\mathcal{N}$ apart from the existing locations $\mathcal{F}$. They try to find the k most influential locations in $\mathcal{N}$. Given a set of potential locations, Qi *et* al. [QZK$^+$12] also study this optimal location selection problem. The main difference between their work and the above studies is the definition of the objective function. They evaluate the goodness of a potential location by the average distance of the clients to their respective nearest facilities. Intuitively, a location can achieve the minimum average distance for the clients is optimal. More recently, Zhan *et* al. [ZZZL12] extend this problem to uncertain objects. A filtering and verification paradigm is applied to efficiently compute the k facilities with the highest expected influence scores.

The problem studied in this thesis differs from Max-inf problem in two major aspects. First, we consider facilities in different types rather than just one single type. Second, we evaluate the influence of a facility by its capability of forming the optimal functional unit of other facilities rather than the number of influenced clients.

2.2.3 Spatial Keyword Queries

Recently, spatial keyword queries have attracted tremendous attention. Chen *et* al. [CCJW13] give a comprehensive experimental evaluation to compare the state-of-the-art geo-textual indices which are widely used in spatial keyword queries. Let $\mathcal{O}$ be a set of spatial objects, each of which is associated with a set of keywords and a location. Given a query q with a set of keywords and a location, the related spatial keyword queries can be roughly divided into two categories: *soft cover* and *hard cover*.

Soft cover. In this category, we do not require that the answer covers all query keywords. A hybrid way, which uses a parameter α to trade off the balance between text relevancy and location proximity, is adopted to evaluate the quality of a candidate result. Existing methods all focus on this type of query. Cong *et* al. [CJW09, WCJ12] propose a new index structure called IR-Tree which applies a spatial-first strategy. That is, all objects/documents are organized into a spatial index tree (e.g., R-tree), and then each tree node is augmented with an inverted index to index all the documents contained in its minimum bounding rectangle (MBR). Rocha-Junior *et* al. [RJGJN11], on the other hand, adopts the textual-first strategy, where the search space of documents is first organized by keywords, and then for each keyword, a spatial data structure is employed to index the documents associated with this keyword. Very recently, Zhang *et* al. [ZCT14] propose a new approach which is based on modeling the problem as a top-k aggregation problem. Basically, they regard the spatial proximity as an extra keyword. Then, together with the existing m keywords in the query, they use the classic top-k aggregation algorithms to resolve the origin problem.

Hard cover. In this category, an answer must cover all query keywords. The top-k spatial keyword search (TOPK-SK) [FHR08, ZZZL13, ZZZL16] aims to re-

trieve the k closest objects in $\mathcal{O}$ each of which contains all keywords in the query. Zhang *et* al. [ZCM+09] propose the m-closest keywords (mCK) query, where each object only contains one single keyword. Given a query by a user, which consists of m keywords, mCK query aims to find the closest m objects each of which matches a keyword in the query, where the distance cost function is defined as the diameter of the m objects (i.e., the maximum distance between any two of the m objects). Although they develop an efficient priori-based search strategy to reduce the search space, their algorithm still runs exponential time in the worst case. This is because the mCK query is a NP-hard problem as shown in [FX10, GCC15] Guo *et* al. [GCC15] extend the mCK problem to the case where each object can have multiple keywords. They propose exact and approximate algorithms to resolve the problem. Most recently, Choi *et* al. [CPL16] propose SK-Cover problem, which considers a more sophisticated distance cost function that can lead to fewer proximate objects in the answer set. They prove that SK-Cover is not only NP-hard but also does not allow an approximation better that $O(\log m)$ in polynomial time. They develop a polynomial time approximation algorithm that achieves the asymptotically optimal approximation. Cao *et* al. [CCJO11] introduce the collective spatial keyword queries (CoSKQ), which is an extension of mCK problem. Different from mCK query, CoSKQ also specifies a query location. Accordingly, the distance cost function takes both the query location and the answer set into consideration. They present both exact algorithms and approximation algorithms along with the proof of NP-hardness of CoSKQ. Long *et* al. [LWWF13] propose a *distance owner-driven* approach for CoSKQ problem, which not only achieves much better scalability but also improves the constant approximation factor.

In the past decade, there are also a series of variant queries following spatial keyword queries, including but not limited to moving queries [WYJC11, WYJ13,

HLTF12, GCC15], reverse queries [LLC11, CCSC16, XLXJ17, CLZZ11, ZGC+17], why-not queries [CLH+15, CXL+16, CLXJ17], and other ad-hoc queries [LFX12, ZZZ+14, GZZC16, ZSZ+15].

If we regard the type of an object as a keyword contained in the object, a subprocedure of our problem is to find a set of objects S such that S covers all the query keywords and the cost of S is minimized. From this point of view, our problem is much related to the above CoSKQ and mCK queries. The major difference between our problem and the above studies is that we need to find the closest object set for each object when evaluating the influence of objects, while they are only interested in retrieving one closest object set. Besides, we use the sum of pairwise distance as the distance cost function, which is not used by the existing work.

2.3 Efficient Set Containment Join

Our efficient set containment join problem is related to query processing on set-valued data. In this section, we cover previous work studied for set containment joins, set containment queries, set similarity joins. In addition, we give brief review on join problems using MapReduce framework, since it is also utilized in this thesis.

2.3.1 Set Containment Joins

As a fundamental operator for analyzing set-valued data, the set containment joins have been well studied in the literature. In this section, we give brief review to existing studies on this problem, and will discuss the state-of-the-art algorithms in detail in Section 5.3.

Helmer *et* al. [HM97] are the first to directly investigate the implementation of

set containment joins. They evaluate several main memory algorithms following the nested-loop computing paradigm, and suggest that the signature-hash based method can achieve the best performance. Later, Ramasamy *et* al. [RPNK00] propose a disk-based algorithm, called *Partitioned Set Join* (PSJ), which aims at reduce the quadratic cost of nested-loop algorithms by utilizing the hash functions to partition the record collections into different buckets. They adopt the same in-memory processing strategy with that in [HM97]. Several follow-up studies [MGM02, MGM03] propose more sophisticated partitioning strategies (i.e., hash functions) to reduce the number of candidates in the partition buckets. To accelerate the signature based in-memory processing, Luo *et* al. [LFHDB15] recently propose a new algorithm based on a trie-based signature subsets enumeration method.

In [ZMR98], the authors show that the inverted file is significantly faster than signature-based indexes for set containment joins. Following this direction, Manoulis [Mam03] proposes a block nested loop join algorithm which first builds inverted index on the right-hand record collection $\mathcal{S}$, and then applies the intersection operator to calculate the join results for each record in the left-hand record collection $\mathcal{R}$. Instead of processing each record in $\mathcal{R}$ individually, Jampani *et* al. [JP05] propose PRETTI which builds a prefix tree on $\mathcal{R}$ to share computation cost while handling similar records. To reduce the size of the prefix tree, Luo *et* al. [LFHDB15] introduce an extension of PRETTI, namely PRETTI+, which employs a compact prefix tree to replace the prefix tree in PRETTI. To avoid exploring many inverted lists for the large size records within $\mathcal{R}$, Bouros *et* al. [BMGT15] propose a new algorithm, called LIMIT, which only builds a prefix tree with limited height, and performs the join process following a two-phase procedure which involves *candidates generation* and *candidates verification*. Most recently, Kunkel *et* al. [KRS+16] propose a two-tree based method to improve the performance of intersection based

method by exploiting advanced index technique on $\mathcal{S}$.

2.3.2 Set Containment Queries

As a closely related problem, set containment query has also caught much research interest in the literature. Yan *et* al. [YGM94] study the problem of selective dissemination of information, which can be regarded as set containment query under the boolean model. They investigate several index structures for disk implementations, and provide analysis and simulation results to compare their performance under different scenarios. In [TPVS06], the authors superimpose a trie-tree on top of the classic inverted file, which optimize the indexing set-valued data with skewed item distributions. Later, Terrovitis *et* al. [TBV$^+$11] present a $B+$ tree based indexing structure, which aims to reduce the I/O cost for containment queries against set-valued data with skewed item distributions. To facilitate the query processing, Chaudhuri *et* al. [CCKS07] build inverted indexes for careful chosen frequent term combinations in addition to the traditional single-term based inverted indexes. Li *et* al. [LLL08] propose an efficient list merging algorithm to solve the generalized T-occurrence query problem. By setting T to the size of the query record, their techniques can be immediately employed to process set containment search. Following the similar indexing strategies in [YGM94], Hmedeh *et* al. [HKC$^+$12] develop efficient main memory based algorithms for publish/subscribe systems. Ibrahim *et* al. [IF13] study the containment queries on nested sets, where an element in a record might be a set as well. Agrawal *et* al. [AAK10] study the problem of error-tolerant set containment search. To boost the query performance, they propose an frequent element based index structure that builds inverted index on careful chose element set. Zhang *et* al. [ZCS$^+$12] investigate the problem of probabilistic set containment, where the contents of the sets are uncertain. Their proposed solution

relies on an inverted file that is augmented with item's probability of belonging to a certain record. Zhu *e*t al. [ZNPM16] study the problem of domain search where they use the Jaccard set containment score as the measure of relevance of two domains. Locality sensitive hashing based index structure is developed to efficiently process the problem against large scale domain datasets.

2.3.3 Set Similarity Joins

Set similarity join is fundamental. In the literature, two categories of set similarity join problems are well studied, namely exact set similarity join and approximate set similarity join.

Exact set similarity join. The existing solutions for exact set similarity joins all follow the filtering-verification framework, which can be further classified into two categories based on the filtering mechanism, including prefix-filter based algorithms and partition-filter based algorithms. Bayardo *e*t al. citebayardo2007scaling first propose prefix-filter based framework. Based the same framework, Xiao *e*t al. [XWLY08] introduce two more filter rules, namely positional filter and suffix filter. Later, Mann *e*t al. [MA14] devise an optimized length filter. In [WLF12], an adaptive length prefix is developed to strengthen filtering power. Other heuristics [RH11, BGM12] are introduced to improve the algorithm. By considering the relations of records to share computation cost in join processing, Wang *e*t al. [WQL$^+$17] recently propose an efficient prefix-filter based algorithm. In [MAB16], an efficient candidate verification algorithm is proposed that is beneficial to all existing prefix-filter based algorithms. For partition-filter based algorithms, a two-level algorithm with partitioning and enumeratiing to find exact similar sets is devised in [AGK06]. Based on the pigeonhole principle, Deng *e*t al. [DLWF15] develop a efficient partition scheme to partition the sets into sev-

eral subsets. Then inverted lists on the subsets are utilized to filter the candidate pairs.

Approximate set similarity join. The existing solutions for approximate set similarity joins are mostly based on the *locality sensitive hashing* (LSH) technique [IM98], which is a probabilistic scheme by hashing similar records into the same bucket with high probability. Broder [Bro97] propose *MinHash* to quickly estimate the Jaccard similarity between two records. Zhai *et* al. [ZLG11] develop a probabilistic algorithm for similarity search with a low threshold on records with hight dimension. In [SP12], a Bayesian algorithm, called BayesLSH, is introduced to extend LSH for candidate pruning and similarity estimation. Chakrabarti *et* al. [CP15] utilize sequential hypothesis testing on LSH to adaptively prune candidates aggressively and provider tighter qualitative guarantees over BayesLSH. Rather than Jaccard similarity, other MinHash based algorithms [SL15, ZNPM16] are also developed for approximate set similarity search/join targeting set containment similarity.

2.3.4 Join Problems Using MapReduce

To the best of our knowledge, there is no existing work that extends set containment join to MapReduce framework. Recently, Kunkel *et* al. [KRS+16] propose a parallel algorithm PIEJoin to compute the set containment join. Nevertheless, they achieve parallelization by creating a task thread for each recursive call of the crucial *search* function in their approach. This method does not follow the MapReduce framework which involves two important operations, namely *map* and *reduce*.

In the past decade, MapReduce has attracted tremendous interests in both academia and industry communities. Its high efficiency and scalability for batch processing tasks provides elegant solutions for many join problems. Here, we focus

on the set similarity join problem in MapReduce, which is closely related to our set containment join problem. Vernica *et* al. [VCL10] propose a prefix-based partition strategy for set similarity join based on prefix filter, which states that two records must share at least one common element in their prefixes if they are similar. After constructing the inverted index on elements of record prefix, records in the same inverted list are then dispatched to the same task node. Metwally *et* al. [MF12] present V-SMART-JOIN, which similarly builds inverted index on elements and compute the similarity of record pairs by sharing the computation in the element level using MapReduce. Deng *et* al. [DLH+14] propose a partition-based framework to solve set similarity join in MapReduce. The element set of a record is partitioned into $U+1$ disjoint segments, where U is a dissimilarity upper bound. Similar record pairs would be distributed to at least one common task nodes where they share the same segment. Afrati *et* al. [ASM+12] conduct theoretical analysis against multiple MapReduce based similarity join algorithms, where they analyze the map, reduce, and communication cost.

Chapter 3

Influence based Cost Optimization on User Preference

3.1 Overview

In many real applications, the interest of a user towards objects (e.g., products) with multiple attributes may be quantitatively expressed by a preference function. For instance, as shown in Fig. 5.1(a), suppose users are interested in two features of the laptop: product weight and heat emission. Then, a laptop p can be modeled as a point in a 2-dimensional space in Fig. 5.1(c), where $p[1]$ and $p[2]$ denote the weight and emission values of the laptop. Meanwhile, suppose we use the popular linear function as the preference function. As shown in Fig. 5.1(b), a linear function $\mathbf{w}$ is a weighted vector where $\mathbf{w}[1]$ and $\mathbf{w}[2]$ denote the importance (weight) of the product weight and the heat emission, respectively. Given a user preference function $\mathbf{w}$ and an object p, we use $\mathbf{w}(p)$ to denote the score of p w.r.t the user preference function, where $\mathbf{w}(p) = \mathbf{w}[1] \times p[1] + \mathbf{w}[2] \times p[2]$. By doing this, objects are ranked for a user according to her preference function. Generally, we say an

object is *attractive/appealing* to the user if it is ranked the first by her. Given a set of preference functions which are learned or collected from users [QGJ15], we can evaluate the influence/impact of an object by the number of users it can attract. Analyzing the influence of an object regarding the user preference functions and other competing objects is fundamental in a wide range of applications such as marketing and advertising. In recent years, there is a surge of interest on the study of influence analysis on preference functions such as identifying the most influential objects [VDNK10, ADV12, LKC13, PW15] and supporting why-not questions [GLC+15].

In this chapter, we formulate the problem of influence-based cost optimization on user preference functions. Given a set of n user preferences, a set of objects in a d-dimensional space, and a cost function, we aim to find the cost optimal position for a new object in the space such that it attracts at least k users competing with the existing objects. Below is a motivating example in the context of marketing analysis.

Example 3.1 (**Product Positioning**). *Marketers may often conduct marketing research studies to decide the new product positioning by analyzing consumer preferences, the competitors' products and the manufacturing cost. As depicted in Fig. 3.1, there are a set $\mathcal{P}$ of laptops $\{p_1, p_2, ..., p_4\}$ (Fig. 3.1(a)) where each laptop is represented by a point in a 2-dimensional space (Fig. 3.1(c)) and the smaller value is preferred. Meanwhile, there are a set $\mathcal{W}$ of customer preferences $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_5\}$ of 5 users where each $\mathbf{w}$ in $\mathcal{W}$ is a linear preference function (Fig. 3.1(b)). Moreover, we assume the cost function for a point p is $f(p) = 1/p[1] + 1/p[2]$.*

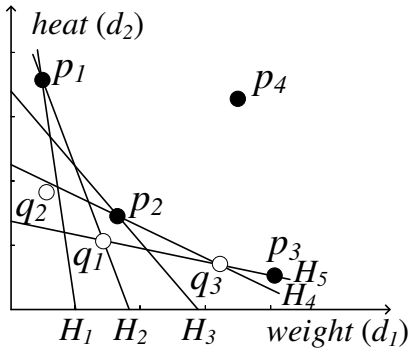
Suppose a company wants to design a new type of laptop (i.e., a point in the 2-dimensional space) to attract at least 4 out of the 5 users; that is, occupying at least 80% of the market share in this example. Following the convention [GLC+15], we

Id	weight	heat
p_1	0.6	3.6
...	...	...
p_4	3.5	3.3

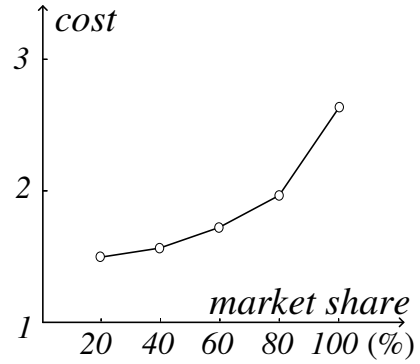
(a) Laptops

Id	weight	heat	top-1
$\mathbf{w}_1$	0.8	0.2	p_1
...	...	...	...
$\mathbf{w}_5$	0.2	0.8	p_3

(b) User preference functions



(c) Laptop space



(d) Optimal laptop cost

Figure 3.1: A motivation example

use a preference line H_i to represent a user linear preference function $\mathbf{w}_i$ where H_i is perpendicular to $\mathbf{w}_i$ and contains its top-1 result. For example, in Fig 3.1(c), H_1 is perpendicular to preference function $\mathbf{w}_1$ and contains p_1 which is the top-1 result of $\mathbf{w}_1$. Then, a new point q is attractive to a user $\mathbf{w}_i$ if q is below or lies on H_i. It is immediate that any point below and lying on at least 4 preference lines (e.g., q_1 and q_2 in Fig. 3.1(c)) in this example satisfies the influence constraint. Considering the manufacturing cost, it is desirable to identify the point which satisfies the influence constraint and has the lowest cost, which is q_1 in this example. Fig. 3.1(d) plots the growth of the optimal manufacturing cost with the increase of the targeted market share occupation rate (i.e., influence in this thesis), which provides a comprehensive view for the new product positioning.

Challenges. A straightforward implementation of our problem is to explicitly enumerate $\binom{n}{k}$ possible groups of k out of n user preference functions. For each group of k users, we find the cost optimal solution to attract these particular k

users. Then we may come up with the optimal solution. However, this approach is computational cost-prohibitive for even a small n and k values. Moreover, we aim to support the general monotonic and convex cost function, instead of a specific one. These present great computational challenges to our problem.

Although efficient algorithms have been proposed to compute the influence of an object (e.g., [VDKN10, GMC+13, VDNK13, CSLZ14]) and identify the most influential object [VDNK10, ADV12, KLC14], it is infeasible to enumerate the infinite possible positions of the new object. Moreover, their techniques do not consider the cost function. Recently, Gao *e*t al. [GLC+15] develop a novel algorithm to find the cost optimal point to attract a specific group of k users. However, we cannot afford to enumerate $\binom{n}{k}$ possible groups in our problem. Besides, their technique only supports quadratic cost function since quadratic programming method is employed.

To tackle the computational challenge, we need to develop efficient query processing techniques. By taking advantage of the k-level computing techniques (e.g., [Mul91, ADBMS98]) and the monotonicity and convexity of the cost function, we reduce the solution space to a finite number of points on the k-level of n hyperplanes derived from n preference functions and objects. Although we can immediately come up with two baseline algorithms by applying the on-the-shelf k-level computing techniques: traverse based algorithm [AACS98, CSLZ14] for the 2-dimensional space and randomized incremental algorithm [Mul91] for the general multi-dimensional space, the computational cost is still expensive due to the high combinatorial complexity of k-level, $O(nk^{1/3})$ and $O(n^{\lfloor d/2 \rfloor}k^{\lceil d/2 \rceil})$ for 2 and d-dimensional spaces, respectively.

In this thesis, we develop an efficient traverse based algorithm for 2-dimensional spaces with time complexity $O(n)$ by exploiting the nice geometric properties of

our problem in 2-dimensional spaces. It significantly improves the time complexity of the baseline approach, which is $O(n^2k^{1/3})$.

To deal with the general multi-dimensional spaces, we also develop a space partition based algorithm. Although we cannot improve the theoretical time complexity of the baseline solution, a variety of pruning techniques are developed to significantly enhance the performance of the algorithm by utilizing the cost bounds, influence bounds and local dominance relationship. Observe that it is cost expensive to maintain the k-level of hyperplanes in a partition, we further devise a novel sampling based approach such that the problem can be reduced to the classical *half-space intersection* problem in each round of samples. Empirical study shows that our sampling technique can significantly outperform the exact algorithm with high accuracy for space with $d \geq 3$.

Raodmap. The rest of the chapter is organized as follows. In Section 3.2, we first formally formulate the problem, and then introduce some preliminaries including naive solutions. Traverse based method is introduced in Section 3.3. Space partition based method is devised in Section 3.4. Extensive experiments are reported in Section 3.5. Finally, Section 3.6 concludes this chapter.

3.2 Background

In Section 3.2.1 we propose and formulate the problem of influence-based cost optimization. Section 3.2.2 introduces the k-level techniques, which is employed in Section 3.2.3 to develop naive solutions. Table 3.1 summarizes the mathematical notations used throughout this paper.

Notation	Definition
p, $\mathcal{P}$	a tuple (point), a set of tuples (points)
$p_1 \prec p_2$	point p_1 dominates point p_2
$\mathbf{w}$ ($H_{\mathbf{w}}$)	a user preference function (hyperplane)
$\mathcal{W}$ ($\mathcal{H}_{\mathcal{W}}$)	a set of user preference functions (hyperplanes)
n	the number of preference functions/hyperplanes
m	the number of anchor tuples
$H, \mathcal{H}$	hyperplane, a set of hyperplanes
$f, f(p)$	a cost function (cost of point p)
$I(p)$	influence score of point p

Table 3.1: The summary of notations

3.2.1 Problem Definition

Let $\mathcal{P}$ denote a set of *tuples*. Each tuple $p \in \mathcal{P}$ is described by a point $(p[1], p[2], ..., p[d])$ in a d-dimensional space $\mathcal{R}^d$ where $p[i]$ represents the i-th dimension (i.e., attribute) value of the point p. Without loss of generality, in this thesis, we assume that smaller values are preferred and $p[i] > 0$ for $1 \leq i \leq d$. In the following, we use "tuple" to denote an element in $\mathcal{P}$, and "point" to denote any position in $\mathcal{R}^d$, which might be an existing tuple in $\mathcal{P}$.

Given two points p_1 and p_2, we say p_1 *dominates* p_2, denoted by $p_1 \prec p_2$, if p_1 is not larger than p_2 on all dimensions and p_1 is smaller than p_2 on at least one dimension.

For each user, we assume there is a *preference function* $\mathbf{w}$ which is a linear function represented by a weight vector ($\mathbf{w}[1]$, $\mathbf{w}[2]$, ..., $\mathbf{w}[d]$) where $\mathbf{w}[i]$ indicates the significance of the i-th dimension for the user. Following the convention [VDKN10, GLC+15], we assume $\mathbf{w}[i] \geq 0 (1 \leq i \leq d)$ and $\sum_{i=1}^{d} \mathbf{w}[i] = 1$. Then the score of a point p with respect to the preference function $\mathbf{w}$, denoted by $\mathbf{w}(p)$, is defined as their dot product, i.e., $\mathbf{w}(p) = \mathbf{w} \cdot p = \sum_{i=1}^{d} \mathbf{w}[i] \cdot p[i]$. Therefore, the smaller score values are preferred in this paper. By $\mathcal{W}$, we denote the set of n preference functions (users).

In this thesis, we say a point p is *attractive* to a user if p is ranked the first regarding her/his preference function $\mathbf{w}$, i.e., $\mathbf{w}(p) \leq \mathbf{w}(p')$ for any point $p' \in \mathcal{P} \cup \{p\}$. Note that p might be an arbitrary point in the space $\mathcal{R}^d$.

Below we define the influence set of a point which reflects its popularity among $|\mathcal{W}|$ users (i.e., preference functions).

Definition 3.1 (**Influence set** and **Influence score**). *For a given point $p \in \mathcal{R}^d$, we use $S(p)$ to denote the set of preference functions of $\mathcal{W}$ attracted by the point p, namely influence set of p. More specifically, we have $S(p) = \{\mathbf{w}|\ p$ is ranked the first among points in $\mathcal{P} \cup \{p\}$ w.r.t $\mathbf{w} \in \mathcal{W}\}$. The influence score of p, denoted by $I(p)$, is the size of $S(p)$.*

Example 3.2. *Consider point p_2 in Fig. 3.2. The influence set $S(p_2)$ consists of three preference functions, namely $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$. Therefore, $I(p_2) = 3$.*

We assume there is a cost function f where $f(p)$ represents the cost of a point p. In practice, the function f may vary with respect to different scenarios (e.g., different types of tuples). Here, we assume f is a *monotonic* function, i.e., $f(p_1) \geq f(p_2)$ if $p_1 \prec p_2$ for any two points p_1 and p_2. This is rather intuitive because the cost would not decrease if the point is upgraded. Moreover, we also assume f is a convex function.

Suppose our target influence score of the new point is k. We say q is a k-critical point if it is the most cost-effective choice, which is formally defined as follows.

Definition 3.2 (**k-critical point**). *Given a set $\mathcal{P}$ of tuples, a set $\mathcal{W}$ of preference functions, a cost function f and a number k ($1 \leq k \leq |\mathcal{W}|$), we say a point $q \in \mathcal{R}^d$ is a k-critical point if $I(q) \geq k$ and $f(q) \leq f(p)$ for any point $p \in \mathcal{R}^d$ with $I(p) \geq k$.*

Problem Statement. Given a set $\mathcal{P}$ of tuples, a set $\mathcal{W}$ of user preference functions, a cost function f, and a number k ($1 \leq k \leq |\mathcal{W}|$), we aim to efficiently

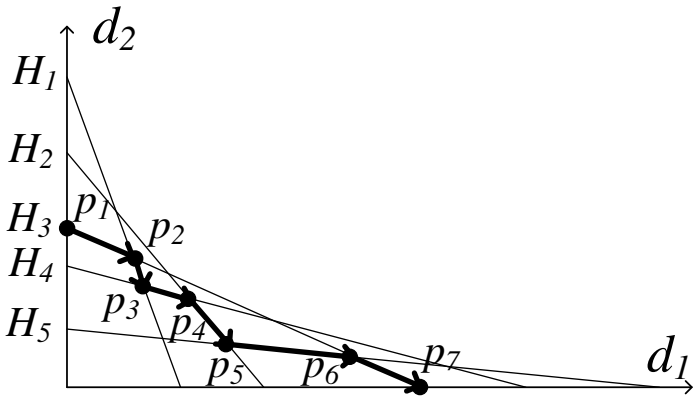


Figure 3.2: The 3-level of a set of hyperplanes

identify the most cost-effective point in $\mathcal{R}^d$ to attract at least k users; that is, find the k-critical point.

3.2.2 K-Level Problem

Given a weight vector $\mathbf{w}$ and a point p in space $\mathcal{R}^d$, we use $H(\mathbf{w}, p)$ to denote a *hyperplane* which is perpendicular to the vector $\mathbf{w}$ and contains the point p. Given a set $\mathcal{H}$ of hyperplanes, the *upper score* of a point p is the number of hyperplanes in $\mathcal{H}$ which are above or contain p.

Definition 3.3 (k**-level** $lev_k(\mathcal{H})$)**.** *The k-level of a set of hyperplanes $\mathcal{H}$, denoted by $lev_k(\mathcal{H})$, is defined to be the point set $\mathcal{Q} \subset (\mathcal{R}^d \cap \mathcal{H})$ with upper score k.*

Example 3.3. *The bold segments in Fig. 3.2 denote the* 3*-level of a set of hyperplanes in a* 2*-dimensional space.*

The k-level problem is a classical problem and its computation is a core component of a wide variety of algorithms (e.g., [Mul91, ADBMS98, AACS98]). Particularly, the current best known upper bound for the combinatorial complexity (i.e., the number of faces) of k-level is $O(nk^{1/3})$ [Dey98] and $O(nk^{5/3})$ [AACS98] for 2 and 3-dimensional spaces, respectively. It turns out to be $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ [CS89] in

higher dimensions where $d \geq 4$. Recently, k-level has also been widely applied in the database community to facilitate rank queries (e.g., [CSLZ14, DGKS07, YAY12]).

3.2.3 Reduce the Search Space

In this subsection, we show that our influence-based cost optimization problem can be solved by taking advantage of the k-level technique, where we can identify a finite number of possible points. Before that, we introduce the concept of *preference hyperplane*.

Definition 3.4 (**Preference hyperplane**). *For a given preference function (i.e., vector)* $\mathbf{w} \in \mathcal{W}$*, we say a hyperplane, denoted by* $H_{\mathbf{w}}$*, is the preference hyperplane of* $\mathbf{w}$*, where* $H_{\mathbf{w}}$ *is perpendicular to* $\mathbf{w}$ *and contains the favorite tuple of* $\mathbf{w}$ *in the tuple set* $\mathcal{P}$*; that is,* $H_{\mathbf{w}} = H(\mathbf{w}, p)$ *where* $p \in \mathcal{P}$ *and* $\mathbf{w}(p) \leq \mathbf{w}(p')$ *for any* $p' \in \mathcal{P}$*.*

By $\mathcal{H}_{\mathcal{W}}$ we denote the corresponding hyperplanes of the preference functions in $\mathcal{W}$, namely *preference hyperplanes*. For presentation simplicity, we may drop the subscript of $\mathcal{H}_{\mathcal{W}}$ whenever it is clear in the context. Based on the geometric property of $H_{\mathbf{w}}$, it is immediate that a new point $q \in \mathcal{R}^d$ is attractive to $\mathbf{w}$ if it is below or contained by $H_{\mathbf{w}}$. Thus, we have the following Lemma.

Lemma 3.1. *Given preference hyperplanes* $\mathcal{H}$*, the influence score of point* $q \in \mathcal{R}^d$*,* $I(q)$*, w.r.t* $\mathcal{H}$ *is equal to the upper score of* q *w.r.t* $\mathcal{H}$ *(i.e., the number of hyperplanes containing or above* q*).*

According to Definition 3.3, we have $I(q) = k$ for any point $q \in lev_k(\mathcal{H})$. Following theorem indicates that we only need to check a limited number of points in the space to identify the k-critical point.

Theorem 3.1. *Given a number* k *with* $1 \leq k \leq n$*, the* k*-critical point* q *can only be one of the intersection points or tangent points on* $lev_k(\mathcal{H})$*'s faces.*

Proof. First, we show q is on $lev_k(\mathcal{H})$. According to Definition 3.3, we know that q must be below or lying on $lev_k(\mathcal{H})$. For any point p below $lev_k(\mathcal{H})$, we can always find a point p' on $lev_k(\mathcal{H})$ such that $p \prec p'$ and hence $f(p) > f(p')$. Therefore, we find that q is on $lev_k(\mathcal{H})$.

Then, we show that q is one of the intersection points or tangent points on $lev_k(\mathcal{H})$'s faces. As we know, $lev_k(\mathcal{H})$ consists of 0-faces (vertices), 1-faces (edges), ..., and $(d-1)$-faces (facets) in $\mathcal{R}^d$. Given a l-face $\mathcal{F}$, $0 \leq l \leq d-1$, the affine hull of $\mathcal{F}$ is called a l-flat. The terms "points", "lines", and "planes" are used to designate 0-flats, 1-flats, and 2-flats in $\mathcal{R}^2$ or $\mathcal{R}^3$ (See [EOS86] for details). For any $(d-1)$-face $\mathcal{F}$ in $lev_k(\mathcal{H})$, the cost function f can achieve the minimum value at the tangent point p_o on the corresponding $(d-1)$-flat (See [CC87] for the continuity and differentiability of monotone real functions) because f is a convex function. Since $\mathcal{F}$ is a closed region, p_o might locate outside of $\mathcal{F}$. In this case, the minimum value of f must be achieved on one of the $(d-2)$-face boundaries as f is monotonic and convex. We repeat this process until we reach edges. If we still find that the tangent point is not on the edge, we know the minimum cost point must be one of the two end vertices which are the intersection points of the hyperplanes. Thus, the proof is complete. □

Example 3.4. *As shown in Fig. 3.2, the* 3*-critical point* q *can only be one of the points* p_1*,* p_2*, ...,* p_7 *together with several possible tangent points on the bold segments.*

3.2.4 Naive Solutions

Following Theorem 3.1, we may visit the faces of the k-level to find the candidate intersection points and tangent points. Then, the point with the lowest cost is the k-critical point. In the literature, *traverse based* and *randomized incremental*

approaches have been proposed to construct the k-level of a set of hyperplanes, which can be easily extended for our problem.

Traverse based Method. In [CSLZ14, AACS98], the k-level is constructed by traveling along the adjacent faces on $lev_k(\mathcal{H})$ in a 2-dimensional space. In general, we start from the left most k-level point. Whenever we meet an intersection point of two [1] hyperplanes (i.e., lines in a 2-dimensional space) on the way to the right, we make a turn (i.e., switch to another line). We terminate this traveling process once reaching the right most k-level point. The time complexity of the above traverse algorithm is $O(n^2k^{1/3})$ because it takes $O(n)$ time to identify the next intersection point in the worst case and the number of faces on k-level is bounded by $O(nk^{1/3})$.

Example 3.5. *As illustrated in Fig. 3.2, to construct the* 3*-level of* $\mathcal{H}$*, we start from point* p_1 *and traverse along* H_3*. Then we meet an intersection point* p_2*, and therefore we make a turn to travel along* H_1*. We repeat this process until we reach the right most point* p_7*.*

The above algorithm can be easily extended to our problem with the same time complexity $O(n^2k^{1/3})$, where the cost of the tangent point on each visited line on k-level is also calculated. However, this traverse based method cannot effectively take advantage of the minimum cost for the candidate points seen so far because we have to visit these candidates along a fixed order. Thus, in Section 3.3, we propose a novel algorithm for 2 dimensions where a flexible traverse order is employed to utilize some pruning techniques.

As mentioned in [CSLZ14], it is difficult to apply the traverse based approach for higher dimensional spaces. The main obstacles are the exponential growth of the combinatorial complexity with the dimensionality and the difficulty to enumerate the adjacent unvisited faces at each intersection point. Consequently, we resort to

[1] The technique can be trivially extended for intersection points with more than two lines

the randomized incremental approach for higher dimensional spaces ($d > 2$).

Randomized Incremental Method. Efficient algorithm has been proposed in [Mul91] following the randomized incremental paradigm [CS88]. In a nutshell, we start with d randomly sampled hyperplanes to construct the initial k-level structure. Then the remaining $(n - d)$ hyperplanes are added one by one in a random order. For each arriving hyperplane, we will refine the k-level structure for hyperplanes seen so far by discarding the faces already above the current k-level. The expected time complexity of the algorithm is $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ in a d-dimensional space.

This randomized incremental algorithm can be trivially extended to identify the k-critical point. Unfortunately, as shown in our empirical study, the performance is unsatisfactory even for the low dimensional space because the time complexity is very sensitive to n (i.e., the number of hyperplanes). In Section 3.4, we propose the partition strategy and a variety of pruning techniques to reduce the number of hyperplanes involved in the computation of each partition, and hence significantly improve the performance. We also devise sampling method to further enhance the computational performance by orders of magnitude with high accuracy.

3.3 Traverse Based Method (2 dimension)

In this section, we propose an efficient method for 2-dimensional spaces.

3.3.1 Motivation

As discussed in Section 3.2, the main drawback of the traverse based approach is that we have to visit the k-level following the turning points sequentially, and hence cannot prune candidate points based on the minimum cost seen so far. In

this section, we circumvent the problem by partitioning the solution space into a number of groups such that we can prune some regions (i.e., candidate points in these regions) during the computation. Meanwhile, we can also quickly identify candidate points for each survived region without the knowledge of the previous turning point. Below is the motivation of our method.

In this thesis, we say a tuple $a \in \mathcal{P}$ is an *anchor tuple* if a is attractive to at least one user. Let $\mathcal{A}$ denote all m anchor tuples for the given tuples $\mathcal{P}$ and preferences $\mathcal{W}$. The preference hyperplanes (lines in this Section) $\mathcal{H}$ is partitioned into m disjoint groups [2]. By $\mathcal{H}_a$, we denote the preference lines crossing the anchor tuple a and we assume the lines are sorted by their slopes. For a pair of anchor tuples a_i and a_j with $i < j$ in $\mathcal{A}$, we use $R_{i,j}$ to denote the region constructed by their boundary lines, namely *anchor-pair region*. In Fig. 3.3, we show three anchor-pair regions (depicted in dashed quadrilaterals) constructed from three anchor tuples a_1, a_2 and a_3. Clearly, each line of $lev_k(\mathcal{H})$ intersects with at least one region.

In general, we follow the branch and bound paradigm to identify the optimal solution. For each anchor-pair region R, we can easily come up with the lower bound of the cost for any point in R. A region is immediately pruned if its cost lower bound is larger than the current lowest cost. Moreover, we can also prune a region based on the upper and lower bounds of the influence scores for any point in the region. Meanwhile, for each survived region, we can quickly identify the candidate points because the lines from the same anchor tuple are sorted by their slopes.

The key of our new method is that, by utilizing some nice geometric properties of our problem in 2-dimensional spaces, we ensure that only $3m$ out of the $\frac{m^2}{2}$ regions are touched during the computation, while each line in $\mathcal{H}$ is visited at most twice.

[2]For the lines containing more than one anchor tuple, we randomly choose one.

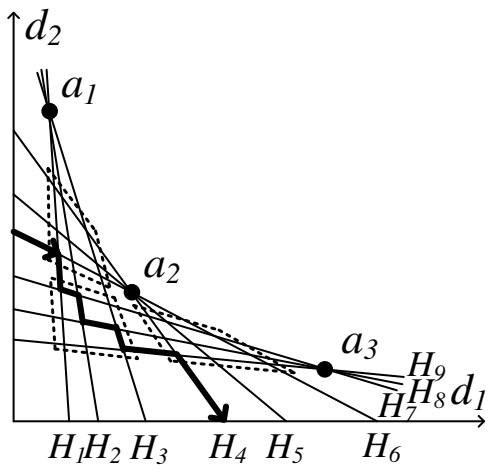


Figure 3.3: Anchor-pair(AP) regions

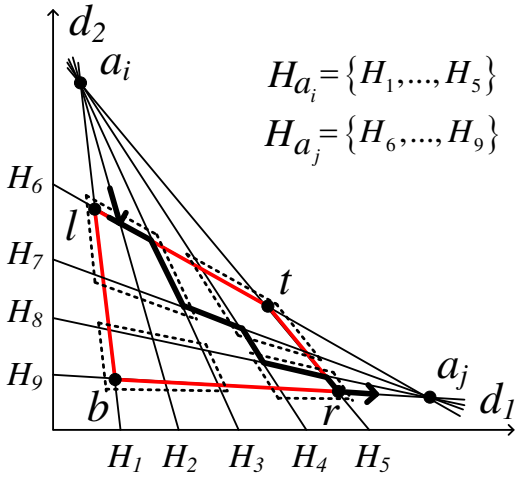


Figure 3.4: Traverse AP-region

Thus, we come up with a new algorithm with time complexity $O(m+n) = O(n)$ since $m \leq n$.

3.3.2 Anchor-Pair Region

In this subsection, we first introduce some geometric properties relevant to the anchor-pair region. Then pruning and traverse methods are presented.

According to [CBC+00] and the fact that the addictive linear function (i.e., $\mathbf{w}[i] \geq 0$ with $1 \leq i \leq d$) is adopted, $\mathcal{A}$ is a subset of the convex hull points of $\mathcal{P}$. Without loss of generality, we assume anchor tuples $\{a_1, \ldots, a_m\}$ from $\mathcal{A}$ are ordered by the values of their first dimension where $a_i[1] < a_j[1]$ for any $1 \leq i < j \leq m$. Moreover, we have $a_i[2] > a_j[2]$ because, otherwise, a_j cannot be an anchor tuple.

Below, we introduce two important lemmas.

Lemma 3.2 indicates that the slopes of the sorted preference lines are well divided by the anchor tuples.

Lemma 3.2. *We have slope*$(H) >$ *slope*(H') *if* $H \in \mathcal{H}_{a_i}$ *and* $H' \in \mathcal{H}_{a_j}$ *where*

$i < j$.

Proof. Let $\mathbf{w}$ and $\mathbf{w}'$ be the preference functions of H and H' respectively. According to the definition of anchor tuple, we have that $\mathbf{w} \cdot a_i < \mathbf{w} \cdot a_j$ and $\mathbf{w}' \cdot a_j < \mathbf{w}' \cdot a_i$. We then have the following two inequalities.

$$\mathbf{w}[1]a_i[1] + \mathbf{w}[2]a_i[2] < \mathbf{w}[1]a_j[1] + \mathbf{w}[2]a_j[2] \tag{3.1}$$

$$\mathbf{w}'[1]a_j[1] + \mathbf{w}'[2]a_j[2] < \mathbf{w}'[1]a_i[1] + \mathbf{w}'[2]a_i[2] \tag{3.2}$$

By multiplying these two inequalities, we get $(\mathbf{w}[2]\mathbf{w}'[1] - \mathbf{w}[1]\mathbf{w}'[2])(a_i[2]a_j[1] - a_i[1]a_j[2]) < 0$. Since $a_i[1] < a_j[1]$ and $a_i[2] > a_j[2]$, we have $a_i[2]a_j[1] - a_i[1]a_j[2] > 0$ and hence $\mathbf{w}[2]\mathbf{w}'[1] - \mathbf{w}[1]\mathbf{w}'[2] < 0$. Thus, $\mathbf{w}[1]/\mathbf{w}[2] > \mathbf{w}'[1]/\mathbf{w}'[2]$ follows. Therefore, we have $slope(H) > slope(H')$. □

For each anchor tuple a, we use $l^+(a)$ and $l^-(a)$ to denote its upper and lower boundary lines, which have the largest and smallest slopes among $\mathcal{H}_a$, respectively. According to Lemma 3.2, the anchor-pair region is a convex region. Four corners of an anchor-pair region, namely *left*, *right*, *top* and *bottom* points, are denoted as l, r, t and b respectively when the context is clear (See Fig. 3.4).

The following lemma shows that if we traverse the intersection points of a preference line H with increasing order of the first dimension (i.e., from left to right), the slopes of the crossing lines will strictly decrease. We remark that this property is essential to the time complexity of our algorithm.

Lemma 3.3. *Given a preference line H, let H_1 and H_2 be two other preference lines intersecting with H at points p_1 and p_2, respectively. We have $slope(H_1) > slope(H_2)$ if $p_1[1] < p_2[1]$.*

Proof. Consider the example in Fig. 3.5 where we assume that $slope(H) > slope(H_1)$. According to the definition of preference hyperplane, we know that

there is no tuple in $\mathcal{P}$ lying below H or H_1, which means there is no tuple below curve $\overline{ap_1b}$. We now show that the following two cases do not hold. (i) $slope(H_1) < slope(H_2) < slope(H)$ (e.g., H_2' in Fig. 3.5). As shown in Fig. 3.5, H_2' is completely below curve $\overline{ap_1b}$, which means that there is no tuple in $\mathcal{P}$ lying on H_2'. According to the definition of preference hyperplane, we know that H_2' is not valid. (ii) $slope(H) < slope(H_2)$ (e.g., H_2'' in Fig. 3.5). Obviously, H_2'' is completely above $\overline{p_1a}$, which implies that there is at least one tuple in $\mathcal{P}$ (on $\overline{p_1a}$) below H_2'', which violates the definition of preference hyperplane. Therefore, H_2'' is not valid. From (i) and (ii), we have $slope(H_1) > slope(H_2)$. Similarly, we can verify the case where $slope(H) < slope(H_1)$. Thus, the lemma holds. □

We have the following corollary based on Lemma 3.2 and Lemma 3.3.

Corollary 3.1. *Given an anchor-pair region $R_{i,j}$ constructed from a_i and a_j, only preference lines from $\mathcal{H}_{a_i}$ and $\mathcal{H}_{a_j}$ cross the region $R_{i,j}$.*

Influence-based Pruning. We use $I^+(R)$ and $I^-(R)$ to denote the upper and lower influence bounds of a region R, respectively. We can safely prune a region if $I^+(R) < k$ or $I^-(R) > k$. Following theorem indicates that we can directly derive them based on the number of preference lines in each anchor tuple.

Theorem 3.2. *Given an anchor-pair region $R_{i,j}$, we have $I^+(R_{i,j}) = \sum_{i \le s \le j} |\mathcal{H}_{a_s}|$ and $I^-(R_{i,j}) = \sum_{i < s < j} |\mathcal{H}_{a_s}| + 2$ respectively.*

Proof. According to Lemma 3.2, for any point $p \in R_{i,j}$, p is above any preference line l_{out} from anchor tuple a_s with $s < i$ because $p[1] > a_i[1]$, $slope(l_{out}) > l^+(a_i)$, and p is above $l^+(a_i)$. Similarly, we know p is above any line l_{out} from anchor tuple a_s with $s > j$. Therefore, we have $I(p) \le \sum_{i \le s \le j} |\mathcal{H}_{a_s}|$ which means $I^+(R_{i,j}) = \sum_{i \le s \le j} |\mathcal{H}_{a_s}|$. Now let l_{in} be any preference line from anchor tuple a_s with $i < s < j$.

According to Lemma 3.2, we have $slope(l^-(a_i)) > slope(l_{in}) > slope(l^+(a_j))$. With Lemma 3.3, we know the intersecting point between l_{in} and $l^-(a_i)$ (resp. $l^+(a_j)$) must be on segment $\overline{a_i t}$ (resp. $\overline{t a_j}$), from which we know that l_{in} is above $R_{i,j}$. Therefore, $I(p) > \sum_{i<s<j} |\mathcal{H}_{a_s}|$. Since p must be on or lie below $l^-(a_i)$ as well as $l^+(a_j)$, we have $I(p) \geq \sum_{i<s<j} |\mathcal{H}_{a_s}| + 2$ which means $I^-(R_{i,j}) = \sum_{i<s<j} |\mathcal{H}_{a_s}| + 2$. Now we show the lower and upper bounds are tight. As shown in Fig. 3.4, $I(b)$ can achieve the upper bound which is $\sum_{i\leq s\leq j} |\mathcal{H}_{a_s}|$, while $I(t)$ can reach the lower bound which is $\sum_{i<s<j} |\mathcal{H}_{a_s}| + 2$. □

Cost-based Pruning. We use $f^+(R)$ and $f^-(R)$ to denote the upper and lower cost bounds of a region R, respectively. As shown in Fig. 3.4, $\overline{ltr}$ (resp. $\overline{lbr}$) represents the upper (resp. lower) boundary of $R_{i,j}$. It is immediate that for any point $p \in R_{i,j}$, there is a point p' on $\overline{ltr}$ (resp. $\overline{lbr}$) such that p dominates (resp. is dominated by) p'. With the same argument in Theorem 3.1, we can derive the cost lower (resp. upper) bound based on a few points obtained from $\overline{ltr}$ (resp. $\overline{lbr}$). Specifically, $f^-(R)$ is derived based on l, t, r as well as two possible tangent points on $\overline{lt}$ and $\overline{tr}$, while $f^+(R)$ relies only on three points l, b and r.

Traverse the Anchor-Pair Region. If an anchor-pair region $R_{i,j}$ cannot be pruned, we need to explore the preference lines among $\mathcal{H}_{a_i}$ and $\mathcal{H}_{a_j}$ to identify $lev_k(\mathcal{H})$ within $R_{i,j}$. Thanks to Corollary 3.1, we only need to consider the preference lines from $\mathcal{H}_{a_i}$ and $\mathcal{H}_{a_j}$. Below, we show how to quickly find the turning points of $lev_k(\mathcal{H})$ (i.e., candidate points) with one scan of these preference lines in the following two steps.

(*i*) *Find the entrance line.* Let n_i and n_j denote the size of $\mathcal{H}_{a_i}$ and $\mathcal{H}_{a_j}$ respectively. Clearly, $lev_k(\mathcal{H})$ must enter $R_{i,j}$ from one of the $n_i + n_j$ lines. To this end, we use an array L to sequentially keep these lines based on their encountering order along the boundary $\overrightarrow{tlb}$. As shown in Fig. 3.4, we have $L = \{H_5, \ldots, H_1, H_6, \ldots, H_9\}$.

Let $\Delta = k - (I^-(R_{i,j}) - 2)$. The entrance line is the Δ-th line in L. Note that we only need to check the tangent point on the two boundary lines if $k = 1$. Regarding the example in Fig. 3.4, the entrance line is H_2 if $I^-(R_{i,j}) = 2$ and $k = 4$. Similarly, it is H_8 when $k = 8$.

(*ii*) *Traverse.* Given the entrance line, we can sequentially visit the turning points following $lev_k(\mathcal{H})$ traverse algorithm in Section 3.2.4. By taking advantage of the fact that preference lines are well divided by anchor tuples (Lemma 3.2), we can immediately identify the next turning point in $O(1)$ time, which is $O(n)$ time for the general $lev_k(\mathcal{H})$ traverse problem. Particularly, let s_i and s_j denote the current subindex of two lines in $\mathcal{H}_{a_i}$ and $\mathcal{H}_{a_j}$, respectively, which contribute to the current turning point. We first assume the entrance line is from a_i where $s_i = n_i - \Delta + 1$, and hence $s_j = 1$ (i.e., the boundary line $l^+(a_j)$). According to Lemma 3.3, we can immediately identify the turning points by iteratively increasing s_i and s_j. As shown in Fig. 3.4, we have $s_i = 2$ and $s_j = 1$ at the beginning. Similarly, we have $s_i = 1$ and $s_j = \Delta - n_i$ if the entrance line is from a_j. The above traverse process terminates once the boundary line of a_i or a_j is reached (i.e., $s_i = n_i$ or $s_j = n_j$). Note that if the line leaving the region $R_{i,j}$ is from a_j, the anchor tuple a_j should also be considered as a candidate. Moreover, the tangent points of the visited lines are also considered according to Theorem 3.1.

In this way, we can enumerate the candidate points for an anchor-pair region in $O(n_i + n_j)$ time.

3.3.3 Algorithm

Intuitively, we may enumerate all possible pairs of anchor tuples to identify the k-critical point; that is, we may access $O(m^2)$ anchor-pair regions, where m is the number of anchor tuples in $\mathcal{A}$. By utilizing the following lemma, we show that we

only need to access at most $3m$ pairs in our exact algorithm.

Lemma 3.4. *Given an anchor-pair region $R_{i,j}$, we do not need to access an anchor-pair region $R_{u,v}$ with $i < u < v < j$ if one of the following two conditions holds:*
(i) $I^+(R_{i,j}) < k$;
(ii) $I^-(R_{i,j}) \leq k \leq I^+(R_{i,j})$

Proof. According to Theorem 3.2, for case (i), we have $I^+(R_{u,v}) = \sum_{u \leq s \leq v} |\mathcal{H}_{a_s}| < \sum_{i \leq s \leq j} |\mathcal{H}_{a_s}| = I^+(R_{i,j}) < k$. Therefore we can prune $R_{u,v}$. Similarly, for case (ii), we have $I^+(R_{u,v}) = \sum_{u \leq s \leq v} |\mathcal{H}_{a_s}| < \sum_{i < s < j} |\mathcal{H}_{a_s}| + 2 = I^-(R_{i,j}) \leq k$. Thus, we can prune $R_{u,v}$ as well. □

Algorithm 1 illustrates the details of our traverse based method. We use T to keep the best solution seen so far, and a priority queue Q is used to maintain the anchor-pair regions where the key is the cost lower bound (i.e., $f^-(R)$). Let i and j denote the subindexes of an anchor-pair region $R_{i,j}$ which are initialized as 1 and 2. Lines 2-10 illustrate how we enumerate the regions based on Lemma 3.4, together with the fact that we have $I^-(R_{i,j-1}) \leq k$ if $I^-(R_{i,j}) > k$ according to Lemma 3.4. Since either i or j will be increased by 1 in each iteration and j will be decreased by at most m times (Line 8), the number of iterations (i.e., possible regions) is bounded by $3m$, and there are at most $2m$ feasible regions to be explored. Then, we derive the k-critical point by traversing the regions survived the cost-based pruning (Line 14).

Complexity Analysis. We first show that the number of line segments (faces) of $lev_k(\mathcal{H})$ is bounded by $2n$ in our problem when $d = 2$.

Lemma 3.5. *The combinatorial complexity of $lev_k(\mathcal{H})$ is bounded by $2n$.*

Proof. We first show that any preference line $H \in \mathcal{H}$ appears at most twice on $lev_k(\mathcal{H})$ for $1 \leq k \leq n$. As $lev_k(\mathcal{H})$ consists of a set of consecutive segments, we

Algorithm 1: **TraverseBased-2d($\mathcal{A}$, $\mathcal{H}$, f, k)**

Input : $\mathcal{A}$: the anchor points of size m,
$\mathcal{H}$: preference lines of size n,
f : the cost function, k : target influence score

Output : T : k-critical point

```
1  Q := ∅; i := 1; j := 2;
2  while i < m do
3      while j ≤ m do
4          Compute cost and influence bounds for R_{i,j};
5          if I⁻(R_{i,j}) ≤ k ≤ I⁺(R_{i,j}) then
6              Q.enqueue(R_{i,j});
7          else if I⁻(R_{i,j}) > k then
8              j ← j − 1; break;
9          j ← j + 1;
10     i ← i + 1;
11 while !Q.isEmpty() do
12     R_{i,j} ← Q.dequeue();
13     if f⁻(R_{i,j}) < f(T) then
14         Traverse R_{i,j} and update T;
15 return T
```

can sequentially label these segments from left to right with 1, 2, ..., s if there are s segments in total. Now, we divide them into two sets, one with odd ids and the other with even ids. According to Lemma 3.3, we know that the slope of segments in the same set strictly decreases from left to right. If H appears more than twice on $lev_k(\mathcal{H})$, there would be at least two segments of H appearing in the same set which contradicts to the fact that the slope is strictly decreasing in the same set.

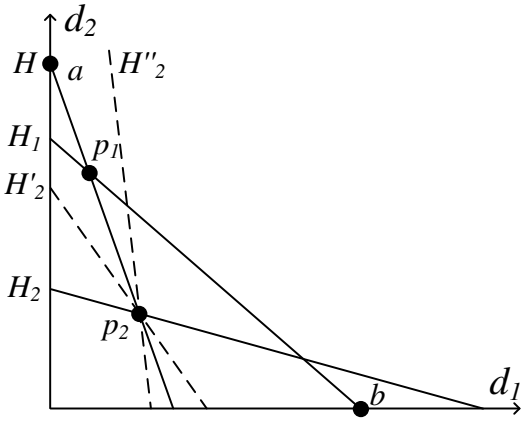


Figure 3.5: Line intersection order

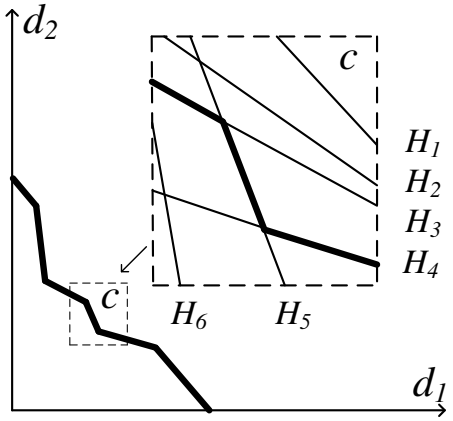


Figure 3.6: Space partition

Thus, H appears at most twice on $lev_k(\mathcal{H})$. Since there are n lines in total, we know that the number of 1-faces (i.e., edges) is bounded by $2n$. Clearly, the number of 0-faces (i.e., vertices) is also bounded by $2n$. Therefore, the proof is complete. □

As shown in the traverse algorithm for each individual anchor-pair region in Section 3.3.2, it takes $O(1)$ time to identify each turning point. Consequently, the traverse cost of Algorithm 1 is $O(n)$ since there are at most $2n$ line segments in $lev_k(\mathcal{H})$. Together with the facts that it takes $O(m)$ time to construct candidate anchor-pair regions, we have the following theorem since $m \leq n$.

Theorem 3.3. *The time complexity of Algorithm 1 is* $O(n)$.

3.3.4 Indexing

To deal with a large number of user preferences, we employ a $B+$ tree to organize the preference lines for each anchor tuple. Then each tree node can be treated as a *virtual anchor tuple* associated with a subset of consecutive preference lines. Consequently, an anchor-pair region $R_{i,j}$ can be decomposed into a set of *virtual anchor-pair regions* (depicted as dashed quadrilaterals in Fig. 3.4) based on four

virtual anchor tuples from a_i and a_j. Consequently, we can conduct influence-based and cost-based pruning for a pair of anchor tuples in a level by level fashion. Regarding each survived virtual region constructed from two *leaf nodes*, the traverse algorithm is the same as the anchor-pair region based one, where the lower and upper bounds of its influence score are carefully maintained during the tree traversal. Besides, we use an additional $B+$ tree to maintain the anchor tuples.

We can easily modify Algorithm 1 to accommodate the new data structures. Specifically, Line 4 will use the root node of each anchor tuple to construct anchor-pair regions. Moreover, if a_i or a_j at Line 14 is not a leaf node, we will expand their child nodes to construct new anchor-pair regions and apply the cost and influence based pruning. The survived regions will be pushed into Q for further consideration. In case both a_i and a_j are leaf nodes, we will traverse this region to identify k-critical point candidate.

Index Maintenance. Since the anchor tuple is one of the convex hull points in $\mathcal{P}$, we may continuously maintain them in $O(\log m)$ time in a 2-dimensional space [PS85]. By doing this, we can easily handle the update of user preference functions $\mathcal{W}$ and tuple set $\mathcal{P}$.

3.3.5 Discussion

The concept of anchor-pair region can be generalized to higher dimensions ($d \geq 3$) where every d anchor tuples form a region. However, we cannot derive tight cost and influence bounds because the preference hyperplanes are not well divided like 2-dimensional case. Moreover, as discussed in Section 3.2.4, the traverse based approach is not suitable to higher dimensional space. Consequently, we resort to the space-partition based approach for higher dimensions in Section 3.4.

3.4 Space Partition Based Method

In this section, we present the space partition based framework for efficiently computing the k-critical point in general dimensional spaces. First, we present an exact method which essentially computes the local k-level structure for each partitioned hypercube. To accelerate the computation, we then propose a randomized sampling method which can efficiently generate solutions with high accuracy.

3.4.1 Motivation of Space Partition

In this section, we briefly introduce the motivation of our space partition based method.

As discussed in Section 3.2, the main obstacle of our problem is the high combinatorial complexity $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ [CS89] of the $lev_k(\mathcal{H})$. Intuitively, we can alleviate this problem by partitioning the space into hypercubes such that the number of preference hyperplanes involved in each partition (i.e., hypercube) is significantly reduced. Moreover, we can exploit the *local dominance* relationship among the hyperplanes in a partition c to further reduce the number of hyperplanes involved, which is denoted by n_c. Here, we say hyperplane H_1 *locally dominates* hyperplane H_2 regarding c if any point on $H_1 \cap c$ is lying on or below H_2. Intuitively, by counting the number of dominated hyperplanes, we can estimate the influence score lower and upper bounds of a hyperplane within c, which can be utilized to prune a hyperplane directly for the given k value. On the other hand, a partition may be immediately pruned based on its influence score or cost lower and upper bounds. Moreover, we can compute the k_c-level against the hyperplanes within the partition where k is reduced to k_c by utilizing the influence score lower and upper bounds as well as the local dominance relationship. Then the randomized

incremental method in Section 3.2 can be applied on the n_c hyperplanes, denoted by $\mathcal{H}_c$, for the k_c-level computation. In practice, n_c and k_c are much smaller than n and k after a considerable number of space decompositions.

Example 3.6. *Consider the example in Fig. 3.6. Assume that* $n = 100$ *and* $k = 50$. *After a series of partitions, we reach partition* c *with influence score lower and upper bounds being* 46 *and* 52, *respectively. Thus, we know that initially* $n_c = 6$ *and* $k_c = 4$. *By using the local dominance relationship, we derive two hyperplane pruning rules. The first rule is essentially to prune hyperplanes that are totally below* k_c*-level in* c *(i.e.,* H_6 *in Fig. 3.6), while the other is employed to prune those completely above* k_c*-level (i.e.,* H_1 *and* H_2 *in Fig. 3.6). Note that as long as we prune a hyperplane above* k_c*-level, we have to reduce* k_c *by* 1. *After pruning these hyperplanes, we have that* $n_c = 3$ *and* $k_c = 2$ *and only* H_3, H_4, *and* H_5 *survived. We then use the randomized incremental method introduced in Section 3.2 to compute the* k_c*-level in* c.

3.4.2 Space Partition Based Pruning Techniques

In this section, we first introduce influence-based and cost-based pruning techniques for each partition. Then the local dominance technique is presented to reduce the number of hyperplanes involved.

Pruning A Partition. Given a partition c, we use $I^+(c)$ and $I^-(c)$ to denote the upper and lower influence bounds of c respectively. $f^+(c)$ and $f^-(c)$ denote the upper and lower cost bounds of c, respectively. Then, we can easily derive these bounds by its lower left corner and upper right corner, denoted by c_l and c_h respectively. For any point p in c, we have p dominates c_h and is dominated by c_l. Thus, $I^+(c) = I(c_l)$ and $I^- = I(c_h)$. Similarly, $f^+(c) = f(c_l)$ and $f^-(c) = f(c_h)$. Now, for a given partition c, we can use k as well as the current best solution cost

f_k to check if c can be pruned based on these bounds.

Exploiting Local Dominance. Given two preference hyperplanes H_1 and H_2 in a partition c, we say H_1 dominates H_2 if any point on H_1 within c is lying on or below H_2. As shown in Fig. 3.6, H_6 dominates all remaining hyperplanes except H_4, while H_1 is dominated by any of other hyperplanes. Using the local dominance relationship, we may exclude some hyperplanes from the computation. Below are two local dominance pruning rules.

(1) Dominance Pruning Rule (lower bound). Given a hyperplane $H \in \mathcal{H}_c$, if H dominates at least k_c hyperplanes in $\mathcal{H}_c$, then we can prune H safely.

For any point $p \in H \cap c$, we have $I(p) \geq I^-(c) + k_c$ as p is below or lying on at least k_c other hyperplanes. Therefore, H is below $lev_k(\mathcal{H})$ within c (i.e., k_c-level) and we can safely exclude H from the computation of c.

(2) Dominance Pruning Rule (upper bound). Given a hyperplane $H \in \mathcal{H}_c$, assume H dominates i hyperplanes in $\mathcal{H}_c$ and intersects with j hyperplanes in $\mathcal{H}_c$, if $i+j < k_c - 1$, then we can prune H safely and decrease k_c by 1.

For any point $p \in H \cap c$, we have $I(p) \leq I^-(c) + i + j + 1 < I^-(c) + k_c = k$ if $i + j < k_c - 1$; that is, H is always above $lev_k(\mathcal{H})$ within c. Consequently, we can simply decrease k_c by 1 and exclude H from further computation.

Example 3.7. *In Fig. 3.6, since H_6 dominates 4 hyperplanes and $k_c = 4$, we know H_6 can be pruned. Similarly, we can prune H_1 and H_2 using the second pruning rule where k_c is set to 2.*

Local dominance check. Given two hyperplanes H_1 and H_2 in a partition c, A denotes the intersection points from H_1 and c's 1-faces (i.e., edges). We say H_1 dominates H_2 regarding c if every point in A is below or lying on H_2. The time complexity is $O(2^{d-1}d)$.

3.4.3 Exact Algorithm

In this thesis, we adopt the linear Quad-tree to recursively partition the space. Algorithm 2 summarizes our space partition based exact approach. We maintain a priority queue Q to store the partitions that might contain the optimal solution where the key is the cost lower bound (i.e., $f^-(c)$). When a partition c is popped, we may prune c with the current best solution T (Line 4). It may be further split based on the depth and the number of hyperplanes in c (Line 5) where the influence lower and upper bounds of sub-partitions can be easily computed based on $\mathcal{H}_c$ and $I^-(c)$. Otherwise, we conduct the local dominance pruning rules to reduce the number of hyperplanes (Line 9) and then apply the randomized incremental algorithm in Section 3.2.4 to find candidate k-critical point.

Time Complexity Analysis. By n_c and k_c, we denote the average number of hyperplanes and level of the partitions. As shown in Section 3.2, the time complexity of computing a survived partition c is $O(n_c^{\lfloor d/2 \rfloor} k_c^{\lceil d/2 \rceil})$ in d-dimensional spaces, which is the dominant cost compared with the pruning techniques. Let α be the number of partitions survived from cost-based and influence-based pruning, the complexity of Algorithm 2 is $O(\alpha\ n_c^{\lfloor d/2 \rfloor} k_c^{\lceil d/2 \rceil})$.

Indexing. Instead of partitioning the space on the fly for each influence-based query, we can keep the Quad-tree in the disk where the related information of each partition (e.g., influence lower and upper bounds) is maintained. The hyperplanes are recorded in their corresponding leaf nodes where each hyperplane may appear in multiple leaf nodes. In practice we cannot afford to keep the whole Quad-tree as the depth required for a good performance is usually large and we support all possible k values. Consequently, in our implementation, we only materialize the higher level of the Quad-tree for a given space budget, and the nodes may be further split when necessary. For a new user preference function $\mathbf{w}$, we may immediately apply the

Algorithm 2: **PartitionBased-Exact**($\mathcal{H}$, f, k, N, D)

Input : $\mathcal{H}$: a set of hyperplanes, f : the cost function,
k : target influence,
N : leaf node capacity, D : the maximum depth

Output : T : k-critical point

```
1  T := ∅; Initialize Q with the domain of P;
2  while !Q.isEmpty() do
3      c ← Q.dequeue();
4      if f⁻(c) < f(T) then
5          if |H_c| > N and Depth(c) < D then
6              SplitPartition(c);
7              Push into Q sub-partitions which survive both influence and cost
               based pruning;
8          else
9              Apply local dominance based pruning on c;
10             Apply randomized incremental algorithm on c and update T;
11 return T
```

existing top-k technique to identify its top-1 result and insert its corresponding preference hyperplane to the Quad-tree. Regarding the arrival of a new tuple p, we need to update the related preference hyperplanes by issuing a reverse top-1 query. The Quad-tree will be updated accordingly. The deletion of user preferences and tuples can be handled in a similar way.

3.4.4 Sampling Based Solution

Motivation. The space partition method can significantly reduce the local computational cost for each individual partition c. Nevertheless, n_c may be still con-

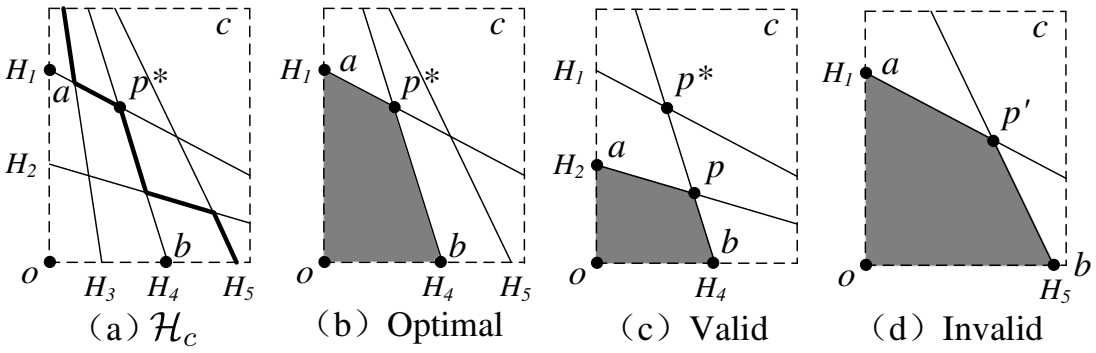


Figure 3.7: Sampling Approach

siderably large because a hyperplane may appear in multiple partitions and it is not cost-effective to have many partitions. Consequently, the cost of Algorithm 2 is still expensive when there are a large number of hyperplanes, due to the inherent high combinatorial complexity of $O(n_c^{\lfloor d/2 \rfloor} k_c^{\lceil d/2 \rceil})$ for the k_c-level in the partition c. This motivates us to devise sampling based approach for each partition such that, instead of k_c-level, we only need to maintain the *half-space intersection* of the sampled hyperplanes. The combinatorial complexity is reduced to $O(n_s^{\lfloor d/2 \rfloor})$ where n_s is the number of sampled hyperplanes in each round. In each round, we may come up with a candidate solution which satisfies the influence constraint even if we miss the optimal solution. Thus, as shown in our empirical study, we can quickly find candidate solutions with high accuracy after a considerable number of sampling rounds.

According to Theorem 3.1, the optimal solution is determined by at most d hyperplanes. Suppose p^* is the optimal solution in Fig. 3.7(a). We use $\mathcal{R}^*$ to represent the hyperplanes contributing to the optimal solution (i.e., H_1 and H_4 in Fig. 3.7(a)), and $\mathcal{B}$ to denote the hyperplanes that are strictly below p^* (i.e., H_2 and H_3 in Fig. 3.7(a)). By $\mathcal{H}_s$, we denote the sampled hyperplanes from $\mathcal{H}_c$ in each round. Then we say $\mathcal{H}_s$ is *optimal* if it contains (*i*) *all* hyperplanes in $\mathcal{R}^*$ and (*ii*) *none* of the hyperplanes in $\mathcal{B}$. Otherwise, we say $\mathcal{H}_s$ is *valid* if the final

solution is below or lying on k_c-level, and *invalid* if the final solution is above k_c-level. As we can see, the instances of $\mathcal{H}_s$ in Fig. 3.7(b), Fig. 3.7(c), and Fig. 3.7(d) are optimal, valid, and invalid, respectively. Note that we may come up with a candidate solution (e.g., p in Fig. 3.7(c)) in a valid $\mathcal{H}_s$.

Algorithm 3: **Sampling-Algorithm($\mathcal{H}_c$, ρ, n_r)**

Input : $\mathcal{H}_c$: a set of hyperplanes in partition c,
ρ : hyperplane sampling probability,
n_r : number of sampling rounds

Output : T : current best solution

```
1 for i ← 1 to n_r do
2     HI ← c; T' ← c_h;
3     for j ← 1 to n_c do
4         Sample H_j ∈ H_c with probability ρ;
5         Update HI and T' with H_j if sampled;
6     if I(T') ≥ k_c and f(T') < f(T) then
7         Update T by T';
8 return T
```

Sampling Algorithm. To devise the space partition based sampling method, we only need to replace Line 10 in Algorithm 2 with sampling approach (i.e., Algorithm 3). In the algorithm, we sample n_r rounds (Lines 1-7). In each round, we maintain a half-space intersection $\mathcal{HI}$ and round optimal point T', which are initialized as the partition c and its upper right corner c_h (Line 2), respectively. We pick a hyperplane $H_j \in \mathcal{H}_c$ with probability ρ (Line 4). If H_j is chosen, we update $\mathcal{HI}$ using the existing half-space intersection technique [CS88], as well as the candidate solution T' when necessary (Line 5). Note that we can terminate the current round early if $f(T') \geq f(T)$ since $f(T')$ is the cost lower bound of this

round due to half-space intersection shrink. After a sampling round, we update T by T' (Line 7) if (i) the local influence of T' is not smaller than k_c and (ii) the cost of T' is smaller than that of the current best solution T.

Time Complexity. By n_c and k_c, we denote the average number of hyperplanes and level of the partitions. The expected sampled size in each round is ρn_c. Since the time complexity of the half-space intersection problem is $O((\rho n_c)^{\lfloor d/2 \rfloor+1})$, it takes sampling approach $O(n_r(\rho n_c)^{\lfloor d/2 \rfloor+1})$ time to compute a partition where n_r is the number of sampling rounds.

Theoretical Analysis. Let $k_s = n_c - k_c$. We have $|\mathcal{B}| = k_s$, and $|\mathcal{R}^*| \leq d$, where $\mathcal{B}$ keeps the "bad" hyperplanes which are strictly below the optimal solution and $\mathcal{R}^*$ represents the hyperplanes contributing to the optimal solution p^*. Suppose each hyperplane is picked from $\mathcal{H}_c$ with probability $\rho = 1/k_s$, a set of sampled hyperplanes $\mathcal{H}_s$ is optimal in each round with probability θ, where

$$\theta = (\frac{1}{k_s})^{|\mathcal{R}^*|}(1 - \frac{1}{k_s})^{|\mathcal{B}|} \geq \frac{1}{k_s^d}(1 - \frac{1}{k_s})^{k_s} \geq \frac{1}{8k_s^d}.$$

Since each round of the sampling is independent to others, when $n_r = 8k_s^d \ln \frac{1}{\delta}$, we can identify the optimal solution with probability $1 - \delta$.

3.5 Experimental Study

In this section, we empirically evaluate the efficiency and effectiveness of the proposed techniques.

3.5.1 Experimental Setup

Algorithms. To the best of our knowledge, there is no existing work investigating influence based cost optimization problem. In this thesis, we implement and evaluate following algorithms.

Parameter	Values
Data dimensionality d	2, **3**, 4, 5
Tuple set $\mathcal{P}$	COLOR (C), **HOUSE (H)**
Preference function set $\mathcal{W}$	Uniform (UN), **Clustered (CL)**
Cardinality $\lvert\mathcal{W}\rvert$	10K, 0.2M, 0.4M, 0.6M, 0.8M, **1M**
Target influence score k	0.1, 0.3, **0.5**, 0.7, 0.9 ($\times\lvert\mathcal{W}\rvert$)
# of sampling rounds n_r	10, **100**, 1000, 10000
# of clusters for $\mathcal{W}$	10
Variance σ^2	0.05^2
cost function	$\mathbf{f_1}(\mathbf{p}) = \sum_{\mathbf{1\le i\le d}} \frac{\mathbf{1}}{\mathbf{p[i]}}$, [LJ12] $f_2(p) = \prod_{1\le i\le d} \frac{1}{p[i]}$, [GMC+15] $f_3(p) = \sum_{1\le i\le d}(p[i]-q[i])^2$, [GLC+15]

Table 3.2: Experimental Parameters

- **TRAVERSE.** The traverse based method proposed in Section 3.3, which works for 2-dimensional spaces.
- **PARTITION.** The space partition based exact method devised in Section 3.4.3.
- **EXACT.** We use EXACT to denote TRAVERSE (PARTITION) when $d = 2$ ($d \geq 3$).
- **SAMPLING.** The space partition based sampling method proposed in Section 3.4.4.

Note that the naive approaches proposed in Section 3.2.4 are very slow in practice. For instance, it takes more than one day to handle 800 preference functions in 3 dimensions. Thus, they are excluded from our performance evaluation.

Datasets. We use two real datasets as the given tuples $\mathcal{P}$. COLOR (C) (http://kdd.ics.uci.edu) consists of 68,036 records in 9-dimensional space, while HOUSE (H) (http://www.ipums.org) contains 127925 6-dimensional tuples. We

randomly select subspaces to generate tuples with lower dimensionality. Same as [VDKN10, VDNK10, VDNK13], we generate two kinds of synthetic datasets for user preference functions following two widely used data distributions, namely uniform (UN) and clustered (CL). For the uniform dataset, we randomly generate a normalized vector from a d-dimensional space. For the clustered dataset, we first randomly select c cluster centroids each of which is a preference function in d-dimensional space. Then, preference functions are generated around the centroids by following a normal distribution with variance σ^2 in each dimension. For ease of presentation, we use the combination of the abbreviated names. For example, by 2d C/UN we denote the dataset where 2-dimensional COLOR and uniform data are deployed as the tuple set and preference functions, respectively.

Parameters. We conduct experiments in different settings, including dimensionality d, the cardinality $|\mathcal{W}|$, the value of k, and the number of sampling rounds for SAMPLING (n_r). The experimental parameters together with default values (in bold) are shown in Table 3.2. Note that the space upper right corner is used as q in cost function f_3. During the computation of PARTITION, the leaf node capacity and depth of Quad-tree are set to 100 and 17, respectively. Note that we only materialize the Quad-tree with $100M$ space budget. In the experiment, we evaluate three metrics, namely running time, I/Os, and accuracy.

All algorithms are implemented in standard C++ with STL library support and compiled with GNU GCC. Our experiments are conducted on PCs with Intel Xeon 3.4GHz CPU and 32GB RAM running Debian Linux. The block size is set to 4KB for both $B+$ tree and Quad-tree.

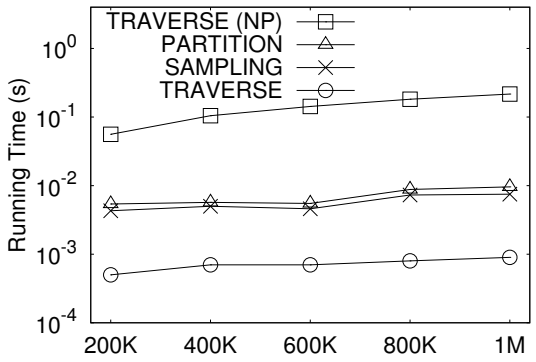


Figure 3.8: Comparing methods ($2d$)

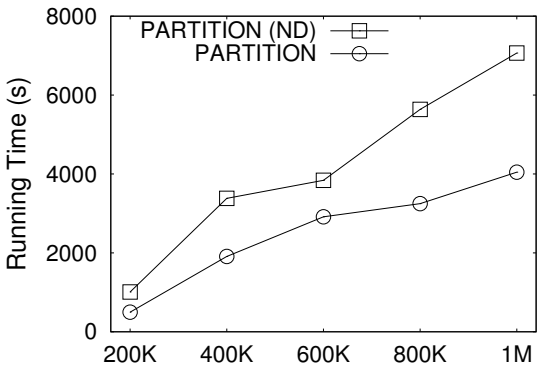


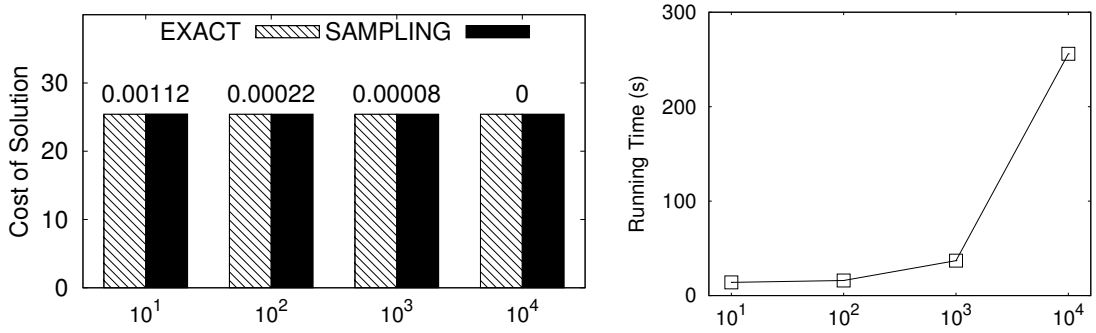
Figure 3.9: Evaluating pruning rule

3.5.2 Performance tuning

Comparison of 2-d algorithms. We start the experiments by investigating four methods in 2-dimensional spaces, namely TRAVERSE (NP), TRAVERSE, PARTITION, and SAMPLING, where TRAVERSE (NP) stands for TRAVERSE algorithm without cost based pruning techniques. It is worth mentioning that the time complexity of TRAVERSE (NP) here is $O(n)$ according to the proof of Theorem 3.3 since we can locate the next intersection point in $O(1)$ time after sorting preference lines. Fig. 3.8 shows that TRAVERSE significantly outperforms TRAVERSE (NP), which justifies the effectiveness of our anchor-pair region based pruning techniques. Meanwhile, TRAVERSE demonstrates superior performance (by an order of magnitude) compared with space partition based methods (i.e., PARTITION and SAMPLING) because the former can utilize the nice geometric properties of our problem in 2-dimensional spaces. On the other hand, SAMPLING is slightly better than PARTITION. In the following experiments, we use TRAVERSE as the default EXACT algorithm in 2-dimensions.

Evaluation of local dominance pruning rule. We evaluate the effectiveness of the local dominance rule by reporting the running time of two algorithms PARTITION and PARTITION (ND) against the growth of the user preference functions,

where PARTITION (ND) means the PARTITION algorithm without local dominance technique. It is shown in Fig. 3.9 that the local dominance technique can save nearly half of the computation cost. Note that we do not report the effectiveness of cost and influenced based pruning techniques for PARTITION, because we cannot handle even small size dataset (say, $n = 1000$ and $d = 3$) without their support.

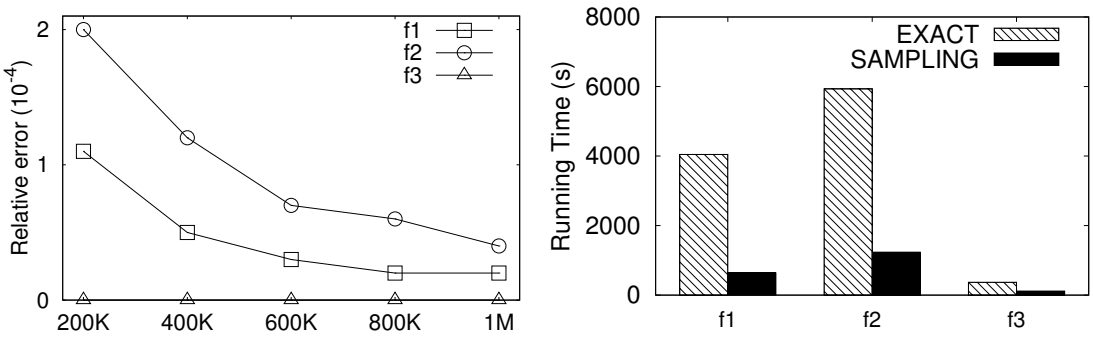


(a) Accuracy (b) Running Time

Figure 3.10: Tuning # of sampling rounds n_r

Effect of sampling rounds. Fig. 3.10 evaluates the effect of number of sampling rounds on the solution cost and running time with $10,000$ preference functions. Not surprisingly, Fig. 3.10(a) shows that we can increase the accuracy by increasing n_r where the relative error between EXACT and SAMPLING are labeled on top of their bar pairs. Meanwhile, the running time grows with the number of sampling rounds (Fig. 3.10(b)). Although it takes $10,000$ sampling rounds (around 250 seconds) to achieve the optimal solution, we already achieve a relative error of 0.0002 with 37 seconds when $n_r = 100$, which is sufficient in practice. Consequently, we take 100 sampling rounds in the following experiments.

Effect of cost function. We evaluate the effect of cost function in Fig. 3.11. Particularly, Fig. 3.11(a) reports the accuracy achieved by SAMPLING on three different cost functions as shown in Table 3.2, where the number of preference



(a) Accuracy vs $|\mathcal{W}|$ (b) Running Time

Figure 3.11: Evaluating effect of cost function

functions ($|\mathcal{W}|$) grows from 200K to 1M. It is reported that SAMPLING achieves very high accuracy (less than around 0.0002 relative error under all settings) on three functions, especially f_3. Fig. 3.11(b) illustrates the running time of EXACT and SAMPLING on three cost functions. It is interesting that two algorithms also achieve the best performance on f_3. This is because f_3 is simply defined as the distance between the searching point and a fixed point q. As a result, our methods are easy to approach the optimal solution.

3.5.3 Evaluating Accuracy

We empirically evaluate the accuracy of SAMPLING algorithm on different settings. We report the costs of EXACT and SAMPLING algorithms, as well as their relative error on the top of their bar pairs, against different dataset (Fig. 3.12), the growth of dimensionality (Fig. 3.13), the number of user preference functions (Fig. 3.14), and the k values (Fig. 3.15). It is observed that the cost of the solution from SAMPLING is very close to that of EXACT under all settings (say, less than 0.001 relative error). Sometimes, SAMPLING even successfully finds the optimal solution. Note that we evaluate the effect of growth of dimensionality with $10,000$ user preferences such that EXACT can finish when $d = 4$.

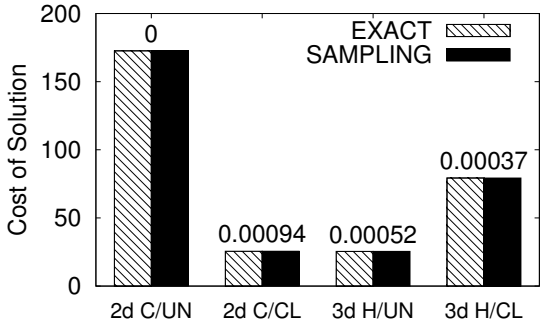


Figure 3.12: Accuracy vs datasets

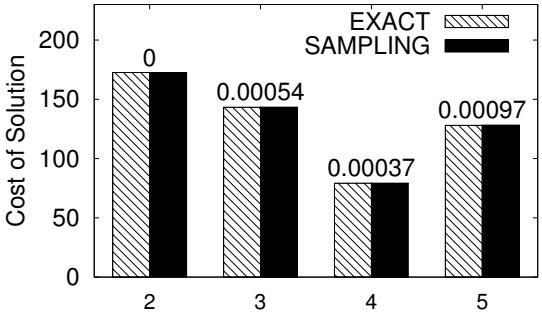


Figure 3.13: Accuracy vs d

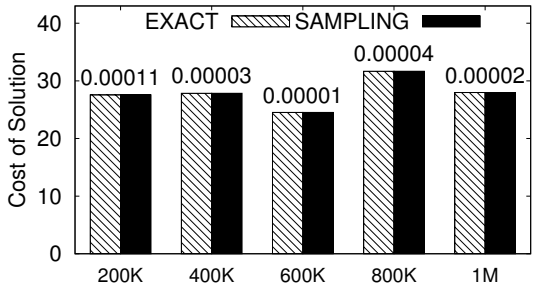


Figure 3.14: Accuracy vs $|\mathcal{W}|$

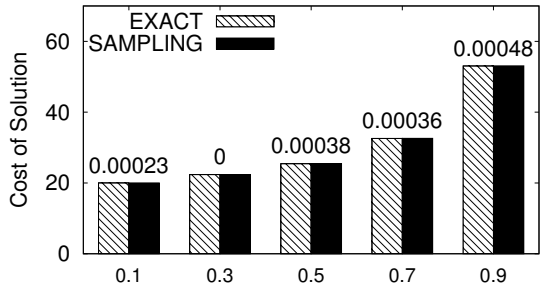
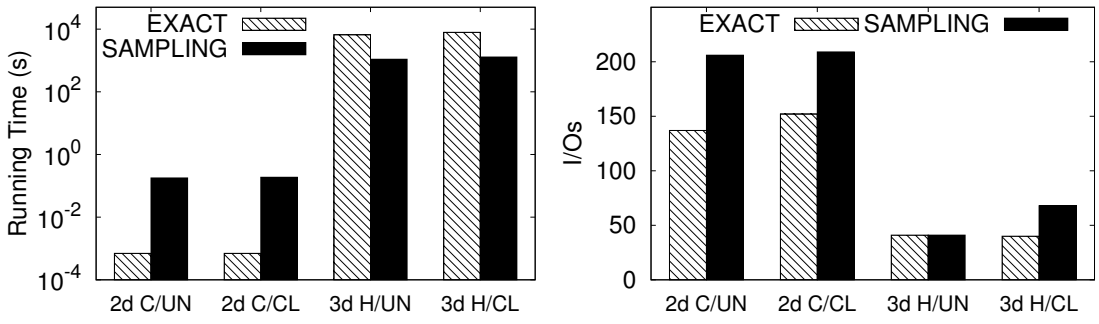


Figure 3.15: Accuracy vs k

3.5.4 Evaluating Efficiency

In this subsection, we evaluate the running time and I/O efficiency of the proposed algorithms under different experiment settings.

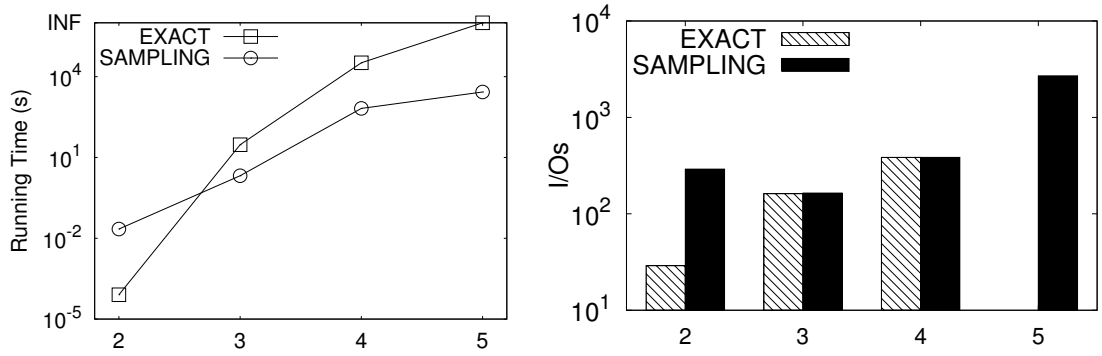


(a) Running time (b) I/Os

Figure 3.16: Performance against different datasets

Evaluation on different datasets. The experiment results against different datasets are reported in Fig. 3.16. It is very interesting that EXACT significantly

outperforms SAMPLING on two 2-dimensional datasets, with more than 2 orders of magnitude faster. This is because EXACT (i.e., TRAVERSE in 2-dimensional spaces) can utilize the nice geometry properties of 2-dimensional data and hence its time complexity is linear to the number of user preference functions. Nevertheless, SAMPLING beats EXACT (i.e., PARTITION for $d \geq 3$) by a large margin on 3-dimensional data because it is inherently difficult to find optimal solution in higher dimensional space due to the nature of the high combinatorial complexity of our problem. We also report the I/O costs of two algorithms on four datasets in Fig. 3.16(b). It demonstrates that the I/O cost of SAMPLING is always larger than that of EXACT, especially in 2-dimensional space. This is because EXACT can quickly approach the optimal solutions at the beginning stage and hence their cost based techniques can save more I/O cost.



(a) Running time (b) I/Os

Figure 3.17: Performance vs dimensionality d

Effect of dimensionality. The experiment results of effect of dimensionality are presented in Fig. 3.17. Fig. 3.17(a) reports that EXACT is more than 2 orders of magnitude faster than its counterpart in 2-dimensional space. However, as d increases, SAMPLING outperforms its counterpart. For instance, it runs faster than EXACT from 1 order of magnitude in 3-dimensions to 2-3 orders of magnitude in 4-dimensional spaces. As we can see that EXACT fails to return the result when

$d = 5$ (we set allowed running time to be one day), while SAMPLING finishes in less than one hour. Same as the observation in Fig. 3.16(b), Fig. 3.17(b) shows that SAMPLING consumes more I/O cost than EXACT.

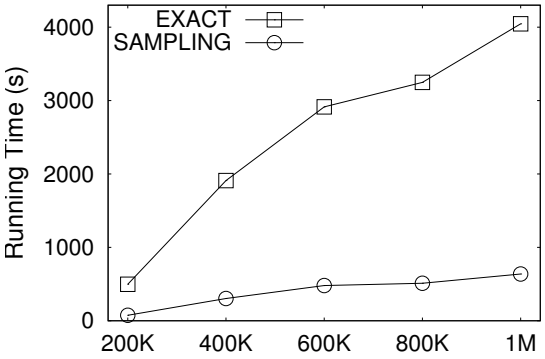


Figure 3.18: Performance vs $|\mathcal{W}|$

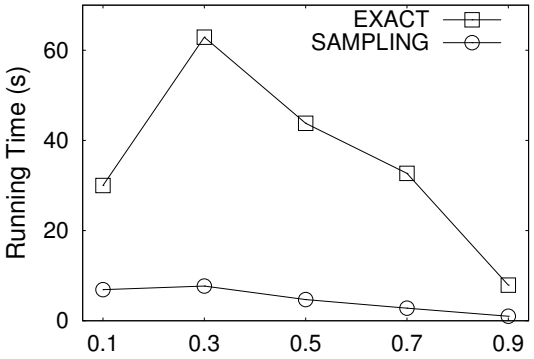


Figure 3.19: Performance vs k

Effect of cardinality. Fig. 3.18 reports the running time of two algorithms where the number of user preference functions ($|\mathcal{W}|$) grows from 200K to 1M. It shows that the margin becomes more significant when $|\mathcal{W}|$ increases, which implies that SAMPLING is much more scalable to $|\mathcal{W}|$ compared with EXACT in 3-dimensional spaces.

Effect of target influence score. Fig. 3.19 reports the running time with respect to different percentage values (i.e., k values) which increase from 0.1 (i.e., $k = 0.1 \times |\mathcal{W}|$) to 0.9 (i.e., $k = 0.9 \times |\mathcal{W}|$). It is observed that the highest running time of two algorithms comes from 0.3, which implies that the combinatorial complexity of k-level peaks when k is around $0.3 \times |\mathcal{W}|$. Moreover, the performance of SAMPLING is less sensitive to k compared to EXACT.

3.6 Conclusion

The investigation of e-business and user preference data has been of great interest recently. In this chapter, we advocate the problem of influence-based cost optimization problem, which aims to find a cost optimal position for a new product such that it can occupy a required amount of market share. To deal with the high combinatorial complexity nature of this problem, we develop efficient pruning and query processing techniques to significantly improve the performance of our algorithms. Specifically, an efficient traverse-based 2-dimensional algorithm and two space partition based algorithms for general multi-dimensional spaces are proposed. Extensive experiments demonstrate that our techniques achieve high efficiency and effectiveness performance.

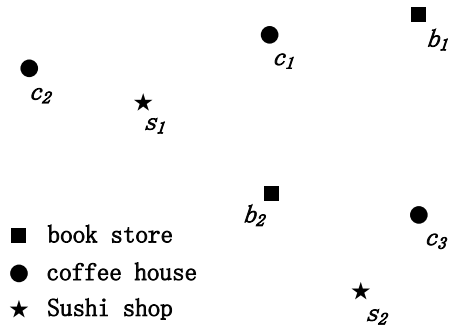
Chapter 4

Categorical Top-k Spatial Influence Query

4.1 Overview

Finding influential objects has been widely studied as an important spatial operator ever since it was introduced in [KM00] due to a wide spectrum of applications such as decision support, profile-based marketing, resource allocation, etc. Existing techniques define the influence of a facility as the number of users that consider it as nearest neighbors, namely, the bichromatic reverse nearest neighbor query. Informally, given a set $\mathcal{F}$ of facilities (e.g., gas station, supermarket) and a set $\mathcal{O}$ of users (e.g., persons, cars), the influence of a facility f can be defined by the number of users whose nearest neighbors are f. However, in practice several types of facilities exist and play different roles in satisfying uses' needs. Therefore, a more sophisticated way to evaluate the influence of facilities is preferable. Instead of considering all facilities as the same type (e.g., finding the most influential supermarket among all supermarkets) as mentioned above, in this thesis, we propose a

novel definition to evaluate the influence of a facility f by its capability of forming a *functional unit* together with facilities of other categories.



Id	OFUPs	Inf
b_1	c_1, s_1	0
b_2	c_3, s_2	5
c_1	b_2, s_1	2
c_2	b_2, s_1	0
c_3	b_2, s_2	2
s_1	b_2, c_1	3
s_2	b_2, c_3	2

Figure 4.1: Motivating Example

Example 4.1 (**Motivating Example**). *Consider the CBD area in Sydney. There are various types of business/facilities such as restaurants, cafes, bookshops, supermarkets, etc, as shown in Fig. 4.1. A customer may want to spend a leisure afternoon by drinking coffee, reading/purchasing books, and enjoying some sushi. Here, facilities of types* book store*,* cafe*, and* Sushi shop *form a functional unit for this customer where one functional unit consists exactly one facility from each category of desired services/facilities from the user. In Fig. 4.1, facilities* $\{b_1, c_1, s_1\}$ *form a functional unit. One facility may be involved in many functional units. For instance,* b_1 *forms functional units* $\{b_1, c_1, s_1\}$*,* $\{b_1, c_2, s_2\}$*,* $\{b_1, c_3, s_2\}$*, etc. If we consider the cost to form a functional unit to be the sum of pairwise distance of its facilities, clearly the optimal functional unit (OFU) is the one with minimum cost*[1]*. In Fig. 4.1,* $\{b_1, c_1, s_1\}$ *is the optimal functional unit for* b_1*.* c_1 *and* s_1 *are called* participants *of* b_1*'s optimal functional unit (OFUPs).*

After computing the optimal functional unit for each of the facilities, in this paper we define the influence of a facility f *as the total number of optimal functional*

[1]Ties break arbitrarily. We will study other cost models such as maximum pairwise distance in future works.

units that f participates in. As in Fig. 4.1, b_1's influence value is 0 since it does not participate in any other facility's OFU. b_2's influence value is 5 since it participates in the OFUs of c_1, c_2, c_3, s_1 and s_2. Such an influence value clearly represents the importance of one facility in forming optimal functional units with other types of facilities and thus offers a new angle to analyze the potential of one facility in business strategies, urban planing, etc. In an extreme case, if a book store is the only one of this type and is surrounded by many other types of facilities (thus, could attract customers of various needs), the influence value of this book store is high representing huge business potential.

Challenges. In this chapter, we study the problem of categorical top-k spatial influential query as introduced above. A straightforward approach is to compute the optimal functional unit for each object and then sum up the number of optimal functional units that each facility participates in. However, this approach suffers from two drawbacks. Firstly, as we shall show in Section 4.3.1, finding the optimal functional unit for a facility f is an NP-hard problem. Secondly, the size of spatial facilities is usually massive involving a large number of categories. For instance, in the experiment, the real dataset we utilize consists of more than 1.4 million facilities with 73 different categories.

To address the above challenges, it is critical to devise efficient spatial indexing and query processing techniques to support top-k spatial influence query against a massive number of spatial objects. In this paper, we follow the filtering-and-refinement framework based on R-trees style spatial indexes. Efficient and effective pruning techniques are developed to avoid the costly verification as much as possible. Specifically, the framework consists of two main steps, namely possible participants finding and optimal feasible set computation. The major challenge in the first step is to derive a tight pruning distance. We find that the pruning

distances derived by common strategies are substantially larger than the optimal pruning distance. Motivated by this, we propose a nearest neighbor set (NNS) based approach which leads to a performance acceleration by several orders of magnitude. In the second step, we develop two algorithms: one is an efficient exact algorithm and the other is an approximate algorithm with performance guarantee.

Roadmap. The rest of this chapter is organized as follows. Section 4.2 presents problem definition and background knowledge. Section 4.3 formulates the intractability of the problem and presents an efficient framework. Two key steps of our techniques are introduced in Sections 4.4 and 4.5, respectively. Section 4.6 gives the empirical study and Section 4.7 concludes the paper.

4.2 Background

We present problem definition and necessary preliminaries in this section. Table 4.1 summarizes the notations frequently used throughout this chapter.

Notation	Definition
o ($\mathcal{O}$)	object (a set of objects)
$o.\lambda$	the category of object o
$\mathcal{O}_i$	all objects in $\mathcal{O}$ with category i
$q.\psi$	a set of categories specified by query q
$\mathcal{O}_q$	all objects in $\mathcal{O}$ relevant to query q
S (S_o)	functional unit under a specific query (optimal functional unit)
$d(o_1, o_2)$	the Euclidean distance between objects o_1 and o_2

Table 4.1: The summary of notations

4.2.1 Problem Statement

A facility object $o \in \mathcal{O}$ is a point in a d-dimensional numerical space with a particular category, where $\mathcal{O}$ is a set of such facilities. Hereafter, we use object

and facility interchangeably when there is no ambiguity. In this thesis, we focus on 2-dimensional spatial space. Suppose there are totally t categories of facilities, we use $\mathcal{O}_i$ to denote all objects in $\mathcal{O}$ with category i $(1 \leq i \leq t)$. Given two objects o_1 and o_2, $d(o_1, o_2)$ denotes the Euclidean distance between them. A query q consists of a set of desired facility categories $\{t_1, t_2, ..., t_m\}$ denoted by $q.\psi$. Given a query q, a set S of objects is called a functional unit if it contains exactly one object from each set $\mathcal{O}_{t_i}, 1 \leq i \leq m$. A functional unit S_o is called optimal if the cost of S_o is minimized among all possible candidate functional units. We define the facility influence as follows.

Definition 4.1. ***(Facility Influence)*** *Given a query q with facility categories $q.\psi = \{t_1, t_2, ..., t_m\}$, for each object $o \in \mathcal{O}_{t_i}$ $(1 \leq i \leq m)$, let S_o denote the optimal functional unit containing o regarding q. If an object o' participates in S_o, then the influence of o' is increased by 1. The facility influence of an object is defined as the final value of its influence after considering all objects in $\mathcal{O}_{t_1} \cup ... \cup \mathcal{O}_{t_m}$.*

To define the optimal functional unit, in this paper we consider the cost function as the *sum of pairwise distance*. This is because the sum of pairwise distance denotes the expected cost of a route that visits all objects once in the functional unit, and a functional unit with a small sum of pairwise distance cost has a small expected route cost. Other cost functions, such as the maximum distance among all pairwise distances, will be studied in future works.

Definition 4.2. ***(Sum of Pairwise Distance Cost)*** *Given a set S of objects, the sum of pairwise distance cost of S, denoted by $cost_{Sum}(S)$, is equal to the sum of pairwise distance of objects in S. That is,*

$$cost_{Sum}(S) = \sum_{o_i, o_j \in S, i<j} d(o_i, o_j) \tag{4.1}$$

Problem Statement Given set of facilities $\mathcal{O}$, a query q with facility types $q.\psi = \{t_1, t_2, ..., t_m\}$ an integer k, the categorical top-k *Spatial Influence Query* (SInQuery) problem is to find the k facilities with maximum facility influence values in $\mathcal{O}_{t_1}$.

Example 4.2. *Consider the example in Figure 4.1. Given $q.\psi = \{b, c, s\}$ and $k = 1$, we aim to find the most influential object in $\mathcal{O}_b$. To this end, for each object not belonging to type b, we compute its optimal functional unit. Taking c_1 for example, we find that its optimal functional unit is $\{c_1, b_2, s_1\}$. We thus increase the influence of b_2 by 1. After considering the remaining objects similarly, we have that the influence of b_2 is 5 while that of b_1 is 0. Hence, b_2 is the most influential object.*

In the problem statement, we assume that a user is only concerned about comparing the importance of objects in the same category, which is meaningful. For instance, a user maybe ask, among all coffee houses, which one is most investment potential. Without loss of generality, we assume that the first category t_1 is the concerning one.

4.2.2 R-tree Distance Metric Based Cost

In this section, we introduce several R-tree distance metric based functional unit cost computation methods.

Given a query $q = (\{t_1, t_2, ..., t_m\})$ and a set $N = \{N_1, N_2, ..., N_m\}$ of R-tree nodes. Assume that N_i only contains objects of type t_i for $1 \leq i \leq m$. Without loss of generality, we now are interested in finding a cost bound C for N_1 such that any object in N_1 has a functional unit with cost no larger than C.

For ease of illustration, let's consider the example in Figure 4.2 where we simply assume that $q = (\{R, S, T\})$ and $N = \{R, S, T\}$. To get the cost bound of R, a

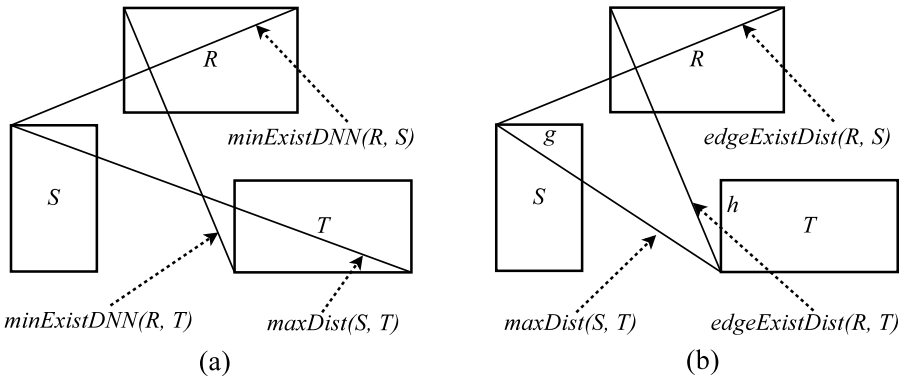


Figure 4.2: Cost construction for index node

straightforward way is to use the *maxDist* metric. That is, $C = maxDist(R, S) + maxDist(R, T) + maxDist(S, T)$. Obviously, any object in R can find an object in S within distance $maxDist(R, S)$, and an object in T within $maxDist(R, T)$. Meanwhile, the distance between any pair of objects in S and T is smaller than $maxDist(S, T)$. Thus, C is indeed a cost bound for R.

However, the *maxDist* based cost bound is not tight enough. In the following, we give two more R-tree distance metric based cost computation methods, which can always achieve smaller cost than the *maxDist* based method.

MinExistDNN based cost. Recently, a new pruning distance metric, called *minExistDNN*, has been proposed to resolve the nearest influential site problem [XZKD05], which has been generalized to arbitrary dimensions [CP07]. The definition of $minExistDNN(R, S)$ is as follows:

$$minExistDNN(R, S) = \max\{minMaxDist(r, S) | \forall \text{object } r \in R\} \tag{4.2}$$

Given an object $r \in R$, we know that r can always find a neighbor in S within distance $minMaxDist(r, S)$ [RKV95]. It is also easy to verify that $minMaxDist(r, S) < maxDist(R, S)$. According to Equation 4.2, we have the following conclusion. Metric *minExistDNN(R, S)* guarantees that, within this dis-

tance, any object in R can find a neighbor object in S and $minMaxDNN(R, S) < maxDist(R, S)$. With this property, we derive a *minExistDNN* based cost, which basically consists of two parts, the *minExistDNN* from the anchor node to other nodes and the *maxDist* between each pair of other nodes. As shown in Figure 4.2(a), the *minExistDNN* based cost of node R is $minExistDNN(R, S) + minExistDNN(R, T) + maxDist(S, T)$. Clearly, this cost is always smaller than the above *maxDist* based cost.

EdgeExistDist based cost. According to the definition of MBR, we know that there exists at least one object on any edge of an MBR. From this property, we immediately develop another method to compute the cost, which is called *edgeExistDist* based cost, as shown in Figure 4.2(b). The main idea of this method is to pick an edge on each of the other nodes such that the cost is optimal, which are edges g and h in this example. Here, $edgeExistDist(R, S)$ is the maximum distance of R to the selected edge g in S and $edgeDist(S, T)$ is the maximum distance between edges g and h selected from S and T. The final *edgeExistDist* based cost of R is $edgeExistDist(R, S) + edgeExistDist(R, T) + edgeDist(S, T)$.

Since $minMaxDist(R, S)$ is essentially to find an optimal edge of S for objects in R, we have the intuition that $edgeExistDist(R, S)$ is equal to $minExistDNN(R, S)$ in most cases if we select edges well. Further, we deduce that the *edgeExistDist* based method can achieve smaller cost than the *minExistDNN* based method because the second part $edgeDist(S, T)$ is always smaller or equal to $maxDist(S, T)$. Specially, the advantage goes more obviously when the number of query types increases. We conducted experiments using these two methods. The results show that the *edgeExistDist* based method outperforms the other.

It is worth mentioning that there are 4^{k-1} combinations of edge selections in

total for the *edgeExistDist* based method if the query number is k. To avoid this exhaustive search, we use a heuristic approach to solve this problem. For each other nodes (i.e., S and T), we select the edge which is nearest to the anchor node. By nearest, we mean that the maximum distance between any point in the edge and any point in the anchor node is minimum. As shown in Figure 4.2, edges g and h are the nearest edges selected from S and T respectively. Apparently, the time complexity of this heuristic method is linear to k.

4.3 Framework

In this section, we first give the intractability of the *SInQuery* problem and then present a 2-step framework to solve the problem efficiently.

4.3.1 Problem Intractability

Lemma 4.1. *The SInQuery problem is NP-hard.*

Proof. A procedure of *SInQuery*, which computes the optimal functional unit for an object, can be formally described as follows. Given a query q with object categories $\{t_1, t_2, ..., t_m\}$, we aim at finding a set S of objects containing exactly one object from each set $\mathcal{O}_{t_i}, 1 \leq i \leq m$ and $cost_{Sum}(S)$ is minimized. We first give the corresponding decision problem as follows. Given a positive value η and a query q where $q.\psi = \{t_1, t_2, ..., t_m\}$, the problem is to determine whether there exists a set S of objects in $\mathcal{O}_q$ such that S contains exactly one object from each set $\mathcal{O}_{t_i}, 1 \leq i \leq m$ and $cost_{Sum}(S) \leq \eta$. For simplicity, we denote the decision problem by *SInQuery*.

We then prove the NP-hardness of *SInQuery* by a reduction from the well-known NP-C problem 3-SAT [GJ79], which is described as follows. Let U be a set of literals $\{u_1, \overline{u_1}, ..., u_m, \overline{u_m}\}$ where $\overline{u_i}$ is the negation of u_i. Given an expression

$E = C_1 \wedge C_2 \wedge ... \wedge C_m$, where each clause $C_i = \{x_i \vee y_i \vee z_i\}$ and $x_i, y_i, z_i \in U$ for $1 \leq i \leq m$, it determines whether there exists a truth assignment for u_i for $1 \leq i \leq m$ such that E is true.

Given an instance of the 3-SAT problem, we construct an *SInQuery* problem instance as follows. For each clause C_i in E, we create 3 objects, each for its 3 literals x_i, y_i, z_i, and let their category be t_i. For any two objects o_i and o_j, we assume the corresponding literals are u_i and u_j. We then assign the distance $d(o_i, o_j)$ of o_i and o_j according to the following rules. If u_i and u_j are in the same clause, we let $d(o_i, o_j)$ be any positive value as long as the triangle inequality holds. Otherwise, if u_i is the negation of u_j, that is $u_i = \overline{u_j}$, we let $d(o_i, o_j) = 2$. Otherwise, $d(o_i, o_j) = 1$. We set η to be $\frac{m(m+1)}{2}$. Clearly, the above construction process could be finished in polynomial time.

We now show that the above constructed *SInQuery* problem instance is equivalent to its corresponding 3-SAT problem instance. Assume that the answer to 3-SAT is "yes", that is, there exists a truth assignment A for the literals in U such that E is true. Then, we construct S of $\mathcal{O}_q$ as follows. For each clause C_i, we add in S the object o_i corresponding to the literal u_i with true value. Clearly, S is of size m and contains exactly one object from each set $\mathcal{O}_{t_i}, 1 \leq i \leq m$. For any two objects o_i and o_j of S, we know that $d(o_i, o_j)$ is either 1 or 2. We claim that $d(o_i, o_j)$ cannot be 2 since, otherwise, the corresponding literals u_i and u_j are mutual negative, which can be denied by the fact that both u_i and u_j are true. Therefore, $cost_{Sum}(S) = \frac{m(m+1)}{2} \leq \eta$. Thus, the *SInQuery* problem is "yes".

Consider the other direction. Assume that the answer to *SInQuery* is "yes", that is, there exists a set S of objects such that S contains exactly one object from each set $\mathcal{O}_{t_i}, 1 \leq i \leq m$ and $cost_{Sum}(S) \leq \eta$. Then, we construct a truth assignment A for the literals U as follows. For each object $o_i \in S$, we let the corresponding literal

u_i be true and consequently $\overline{u_i}$ is false. For the remaining literals, we arbitrarily assign their truth value with the constraint that u_i and $\overline{u_i}$ have different truth values. First, we show A is a valid assignment, that is, there exist no pairs of two literals u_i and $\overline{u_1}$ both being true. As $cost_{Sum}(S) \leq \eta = \frac{m(m+1)}{2}$, we know that $d(o_i, o_j) = 1$ for any two objects o_i and o_j in S. Thus, any literal u_i covered by S satisfies the constraint that u_i and $\overline{u_i}$ have different truth values. For the remaining literals that are not covered by S, we know that they meet the constraint as well due to the way we construct A. Second, we show E is true under A. Since we add into S one object from each set $\mathcal{O}_{t_i}, 1 \leq i \leq m$ and let the corresponding literal be true, we know that the clause C_i is true for $1 \leq i \leq m$. Therefore, E is true and thus the answer to the 3-SAT problem is "yes".

Therefore, the proof is complete. □

4.3.2 Overall Framework

Algorithm 4 illustrates an outline of the framework which follows two steps, *possible participant set finding* and *optimal functional unit computation*. Both our exact and approximate algorithms follow this framework.

Algorithm 4: **TopInfluentialObjects**

1 Step 1: *Possible Participant Set Finding.* As long as unable to determine the final result, we select an unprocessed object $o \in \mathcal{O}_{t_1 \cup \ldots \cup t_m} \setminus \mathcal{O}_{t_1}$ to find a set of candidate objects which would become a member of its optimal functional unit.

2 Step 2: *Optimal Functional Unit Computation.* As long as unable to determine the final result, we select an unprocessed object $o \in \mathcal{O}_{t_1 \cup \ldots \cup t_m} \setminus \mathcal{O}_{t_1}$ to compute its optimal functional unit using the possible participant set.

3 Output: k most influential objects in $\mathcal{O}_{t_1}$.

Data Structure. We adopt R-tree to index our dataset; that is the set of all facility objects. Since the dataset contains facilities of different categories, naturally there are two ways of indexing, multiple trees (i.e., one R-tree for each facility category) and a single R-tree (i.e., use one R-tree to index all facility objects of different categories). Considering the fact that the number of total categories would be large and the number of categories specified in a query is usually small (e.g., in our experiments a real dataset contains 73 categories and the query usually specifies less than 15 categories), we choose to index the facility dataset using multiple trees, one for each category of facilities in order to minimize node accesses during query processing. Specifically, given a dataset, we index each category by an R-tree and store the built trees on disk. When a query is invoked, we simply pick and load the relevant R-trees to memory for computation. The experimental results show that the use of single tree can achieve better performance than multiple trees in terms of running time when the total number of categories is small (e.g., no more than 5 categories). However, in all other cases, the multiple trees strategy is better than the other.

Algorithm Implementation. In order to terminate the process of finding the top-k most influential objects as early as possible, we develop a top-down tree traversal method in Algorithm 4. The main idea can be described as below. We maintain a priority queue Q_e, a mutable priority queue Q_r, and a common first in first out queue Q_c.

- $Q_e = \{e | e$ is a visited but not expanded index entry of all relevant R-trees $\}$
- $Q_r = \{e | e$ is a visited R-tree entry of category $t_1\}$.
- $Q_c = \{e | e$ is a visited object entry of all relevant R-trees except category $t_1\}$

We discuss the sorting of Q_e in Section 4.4.1 since the order of R-tree node ex-

pansion has a significant effect to the performance of the algorithms. The elements in Q_r are in the decreasing order of their *maxInf* value. As the *maxInf* value of entries keeps changing during the query, Q_r is designed to be mutable.

Q_e is initialized as root entries of all query relevant R-trees and Q_r is initialized as the root node entry of category t_1. For each entry e in Q_r, we store two values, $\langle minInf, maxInf \rangle$, denoting the minimum and maximum influence values of all facilities indexed by e respectively. Specifically, if e is an object, *minInf* and *maxInf* are the lower bound and an upper bound of its influence value. If e is an index entry, *minInf* and *maxInf* are the lower and upper bound of the influence of all facility objects in the sub-tree rooted at e. We keep expanding the highest priority entry in Q_e and pushing into Q_e its child index entries and Q_c its object entries not belonging to t_1 respectively. At the same time, we update the *maxInf* and *minInf* values of entries in Q_r. We repeat this process until find k objects in Q_r each of which has a *minInf* no less than the largest *maxInf* of all remaining entries. If we do not get the answer after expanding all tree index entries, we then compute the optimal functional unit for objects in Q_c one by one until we can determine to stop the algorithm.

To implement Algorithm 4 efficiently, we store, for each entry e, two sets of entry pointers, namely *dstEnts* and *srcEnts*. Specifically, *dstEnts* stores the entries that are *possible participants* of e, where the formal definition of possible participants is introduced in the following section. Reversely, *srcEnts* stores the entries, each of which regards e as one of its possible participants. The implementation of two steps in Algorithm 4 is discussed in Section 4.4 and Section 4.5 respectively.

There are two major issues to resolve *SInQuery* well.

- Tight possible participant pruning distance finding. Loose pruning distance leads to a large size of possible participant set. Consequently, it will slow

down the speed of estimating *maxInf* and *minInf* accurately. Moreover, it will add heavier burden to Step 2.

- The computation of optimal functional unit for an object o is NP-hard as shown in Lemma 4.1. Trivially enumerating all possible solutions is computationally expensive and slow.

4.4 Possible Participant Set Finding

Step 1 in Algorithm 4 aims to find the possible participant set for each object $o \in \mathcal{O}_q \setminus \mathcal{O}_{t_1}$, Given an object o, in order to find the optimal functional unit of o, we need a set of candidate objects from which we can find o's optimal functional unit. We call these candidate objects o's possible optimal functional unit participants, shortly **possible participants**, and such a candidate set o's **possible participant set**. We can straightforwardly generalize the concept to tree entries. That is, for a tree entry e, its possible participants are a set of tree entries which themselves or their descendant objects would become an optimal functional unit member of objects in e.

Apparently, the relationship between entry possible participants and the estimations of *maxInf* and *minInf* is that the fewer of entry possible participants, the more accurate estimations of *maxInf* and *minInf*. Therefore, we should always keep as few possible participants as possible. Algorithm 5 outlines the main steps to find the possible participant set. In the following subsections, we discuss the three steps in detail.

Algorithm 5: **Step 1**

```
1 Q_e ← {all root index entries of relevant R-trees };
2 Q_r ← root entry of type t_1; Q_c ← ∅;
3 while Q_e ≠ ∅ do
4     Step 1.1: Picking An Entry to Expand. Dequeue the first entry e from Q_e;
      Expand e and insert into Q_e all e's child index entries and Q_c all e's object
      entries not belonging to t_1;
5     Step 1.2: Pruning Distance Finding. For each relevant entry, find a tight
      pruning distance to prune its possible participant set;
6     Step 1.3: Updating maxInf and minInf. Update the maxInf and minInf values
      for relevant entries in Q_r;
7     if the k entires in Q_r with maximum maxInf are all objects, with minInf no less
      than the maxInf of all remaining entries then
8         C ← k objects with maximum maxInf;
9         return C
```

4.4.1 Expanding Entry Picking

As long as there exists an index entry, we pick an entry e with the highest priority to expand. Our aim is to make the *maxInf* and *minInf* estimations of entries as accurate as possible. We consider entry area as the expanding priority of entries in Q_e. Intuitively, if e has a large area, it is easy to deduce that e would affect as well as be affected by many other entries. However, lots of these affecting relations are actually false positive. To expand e, we can prune these false positive affecting relations. As a result, we can reduce the difference between *maxInf* and *minInf* for relevant entries. To verify this intuition, we conduct comparison experiments using entry weight, that is, the number of objects contained in the subtree rooted at the

entry. The results show that the entry area always has the best performance.

4.4.2 Pruning Distance Finding

For each relevant entry e, we need to find a pruning distance to prune its possible participant set. We start with the case how to prune the possible participant set of an object o. We use the best known solution principle, which confines the possible participants of o in a disk. Naively, if we have a functional unit S of o, then we know that any object o' with $d(o, o') > cost(S)$ can be pruned. Nevertheless, this pruning bound is too loose. We present a much tighter pruning distance which is optimal.

Lemma 4.2. *Given an object o and a query q with m categories, let S be a current best functional unit of o. Then, for any functional unit S' of o containing at least one object p with $d(o, p) \geq \frac{cost(S)}{m-1}$, $cost(S') \geq cost(S)$, and this bound is optimal.*

Proof. For any object p' in the remaining $m-2$ objects of S', $d(p', o) + d(p', p)$ will be only minimized when p' is on segment (o, p) according to the triangle inequality, and the minimum distance is $d(o, p)$. Hence, for $m-2$ objects, the sum of minimum distances is $(m-2)d(o, p)$. Let the sum of pairwise distance between these $m-2$ objects be sum. It is obvious that sum will achieve its minimum value 0 only when all these $m-2$ objects are at the same position. Therefore, $cost(S') = (m-2)(d(p', o) + d(p', p)) + d(o, p) + sum \geq (m-1)d(o, p) + sum$. Since $d(o, p) \geq \frac{cost(S)}{m-1}$ and $sum \geq 0$, it follows that $cost(S') \geq cost(S)$.

We now prove the optimality of this bound. Assume that there are $m-1$ objects in the same location. Let p be one of the $m-1$ objects and $d(o, p) = \frac{cost(S)}{m-1} - \varepsilon$ where ε is an arbitrarily small positive value. We further assume that the $m-1$ objects together with o form a functional unit. Denote these m objects by S'.

Clearly, $cost(S') = (m-1)d(o,p) = (m-1)(\frac{cost(S)}{m-1} - \varepsilon) < cost(S)$. Therefore, we know that p might be one of the participants of o's optimal functional unit when $d(o,p) < \frac{cost(S)}{m-1}$. □

The above lemma suggests that all objects that o might choose to form its optimal functional unit must be in a disk $D(o, \frac{cost(S)}{m-1})$ with o being the center and $\frac{cost(S)}{m-1}$ being the radius where S is o's current best functional unit. All other objects can be pruned safely.

Lemma 4.2 can be easily extended to a tree entry e. If the optimal functional unit cost of any object in e can be bounded by $\bar{U}$, then any entry e' with *minDist(e, e')* $> \frac{\bar{U}}{m-1}$ can be pruned from e's possible participant set. Specially, this pruning distance of an MBR R can be denoted by a round corner rectangle (RCR) which consists of all points p with *minDist(p, R)*$=\frac{\bar{U}}{m-1}$ and being outside of R.

With the above property, we know that we need to find a cost upper bound for each relevant entry e to derive its pruning distance. This cost should satisfy two conditions. (i) Within this cost, every object in e can find its optimal functional unit; and (ii) It should be as small as possible. This reminds us of the functional unit cost computation methods that we discussed in Section 4.2.2. However, while they meet condition (i), both of the mentioned methods own two major drawbacks.

Drawback 1. *Rectangle distance metric based cost is substantially larger than the actual cost of object optimal functional unit.*

Example 4.3. *Consider Figure 4.3 where we want to find the pruning distance for entry R. Suppose that entries S_1 and T_1 can achieve the optimal edgeExistDist based cost which is $l_1+l_2+l_3$. Then we can get the corresponding pruning distance for R, which is $\frac{l_1+l_2+l_3}{2}$ as depicted by the round corner rectangle RCR_1 (Collapsed distance is shown for interest of space). Nevertheless, if we know the optimal functional unit*

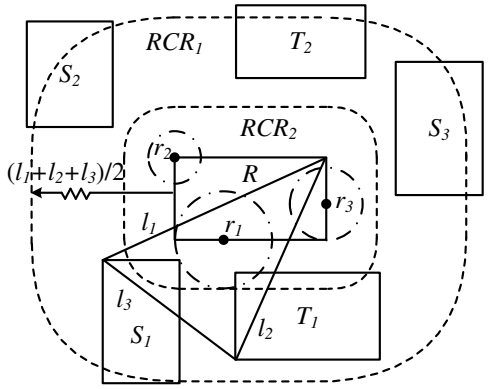


Figure 4.3: Gap between *edgeExistDist* based pruning distance and the optimal pruning distance

cost of each object in R, we can get the corresponding pruning distances, as depicted in the dashed line circles. Using these pruning distances of objects, we can easily derive the optimal pruning distance RCR_2 of R, which is significantly smaller than RCR_1. As we can see, entries S_2, S_3 and T_2 can be pruned by RCR_2, while not by RCR_1.

Drawback 2. *It is time consuming to dynamically find the best pruning distance.*

Example 4.4. *Recall Example 4.3. In order to find out the combination S_1 and T_1 of R, we have to exhaustively verify all pairs from type S joining type T, which is time consuming. This problem goes severely worse when the category number of a query increases.*

According to the above two drawbacks, for an object, we aim to efficiently find a functional unit with a cost close to that of the optimal functional unit. Besides, we can use this cost to construct the pruning distance of its ancestor tree entries. Motivated by this, we propose a **nearest neighbor set (NNS)** based method, which builds the entry pruning distance separately.

Given an object o and type T, the **T-type nearest neighbor** of o, denoted by $NN(o,T)$, is defined to be the nearest neighbor of o in $\mathcal{O}_T$. Given a query $q = (\{t_1, t_2, ..., t_m\})$, without loss of generality, we always suppose that $o.\lambda$ is t_1. We define the **nearest neighbor set** of o, denoted by $N(o)$, to be the set containing o's T-type nearest neighbor for each $T \in q.\phi \setminus t_1$.

Lemma 4.3. *Let $S \subseteq \mathcal{O}$ be an arbitrary set of m objects and $o \in S$ be arbitrary. Define $R = S \setminus \{o\}$ and $sum(o,R) = \sum_{r\in R} d(o,r)$. Then, $cost(S) \leq (m-1)sum(o,R)$.*

Proof. For any two objects $o_i, o_j \in R$, we have $d(o_i,o_j) \leq d(o,o_i) + d(o,o_j)$. Since there are $\frac{(m-1)(m-2)}{2}$ different pairs of o_i, o_j in total, we have the same number of inequalities. By summing up all these inequalities, we have $cost(R) \leq (m-2)sum(o,R)$. Therefore, $cost(S) = cost(R) + sum(o,R) \leq (m-1)sum(o,R)$. □

The above lemma suggests that the cost of an arbitrary set is bounded by the sum of distance of an arbitrary object to others in the set, as long as the distance triangle inequality holds.

Theorem 4.1. *Given an object o and a query q with m types, let $N(o)$ be o's nearest neighbor set and S_o be o's optimal functional unit. Then, $cost(N(o)\cup\{o\}) \leq (m-1)cost(S_o)$.*

Proof. We define $sum(o,S) = \sum_{o'\in S} d(o,o')$. According to Lemma 4.3, we know that $cost(N(o) \cup \{o\}) \leq (m-1)sum(o,N(o))$. On the other hand, we have $cost(S_o) = sum(o, S_o \setminus \{o\}) + cost(S_o \setminus \{o\}) \geq sum(o, S_o \setminus \{o\})$. For each type $T \in q.\psi \setminus \{t_1\}$, we assume that the corresponding object in S_o is o'. According to the definition of T-type nearest neighbor, we have $d(o, NN(o,T)) \leq d(o,o')$. Therefore, $sum(o,N(o)) = \sum_{T\in q.\psi\setminus\{t_1\}} d(o, NN(o,T)) \leq \sum_{o'\in S_o\setminus\{o\}} d(o,o') = sum(o, S_o\setminus\{o\})$.

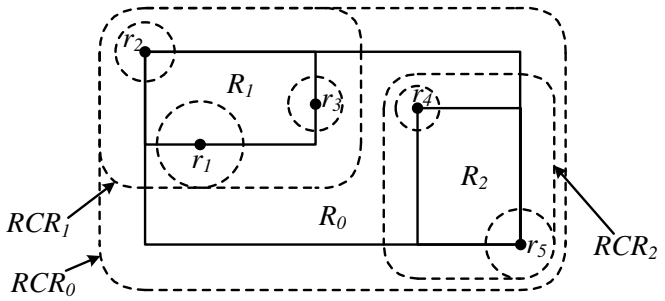


Figure 4.4: Pruning distance construction for R-tree entries

From the above three relations, we have $cost(N(o) \cup \{o\}) \leq (m-1)cost(S_o)$, which completes the proof. □

Theorem 4.1 says that the nearest neighbor set has a cost guarantee regarding the optimal cost for an arbitrary object. At the same time, the nearest neighbors can be efficiently computed in polynomial time. In order to avoid the Drawback 2, we compute the entry pruning distance separately. That is, we build up the pruning distance for all query relevant R-tree entries prior to Step 1 in Algorithm 4. Algorithm 6 gives an outline of NNS based framework. For interest of context clarity, we denote the inserted steps by Step 0a and Step 0b respectively. In Step 0a, we generalize the all nearest neighbor query method [CP07] to find the nearest neighbor set for each object. Then, in step 0b, we build up the pruning distance for every entry in the relevant R-trees in a bottom-up way.

Example 4.5. *Consider Figure 4.4 containing five objects, namely* r_1, r_2, r_3, r_4 *and* r_5*. Suppose the dashed line circles are corresponding NNS-based pruning distances. Using the pruning distance of objects, we build those (*RCR_1 *and* RCR_2*) of their direct parents (*R_1 *and* R_2*). We continue this process until the tree root entry* R_0

For the specific pruning distance computation method, we refer the readers to the previous work [QZK+12]. We note here that the time complexity to compute

the pruning distance for one entry is linear to the number of its child entries which can be bounded by the capacity of an R-tree node.

Now, in Step 1.2, we do not need to find the pruning distance on-the-fly any more. Instead, we simply use the computed pruning distance to prune the possible participant set.

Algorithm 6: **NNS-based TopInfluentialObjects**

1 Step 0a: *Nearest Neighbor Set Finding.* For each object $o \in \mathcal{O}_q$, find its nearest neighbor set in a top-down multiway join manner;

2 Step 0b: *Pruning Distance Building up.* Build up the pruning distance for each entry e in the R-trees corresponding to $\mathcal{O}_q$ in a bottom-up manner;

3 Step 1: *Possible Participant Set Finding.* As long as unable to determine the final result, we select an unprocessed object $o \in \mathcal{O}_q$ to find a set of candidate objects which would become a member of its optimal functional unit;

4 Step 2: *Optimal Functional Unit Computation.* As long as unable to determine the final result, we select an unprocessed object $o \in \mathcal{O}_q$ to compute its optimal functional unit using the possible neighbor set;

5 Output: k most influential objects in $\mathcal{O}_{t_1}$.

4.4.3 Entry Influences Updating

In order to describe how to set and update the *maxInf* and *minInf* values, we introduce a relation between two entries. For any two entries e_i and e_j of different types, we say e_i affects e_j if e_j is a possible participant of e_i. Recall the two sets *dstEnts* and *srcEnts* mentioned in Section 4.3. We know that e_j is in e_i's *dstEnts* and e_i is in e_j's *srcEnts*.

From the definition of our problem, we can see that the influence of an object o is the number of objects that pick o to form their optimal function unit. Hence,

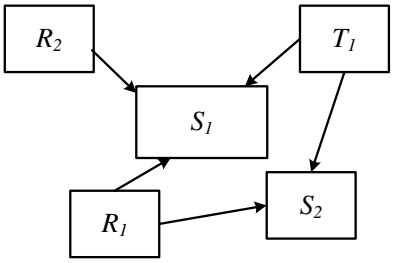


Figure 4.5: An example of affecting state

	maxInf	***minInf***
S_1	$\|R_1\| + \|R_2\| + \|T_1\|$	$\|R_2\|/\|S_1\|$
S_2	$\|R_1\| + \|T_1\|$	0

Table 4.2: Summary of *maxInf* and *minInf* for Figure 4.5

maxInf of entry e is the total weight of entries that affect e, which is the sum of entry weights in e's *srcEnts*. Here, the weight of an entry is the number of objects in its MBR. *minInf* is much more complicated. It can be computed in the following way. For each e_i affects e, if it only affects e for the type e belongs to, which means, for the type e belongs to, objects in e_i will only choose objects in e to form their function unit, we say e_i contributes to e with weight $|e_i|$. Then, we sum the weight over all such entries. Finally, the *minInf* of e is $\sum_{e_i \text{ affects only } e} |e_i|/|e|$. This rationale can be explained by "pigeon hole" principle. That is, if n people have m dollars in total, there must exist one person having at least m/n dollars.

Example 4.6. *Consider Figure 4.5 where we ignore the affecting relationships between type R and T. For type S, R_1 affects both S_1 and S_2, and so does T_1, while R_2 affects only S_1. Table 4.2 summarizes the corresponding maxInf and minInf values of S_1 and S_2.*

When expanding an entry e, we have to update the affecting state of relevant entries and further update their *maxInf* and *minInf* values. Apparently, the relevant entries include the child entries of e and these affecting e. For each of these entries,

we use the corresponding pruning distance that built in Step 0b of Algorithm 6 to prune the possible participant set.

4.4.4 Early Termination

During the processing of Step 1 (i.e., Algorithm 5), we might have a chance to terminate the algorithm early, which is illustrated in Line 7 of Algorithm 5. Formally, when the k entries in Q_r with maximum *maxInf* are all objects and their *minInf* no less than the *maxInf* of remaining entries, we can terminate the algorithm safely and return the k objects as results. We claim that this condition is necessary, which can be justified by the following example.

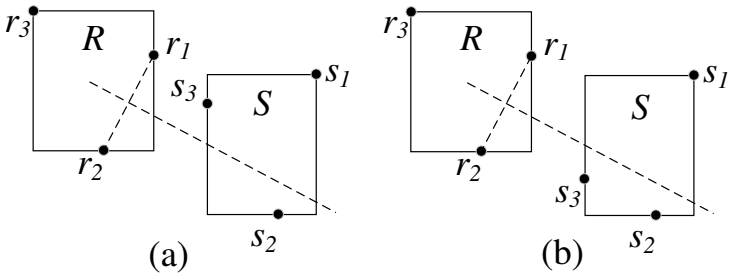


Figure 4.6: An example of early termination in Algorithm 5

Example 4.7. *Consider the example in Figure 4.6 where we assume the query consists of only two categories, namely r and s, each of which contains 3 objects. Now, suppose we aim to find the best (i.e., top-1) object in r. Consider two cases depicted in Figure 4.6(a) and Figure 4.6(b), respectively. Clearly, the best object is r_1 (Resp. r_2) in Figure 4.6(a) (Resp. in Figure 4.6(b)). If we expend R, we can compute the minInf and maxInf values for each object in R with the knowledge of number of objects in S (and this is the only information we can get without expanding S). Specifically, the minInf and maxInf values of r_1 and r_2 are the same, which are 1 and 2 respectively in both Figure 4.6(a) and Figure 4.6(b) (the*

two values of r_3 are clearly 0). Note that these values can be easily computed based on some basic properties of MBR (e.g., there is at least one object on each boundary of an MBR). Apparently, we are not able to determine the best object in R without expanding S.

The above example tells us that even though the algorithm reaches a point where k entries are found with maximum *maxInf* and their *minInf* no less than the *maxInf* of remaining entries, we still need to continue the algorithm and expand entries of other categories. It is not sufficient to focus only on the k entries of Q_r. The rationale here is as follows. In order to tighten the influential bounds *minInf* and *maxInf* of entries belonging to t_1, we have to accurate the candidate participant set of other type entries. To this end, we have to expand entries from any other types, not just t_1.

4.5 Optimal Functional Unit Computation

A crucial phase in Algorithm 4 is to find the optimal functional unit for each object. In order to terminate the algorithm as early as possible, we utilize the *maxInf* and *minInf* values as we do in Algorithm 5. That is, we keep computing the optimal functional unit for objects in Q_c. Once we find there are k objects in Q_r with maximum *maxInf* have *minInf* no less than the *maxInf* of all remaining objects, we terminate the algorithm and return the k objects as the answer. Algorithm 7 summarizes the key steps in optimal functional unit computation. While Steps 2.1 and 2.3 are relatively straightforward, Step 2.2 is critical in Algorithm 7. After Step 1 of Algorithm 4, we have, for each object, a set of possible participants. Among these possible participants, we need to find its optimal functional unit. We propose both exact and approximate algorithms to resolve this problem.

Algorithm 7: **Step 2**

```
1 Q_c ← {all unprocessed objects};
2 while Q_c ≠ ∅ do
3     Step 2.1: Picking An Object to Compute. Dequeue the first object o from Q_c;
4     Step 2.2: Finding Optimal Functional Unit. Find the optimal functional unit
      of o from its possible participant set;
5     Step 2.3: Updating maxInf and minInf. Update the corresponding maxInf and
      minInf values for the relevant objects in Q_r;
6     if the k objects in Q_r with maximum maxInf have minInf no less than the
      maxInf of all remaining objects then
7         C ← k objects with maximum maxInf;
8         return C
```

4.5.1 Exact Algorithm

In order to find the optimal functional unit, a naive way is to enumerate every possible combinations. To make this process more efficient, we utilize a set enlarging strategy which adds into the partial functional unit one object at a time and verifies the validity of the enlarged partial functional unit.

Lemma 4.4. *Given an object o and a query q with m categories, let S be a current best functional unit of o and P be a partial functional unit which contains $j(1 < j < m)$ objects including o. If $cost(P) \geq \frac{j-1}{m-1} cost(S)$, then, for any functional unit S' with P being its subset, $cost(S') \geq cost(S)$, and this bound is optimal.*

Proof. Let $P' = S' - P$. For each $p' \in P'$, it will contribute cost with at least $\frac{cost(P)}{j-1}$ to $cost(S')$ according to Lemma 4.3. Since $|P'| = m - j$, $cost(S') = (m - j)\frac{cost(P)}{j-1} + cost(P) + cost(P') = \frac{m-1}{j-1} cost(P) + cost(P') \geq cost(S)$. We can prove the optimality of this bound by a similar way in the proof of Lemma 4.2 . □

The above lemma suggests that when a partial unit already occupies relatively high cost, there is no need to enlarge it to get a full functional unit. We note here that Lemma 4.2 is a special case of Lemma 4.4. When P contains only one object (excluding o), Lemma 4.4 becomes exactly Lemma 4.2. Algorithm 8 illustrates the process of finding the optimal functional unit for an object o. We initially put o's possible participants into $m-1$ lists (line 2) each of which contains the same type of objects. We utilize o's nearest neighbor set together with o to derive an initial best known functional unit(line 4). Then we basically use a recursive method to enumerate all possible functional units to get the optimal one (line 5). According to Lemma 4.4, we verify each partial functional unit for further enlarging (line 17). Every time we obtain a better functional unit, we use it to update the current best unit S (line 9) as well as the pruning distance (line 10). Besides, we use this pruning distance to prune the possible participant lists (lines 11 and 12).

Time Complexity. Let m be the number of types in a query q. We assume that the number of objects for each type is the same, denoted by n. The cost of line 4 is $(m-1)\cdot \log n$ (it issues $m-1$ NN queries each of which takes $O(\log n)$). Let l be the number of objects in each neighbor list (line 2). Note that $l << n$ (since the pruned possible participant set is in a small region). Therefore, the cost of procedure $Trace$ is $O(l^{m-1})$ since we have to enumerate at most $O(l^{m-1})$ functional units to find the optimal one. In conclusion, the time complexity of NNS-Exact is $O((m-1)\cdot \log n + l^{m-1})$.

4.5.2 Approximate Algorithm

In this section, we propose an approximate algorithm named NNS-Appro which gives a $(2-\frac{2}{m})$-factor approximation where m is the number of categories in a query.

Algorithm 8: **NNS-Exact** (q, o, P)

Input : q : the query, o : the anchor object,
P : the possible participant set of o

Output : S : the optimal functional unit of o

```
1  m ← |q.ψ|;
2  Divide P into m − 1 lists L[1, 2, ..., m − 1] each of which consists of the same type
   of candidate participants;
3  C ← ∅;
4  S ← N(o) ∪ {o};
5  Trace(1);
6  return S
7  procedure Trace(i)
8  if i ≥ m then
9      S ← C ∪ {o};
10     D ← cost(S)/(m − 1);
11     for i ← 1 to m − 1 do
12         Prune from L[i] all objects p with d(p, o) ≥ D;
13 else
14     for each object o′ ∈ L[i] do
15         C[i] ← o′;
16         S′ ← C[1, 2, ..., i] ∪ {o};
17         if cost(S′) ≥ i/(m−1) cost(S) then
18             continue;
19         Trace(i + 1);
```

Before presenting the algorithm, we introduce the concept of "**o-excluding nearest neighbor set**". Given a query q, an anchor object o and an object p

in o's possible participant set, p's o-excluding nearest neighbor set is defined to be the set of objects each of which is the T-type nearest neighbor of p for each $T \in q.\psi - o.\lambda - p.\lambda$.

Example 4.8. *Consider Figure 4.1. Suppose the query $q.\psi$ is $\{b, c, s\}$ and the anchor object is s_1. Then, c_1's s_1-excluding nearest neighbor set is $\{b_1\}$ since $q.\psi - s_1.\lambda - c_1.\lambda = \{b\}$ and c_1's b-type nearest neighbor is b_1. Similarly, b_2's s_1-excluding nearest neighbor set is $\{c_3\}$.*

In algorithm 8, we have to exhaustively enumerate all possible functional units of an anchor node to find the optimal one in the worst case, which is time consuming. Instead, algorithm NNS-Appro, as described in Algorithm 9, utilizes a greedy strategy to solve this problem. We initially set S to be o's nearest neighbor set together with o (line 1). With the current best functional unit, we can always get a pruning distance to prune objects in the possible participant set of the anchor object (line 4). For each of the remaining possible participant objects, we find its o-excluding nearest neighbor set to construct a new functional unit S' (lines 5 and 6). We update S by S' as well as the pruning distance if S' has a smaller cost than S (lines 7-9).

Example 4.9. *Continue Example 4.8. We aim to find s_1's optimal functional unit. First, we get s_1's nearest neighbor set $\{b_2, c_2\}$, from which we know that the initial functional unit S is $\{b_2, c_2, s_1\}$. According to Lemma 4.2, we can prune some unpromising possible participants (e.g., b_1 and c_3) with the cost of S. We find that s_1's remaining possible participant set is $\{c_1, c_2, b_2\}$. Then, we check the s_1-excluding nearest neighbor set for each object in $\{c_1, c_2, b_2\}$ and verify the corresponding intermediate set S'. We shall find that S is still $\{b_2, c_2, s_1\}$.*

According to Example 4.9, the best functional unit of s_1 obtained by NNS-Appro

is $\{b_2, c_2, s_1\}$. However, the actual optimal one is $\{b_2, c_1, s_1\}$, which means that the cost of the functional unit obtained by NNS-Appro might not be minimized.

Algorithm 9: **NNS-Appro** (q, o, P)

Input : q : the query, o : the anchor object,
P : the possible participant set of o

Output : S : a functional unit of o

```
1  S ← N(o) ∪ {o};
2  m ← |q.ψ|;
3  D ← cost(S)/(m − 1);
4  for each object p ∈ P with d(p, o) < D do
5      R ← p's o-excluding nearest neighbor set;
6      S′ ← R ∪ {o} ∪ {p};
7      if cost(S′) < cost(S) then
8          S ← S′;
9          D ← cost(S′)/(m − 1);
10 return S
```

Even though the set S returned by the NNS-Appro algorithm might have a larger cost than the optimal functional unit, the difference is bounded.

Theorem 4.2. *NNS-Appro gives a $(2-\frac{2}{m})$-factor approximation for the SInQuery problem, where m is the category number of a query.*

Proof. Let S_o be the optimal functional unit of an anchor object o and S be the solution returned by NNS-Appro. Let $w \in S_o$ be arbitrary and define $M(w) = \sum_{u \in S_o} d(w, u)$, we know that $cost(S_o) = \frac{1}{2}\sum_{w \in S_o} M(w)$. Choose $o' \in S_o$ such that $M(o') = min_{w \in S_o} M(w)$. Then, we have $cost(S_o) = \frac{1}{2}\sum_{w \in S_o} M(w) \geq \frac{1}{2}\sum_{w \in S_o} M(o') = \frac{m}{2}M(o')$.

Now, we define and discuss an intermediate functional unit S' according to two cases of the value of o'. Before that, we define $Sum(o', S' \setminus \{o'\}) = \sum_{r \in S' \setminus \{o'\}} d(o', r)$

Case 1: $o' = o$. Let S' be the set obtained in line 1 of Algorithm 9. According to the definition of $N(o)$, we can easily know that $Sum(o', S' \setminus \{o'\}) \leq M(o')$.

Case 2: $o' \neq o$. We claim that o' will be processed in line 4 of Algorithm 9 since $o' \in S_o$. Let S' be the corresponding set obtained in line 6 of Algorithm 9. According to the definition of o-excluding nearest neighbor set as well as the fact that o is in both S_o and S', we have $Sum(o', S' \setminus \{o'\}) \leq M(o')$.

Thus, we have $Sum(o', S' \setminus \{o'\}) \leq M(o')$ in either case. According to Lemma 4.3, we know that $cost(S') \leq (m-1)Sum(o', S' \setminus \{o'\})$. Therefore, we have $cost(S') \leq (m-1)M(o') \leq (m-1)\frac{2}{m}cost(S_o) = (2 - \frac{2}{m})cost(S_o)$.

Since S is the final solution returned by Algorithm 9, which means that $cost(S) \leq cost(S')$. Thus, we know that $cost(S) \leq (2 - \frac{2}{m})cost(S_o)$, which completes the proof. □

Time Complexity. Let m be the number of categories in query q. We assume that the number of objects for each type is the same, denoted by n. The cost of line 1 is $(m-1) \cdot \log n$ (it issues $m-1$ NN queries each of which takes $O(\log n)$). Let l be the number of iterations (lines 4-9) in Algorithm 9. It is straightforward that the cost in each iteration is $(m-2) \cdot \log n$ since we need to get the o-excluding nearest neighbor set. Note that $l << |\mathcal{O}_q|$ since l is the number of possible participant of the anchor object. Thus, the time complexity of NNS-Appro is $O(l \cdot m \cdot \log n)$.

4.6 Empirical Studies

In this section, we empirically evaluate the efficiency and effectiveness of the proposed techniques.

4.6.1 Experimental Setup

Algorithms. As far as we know, there is no existing work investigating the problem of categorical top-k spatial influence query. In this thesis, we implement and evaluate following 4 algorithms.

- **NNS-Exact.** Our nearest neighbor set based exact algorithm (Algorithm 8).
- **NNS-Appro.** Our nearest neighbor set based approximate algorithm (Algorithm 9).
- **Baseline1.** Baseline algorithm that uses the *edgeExistDist* based cost (Section 4.2.2).
- **Baseline2.** Baseline algorithm that uses the *minExistDNN* based cost (Section 4.2.2).

Among the 4 algorithms, only NNS-Appro is approximate algorithm.

Datasets. We use two real datasets, CA and GB. CA consists of 104,770 locations of 63 different categories (e.g., church, lake and school) each of which corresponds to a facility type. GB dataset is obtained from G.B. Geological Survey and consists of 1,410,124 locations with 73 types (e.g., Cinema, Bus Stop and Park). To study the effect of data distribution, we also created synthetic datasets, denoted by SYN. The objects were randomly generated in 2-dimensional space $[0, 1]^2$. Specifically, we fixed the number of objects N as 1,000,000 . According to the expectation frequency of each type $\bar{n_i}$, which varies from 2,000 to 10,000 with difference value 2,000 , we generated 5 datasets. The number of types of each dataset was simply the value of $N/\bar{n_i}$. We then randomly assigned a type for each object.

Query Generation. Given a dataset $\mathcal{O}$ and a positive integer m, we can arbitrarily generate a query with m types. In order to control the scale of relevant

objects, we generated a query under the constraint that for each type, the number of objects (type frequency) is in a range, $[n_l, n_u]$.

All algorithms are implemented in standard C++ with STL library support and compiled with GNU GCC. Our experiments are run on a PC with Intel Xeon 2.9GHz (8 Cores) CPU and 32G memory running Debian Linux. In the experiments, we adopt the least recent use (LRU) policy to replace pages when the buffer is full where buffer size and page size are set to 0.5MB and 1KB respectively.

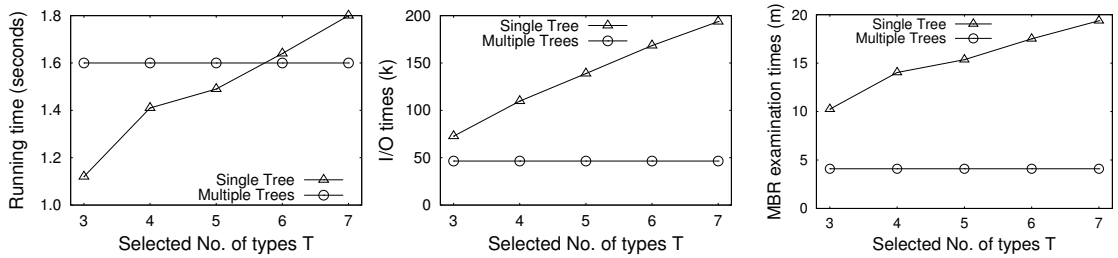
4.6.2 Experimental Results

We consider four measurements, namely, running time, approximation ratio (for approximate algorithm only), I/O times, and MBR examination times. The last two measurements are only considered for the scalability test on real dataset CA. For each set of settings, we randomly generate 20 queries, run the algorithms with each of these 20 queries, and average the measurements. The default value of output size k is 10.

Effect of Data Index Strategy. In section 4.3.2, we mentioned that there are two ways to index the dataset, namely a single tree and multiple trees. The former one is to index the whole dataset with a single tree, while the latter is to index the dataset using multiple trees (i.e., one R-tree for each object category). Intuitively, the choice is ad-hoc and dependent to the total number of categories. In order to evaluate the performance of these two strategies, we run NNS-Exact algorithm on CA under the following settings.

We first sort the categories of CA in decreasing order of their frequency. For the single tree strategy, we select the first T categories to build the tree, where $3 \leq T \leq t$ and t is the total number of categories. We fix the query containing the 3 largest categories. For an R-tree entry, we have to store 5 values, including entry

identity (*id*, integer with 4 bytes), aggregate number of data objects (*dn*, integer with 4 bytes), number of child entries (*cn*, integer with 4 bytes), signature (*sig*, 16 bytes to denote 128 categories at most), and minimum bounding rectangle (*MBR*, $2 * 2 * 8 = 32$ bytes). Note, that when using multiple trees to index data, we do not need to store *sig* any more.



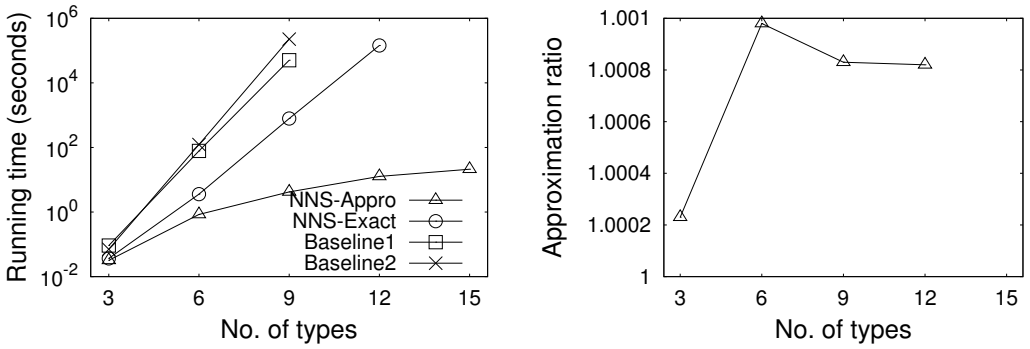
(a) Running Time (b) I/O Times (c) MBR Examination Times

Figure 4.7: Effect of Data Index Strategy on CA

From Figure 4.7, we can see that the single tree strategy runs faster than its counterpart when $T \leq 5$. However, the multiple trees strategy always achieves better I/O performance and examines less MBRs than the other. From the experimental results, we reach the conclusion that it is a better way to index the whole dataset as a single R-tree if the dataset contains only several categories of facilities. However, in most real world applications, the category number might be up to hundreds. As a result, the multiple trees strategy is the first choice.

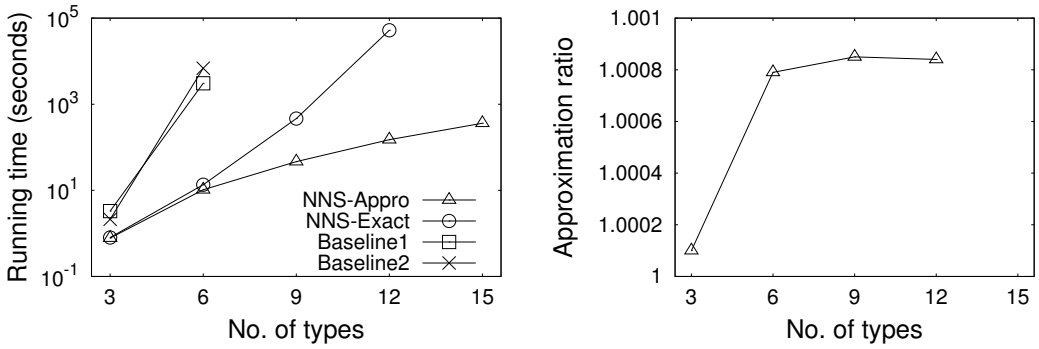
Effect of $|q.\psi|$. We generate 5 types of queries according to different values of $|q.\psi|$. The values we adopted are 3, 6, 9, 12 and 15. We set [100, 1000] and [1000, 15000] as the type frequencies of CA and GB respectively.

As demonstrated in Figure 4.8(a), the NNS based algorithms run faster than the common solution derived algorithms by several orders of magnitude when $|q.\psi|$ is no smaller than 6. We do not show the running time of an algorithm if it runs more than 10 days. Baseline1 and Baseline2 have comparable running time



(a) Running Time (b) Approximation Ratio

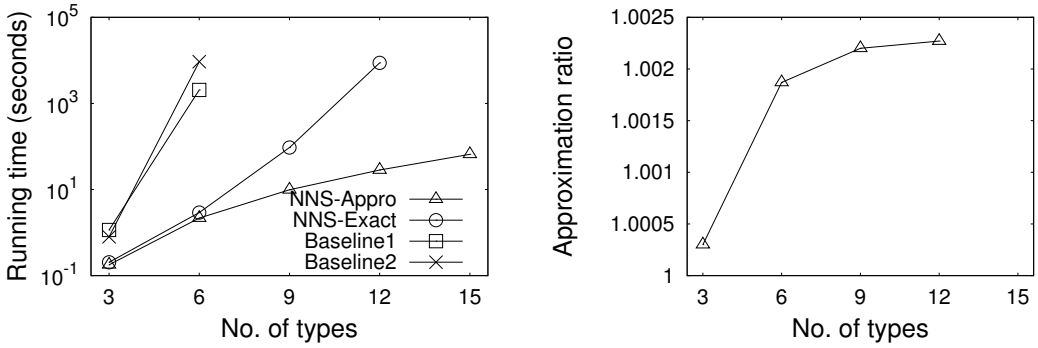
Figure 4.8: Effect of $|q.\psi|$ on CA



(a) Running Time (b) Approximation Ratio

Figure 4.9: Effect of $|q.\psi|$ on GB

when $|q.\psi| = 3$. However, the gap increases when $|q.\psi|$ increases. As we can see, Baseline1 is more than 3 times faster than Baseline2 when $|q.\psi| = 9$. This result verified our theoretical analysis that the *edgeExistDist* based cost is tighter than the *minExistDNN* based cost. Besides, the running time of all 3 exact algorithms increases exponentially due to the NP-hardness of the problem. NNS-Appro, on the other hand, performs the best. For example, NNS-Appro runs less than 100s for all settings, while NNS-Exact takes more than 10 days when $|q.\psi|$ is larger than 12 and Baseline1 and Baseline2 runs more than 10 days when $|q.\psi|$ is larger than 9. According to Figure 4.8(b), the approximation ratio is under 1.0011 in all settings, which shows that NNS-Appro algorithm can achieve extremely high accuracy in



(a) Running Time (b) Approximation Ratio

Figure 4.10: Effect of $|q.\psi|$ on SYN

practice.We note here that we were unable to give the approximation ratio when $|q.\psi|$ is 15 because all exact algorithms failed to give solution under this setting.

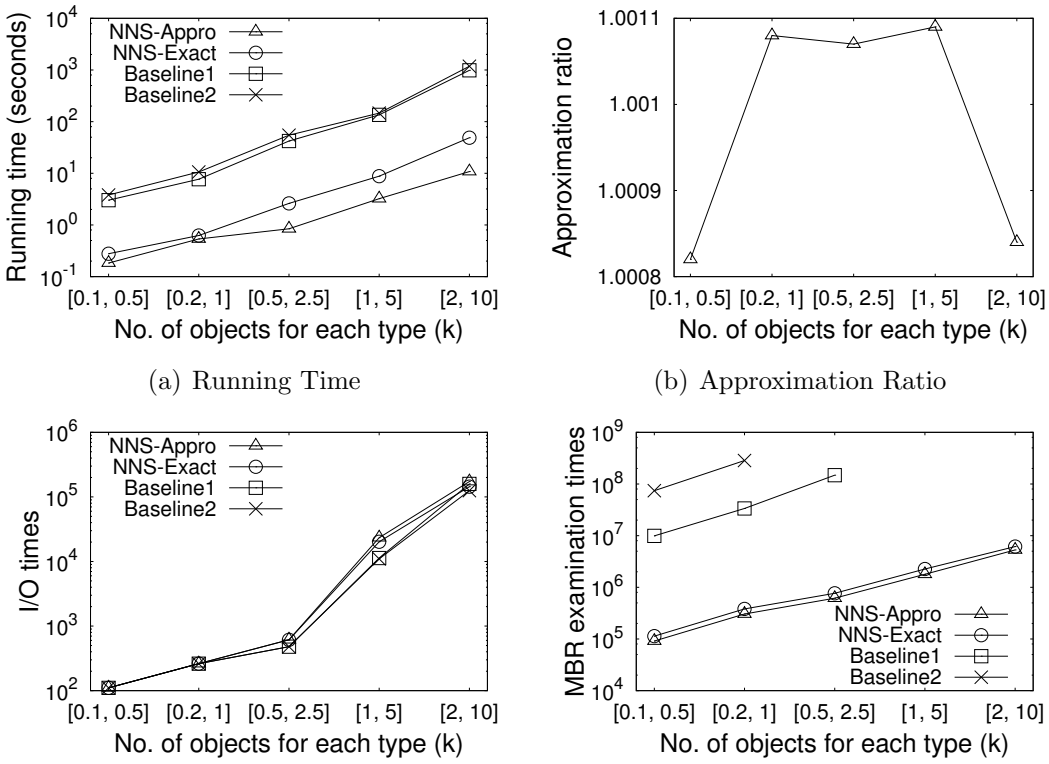
We have similar results on GB (Figure 4.9) but the running time gap between NNS based algorithms and the common solution derived algorithms is more obvious. NNS based algorithms survived when $|q.\psi|$ is 12, while the common solution derived algorithms failed when $|q.\psi|$ is only 9. Interestingly, the performance result on SYN (Figure 4.10) is extremely near that on GB, from which we can deduce that the data distribution has no effect to the performance of our proposed techniques.

Scalability Test. We fix $|q.\psi|$ to 5 in evaluating the scalability of proposed algorithms. 5 types of queries are generated according to different type frequency ranges for the real datasets CA and GB. Table 4.3 summarises the query settings. We conduct the test on all 5 synthetic datasets with $\bar{n_i}$ varying from 2,000 to 10,000 .

Dataset	**Type Frequency Ranges (k)**				
CA	[0.1, 0.5]	[0.2, 1]	[0.5, 2.5]	[1, 5]	[2, 10]
GB	[1, 10]	[2, 20]	[5, 50]	[10, 100]	[20, 200]

Table 4.3: Query Settings for Scalability Test

As shown in Figure 4.11(a), the performance of all 4 algorithms degrades linearly



(a) Running Time

(b) Approximation Ratio

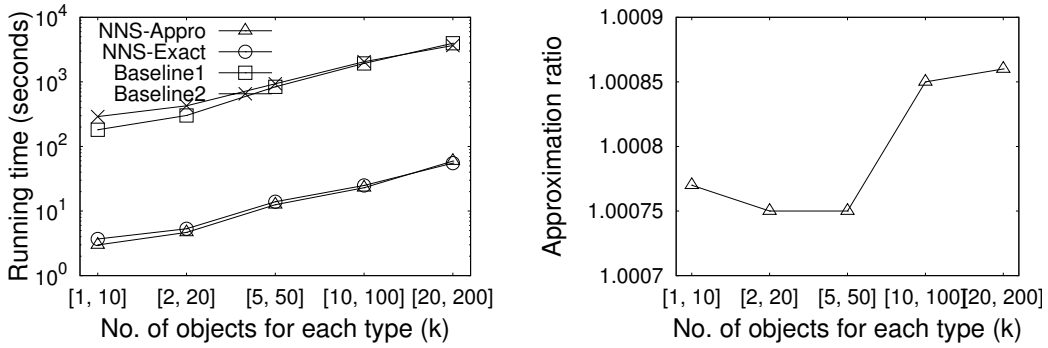
(c) I/O Times

(d) MBR Examination Times

Figure 4.11: Scalability Test on CA

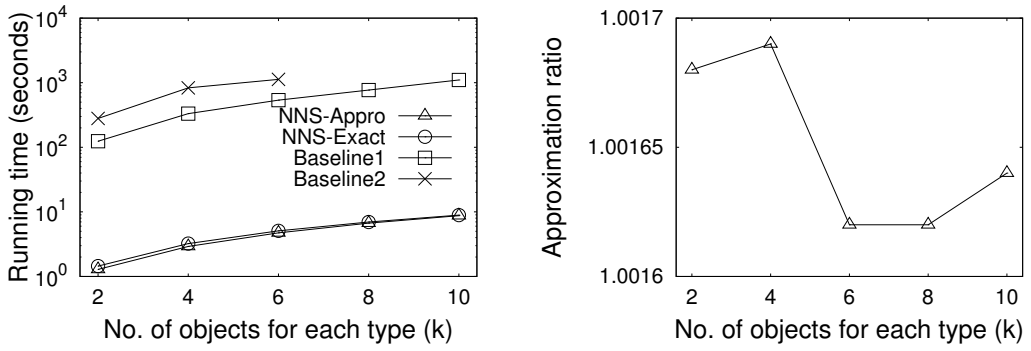
with the increase of the scale of type frequency. The results show that the NNS based algorithms run faster than the common solution derived algorithms by 1-2 orders of magnitude. Besides, NNS-Appro performs 2-4 times faster than NNS-Exact in the case when the scale is no less than [500, 5,000]. Figure 4.11(b) shows that the approximation ratio of NNS-Appro is below 1.0011 in all settings, which demonstrates that the accuracy of NNS-Appro is very high in practice. We note here that the approximation ratio is only relevant to the value of $|q.\psi|$, while not to the scalability (see Theorem 4.2). Therefore, the fluctuation in Figure 4.11(b) is trivial.

In order to give a complete comprehension to our problem, we count the I/O times as well as MBR examination times. From Figure 4.11(c), we can see that the



(a) Running Time (b) Approximation Ratio

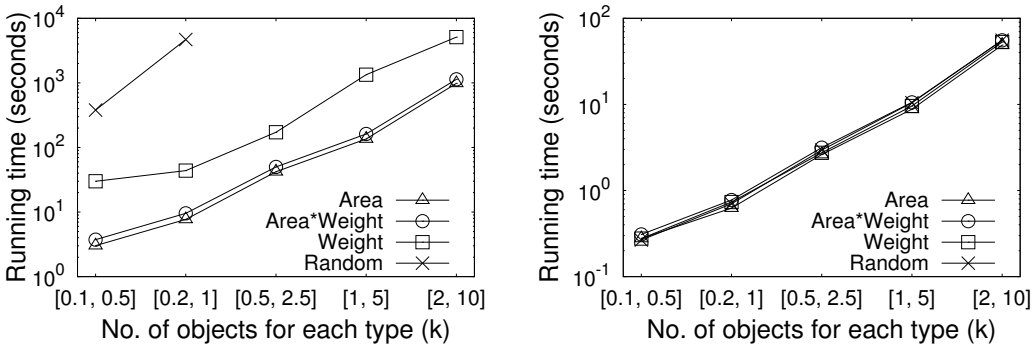
Figure 4.12: Scalability Test on GB



(a) Running time (b) Approximation Ratio

Figure 4.13: Scalability Test on SYN

NNS based algorithms have slightly larger I/O times than the baseline algorithms, which can be explained by the fact that the former has to browser the query relevant trees twice. The first time is to find the nearest neighbor set for each object and the second is to find the possible participants for objects. However, considering the fact that our proposed SInQuery problem is NP-hard, efficient algorithms in terms of running time performance are needed. Therefore, NNS based algorithms are desirable since they can achieve much better running time performance than the baseline algorithms. Besides, in terms of the number of MBR accesses, our proposed algorithms outperforms the baseline methods by 2-3 orders of magnitude (Figure 4.11(d)). This is exactly why our proposed methods are much faster than



(a) Baseline1 (b) NNS-Exact

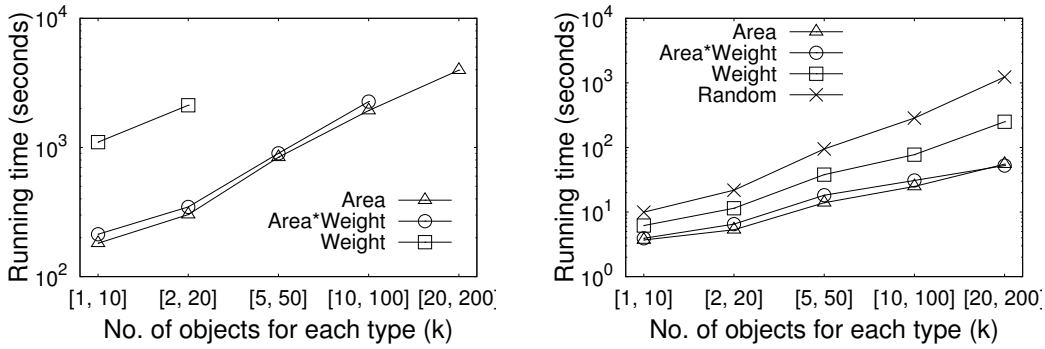
Figure 4.14: Effect of Expanding Strategy on CA

the baseline methods since the later keeps updating the pruning distance during the expansion of R-tree nodes. On the other hand, the pruning distance is quite loose as we discussed in Section 4.4.2. Thus, it can barely prune any neighbors in each round.

Figure 4.12 shows the scalability test results on GB. Similar to CA, the common solution derived algorithms have comparable running time. However, NNS-Exact and NNS-Appro have more close running time, and we can barely notice any advantage of NNS-Appro. We note here that when $|q.\psi|$ is relatively small, the time cost of computing the nearest neighbor set for all objects dominates the overall time cost. Since the only difference between NNS-Exact and NNS-Appro is to find the optimal function unit when the possible participants are given, we can deduce that they have close running time performance.

The scalability test results on SYN are shown in Figure 4.13, where we can see that the NNS based algorithms run faster than the common solution derived algorithms by more than 2 orders of magnitude. Besides, the performance gap between Baseline1 and Baseline2 has been enlarged. In contrast, the NNS based algorithms have more similar running time cost.

Effect of Expanding Strategy. In order to evaluate the effect of entry ex-



(a) Baseline1 (b) NNS-Exact

Figure 4.15: Effect of Expanding Strategy on GB

panding strategy, we choose Baseline1 and NNS-Exact to run the scalability test on CA and GB. We evaluate 4 strategies, namely Random, Area, Weight and Area∗Weight. Specifically, with Random strategy, we randomly pick an entry in the priority queue to expand. For the rest 3 strategies, we prioritize elements according to entry area, entry weight and product of entry area and weight respectively. According to Figure 4.14 and Figure 4.15, Random strategy has the worst performance, while Area strategy always achieves the best performance. From the results, we know that entry area affects the speed of reducing the difference between *maxInf* and *minInf* most. On the other hand, from Figure 4.14 and Figure 4.15, we can see that the entry expanding strategy does not affect NNS-Exact as much as Baseline1, which verifies the theoretical analysis that the common solution derived methods are more dependent on the selection of entry expanding strategy.

4.7 Conclusions

In this chapter, we investigate the problem of categorical top-k spatial influential objects query. We formally give the object influence definition under the setting where data objects are in form of different categories. To resolve this problem

efficiently, we present an efficient framework, which consists of two major steps, possible neighbors finding and optimal functional unit computation. In the first step, we derive a nearest neighbor set based pruning distance to get tight possible neighbor sets for each tree entry. In the second step, we develop exact algorithms and approximate algorithms with performance guarantee. Experimental results demonstrate the effectiveness and efficiency of our techniques.

Chapter 5

Efficient Set Containment Join

5.1 Overview

Set-valued attributes play an important role in modeling database systems ranging from commercial applications to scientific studies. For instance, a set-valued attribute may correspond to the profile of a person, the tags of a post, the links or domain information of a webpage, and the tokens or q-grams of a document. In this chapter, we focus on the problem of *set containment join*. Given two collections $\mathcal{R}$ and $\mathcal{S}$ of records, each of which contains a set of elements, the set containment join, denoted by $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$, retrieves all pairs $\{(r, s)\}$ where $r \in \mathcal{R}$, $s \in \mathcal{S}$, and $r \subseteq s$. As a fundamental operation on massive collections of set values, the set containment join benefits many applications. For instance, companies may post a list of positions on an online job market website, and each of which contains a set of required skills. Let e_i denote a skill, Table 5.1(a) shows the skills required in four job advertisements in $\mathcal{R}$. A job-seeker, on the other hand, can submit her/his curriculum vitae to the website, which lists a set of her/his skills. Table 5.1(b) illustrates the skill records of four job-seekers in $\mathcal{S}$. Naturally, a company would

id	set
r_1	$\{e_1, e_2, e_3\}$
r_2	$\{e_1, e_2, e_4\}$
r_3	$\{e_1, e_3, e_4\}$
r_4	$\{e_2, e_5\}$

(a) $\mathcal{R}$ sets

id	set
s_1	$\{e_1, e_2, e_3, e_5\}$
s_2	$\{e_1, e_2, e_4\}$
s_3	$\{e_1, e_3, e_6\}$
s_4	$\{e_2, e_4, e_5\}$

(b) $\mathcal{S}$ sets

Figure 5.1: A motivation example where $_{e_i}$ denotes a skill, $\mathcal{R}$ consists of four job advertisements with required skills, and $\mathcal{S}$ represents four job-seekers with their skills.

like to consider a job-seeker if her/his skill set covers all required skills for a position. We call such a pair of job-seeker and position a containment match. By executing a set containment join on the positions and job-seekers, the website is able to identify all possible matches, i.e., $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$, and make recommendations.

An algorithmic challenge is how to perform the set containment join in an *efficient* way. A naive algorithm is to compare every pair of records from $\mathcal{R}$ and $\mathcal{S}$, thus bearing a prohibitively $\mathcal{O}(n_r n_s)$ time complexity where n_r and n_s denote the number of records in $\mathcal{R}$ and $\mathcal{S}$, respectively. Recent research focuses on the in-memory set containment join algorithms, and several techniques have been developed following *intersection-oriented* or *union-oriented* computing paradigms. Nevertheless, we observe that two computing paradigms have their limits due to the nature of the intersection and union operators. Particularly, *intersection-oriented* method relies on the intersection of the relevant inverted lists built on the elements of $\mathcal{S}$. A nice property of the *intersection-oriented* method is that the join computation is verification free. However, the number of records explored during the join process may be large because there are multiple replicas for each record in $\mathcal{S}$. On the other hand, the *union-oriented* method generates a signature for each record in $\mathcal{R}$ and the candidate pairs are obtained by the union of the inverted lists of the relevant signatures. The candidate size of the *union-oriented* method

is usually small because each record contributes only one replica in the index. Unfortunately, *union-oriented* method needs to verify the candidate pairs, which may be cost expensive especially when the join result size is large. As a matter of fact, the state-of-the-art *union-oriented* solution is not competitive compared to the *intersection-oriented* ones.

In this thesis, we re-visit and design a new *union-oriented* method, namely *TT-Join*, where an efficient set containment join algorithm is developed based on two different prefix trees built on $\mathcal{R}$ and $\mathcal{S}$, respectively. Through comprehensive cost analysis on simple *intersection-oriented* and *union-oriented* methods in Section 5.4.2, we show that the above two problems suffered by the *intersection-oriented* methods can be easily addressed by a new simple *union-oriented* method which uses the least frequent element as the signature. Not surprisingly, the new simple *union-oriented* method needs to verify candidates due to the inherent limit of *union-oriented* computing paradigm. Moreover, its pruning capability is limited by using only one element as the signature. To circumvent these limits, we propose a new prefix tree structure based on the k least frequent elements of the records within $\mathcal{R}$ such that we can (i) enhance the pruning power with a reasonable overhead, and (ii) integrate the *intersection* semantics to directly *validate* a significant number of join results without invoking the verification. To share the computational cost among records within $\mathcal{S}$, we also build a regular prefix tree on $\mathcal{S}$. Then we develop an efficient *TT-Join* algorithm to perform set containment join against two prefix trees.

Furthermore, to support large scale of datasets, we extend our techniques to distributed systems on top of MapReduce framework We propose a novel signature-based distribution scheme, which dispatch records based on the aforementioned record signature (i.e., the least frequent element). Specifically, we first partition

the element domain into N disjoin intervals, each related to a reduce node. Then, for a record $r \in \mathcal{R}$, we find the interval where its signature falls and dispatch r to the corresponding reduce node. For a record $s \in \mathcal{S}$, it will be dispatched to all reduce nodes whose corresponding intervals cover at least one element of s. With the help of careful designed element domain partition approaches that are guided by the join cost estimation on reduce nodes, our signature-based distribution mechanism can achieve good load-balance, low communication cost, and no duplicate in join results.

Roadmap. The rest of the chapter is organized as follows. Section 5.2 presents the preliminaries. Section 5.3 introduces the existing solutions. Our approach TT-Join is devised in Section 5.4. Distributed set containment join algorithm is presented in Section 5.5. Extensive experiments are reported in Section 5.6. Section 5.7 concludes this chapter.

5.2 Preliminaries

In this section, we introduce basic concepts and definitions used in this chapter. Table 5.1 summarizes the important mathematical notations used throughout this chapter.

In this chapter, each record x consists of a set of elements $\{e_1, e_2, \ldots, e_{|x|}\}$ from element domain $\mathcal{E}$. We use $\mathcal{X}$ to denote a relation with a set-valued attribute, i.e., a collection of records. By default, elements in a record are in decreasing order of their frequency in $\mathcal{X}$. Following the convention, we use $\mathcal{R}$ (resp. $\mathcal{S}$) to denote the left (resp. right) side relation (i.e., a collection of records) for the set containment join. Similar, we use r (resp. s) to denote a record within $\mathcal{R}$ (resp. $\mathcal{S}$).

Given two records r and s, we say r is contained by s, denoted by $r \subseteq s$, if all

Notation	Definition
$x, \mathcal{X}; r, \mathcal{R}; s, \mathcal{S}$	a record, a set of records
$e, \mathcal{E}$	an element, element domain
$\mathcal{R}(s)$	all records $r \in \mathcal{R}$ with $r \subseteq s$
$\mathcal{S}(r)$	all records $s \in \mathcal{S}$ with $r \subseteq s$
σ	signature of a record
$I_{\mathcal{R}}(\sigma)$	inverted list for signature σ in $\mathcal{R}$
$I_{\mathcal{S}}(e)$	inverted list for element e in $\mathcal{S}$
$T_{\mathcal{R}}, T_{\mathcal{S}}$	indexing tree on $\mathcal{R}$ / $\mathcal{S}$
v, w	a node in $T_{\mathcal{R}}$ / $T_{\mathcal{S}}$
$v.e, w.e$	record element in v / w
$v.set, w.set$	elements from root to v / w
$v.prefix, w.prefix$	elements from root to parent of v / w
$v.list, w.list$	records stop at v / w
$P(e)$	frequency distribution of elements
$\theta(l)$	distribution of record cardinality
$\|x\|_{avg}, \|r\|_{avg}, \|s\|_{avg}$	average size of records in $\mathcal{X}$, $\mathcal{R}$, $\mathcal{S}$
$\|x\|_{max}, \|r\|_{max}, \|s\|_{max}$	maximal size of records in $\mathcal{X}$, $\mathcal{R}$, $\mathcal{S}$

Table 5.1: The summary of notations

elements of r can be found in s. That is, for $\forall e \in r$, we have $e \in s$. In the thesis, we also say r is a *subset* of s and s is a *superset* of r if $r \subseteq s$. For a record $r \in \mathcal{R}$, we use $\mathcal{S}(r)$ to denote all records $s \in \mathcal{S}$ with $r \subseteq s$. Similarly, $\mathcal{R}(s)$ denotes all records $r \in \mathcal{R}$ with $r \subseteq s$.

Definition 5.1 (**Set Containment Join**). *Given two collections $\mathcal{R}$ and $\mathcal{S}$ of records, the set containment join between $\mathcal{R}$ and $\mathcal{S}$, denoted by $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$, is to find all pairs (r, s), such that $r \in \mathcal{R}$, $s \in \mathcal{S}$, and $r \subseteq s$. That is $\mathcal{R} \bowtie_{\subseteq} \mathcal{S} = \{(r, s) | r \in \mathcal{R}$, $s \in \mathcal{S}$, and $r \subseteq s\}$.*

Example 5.1. *Consider the example in Fig. 5.1. The result of set containment join is as follows: $\mathcal{R} \bowtie_{\subseteq} \mathcal{S} = \{(r_1, s_1), (r_2, s_2), (r_4, s_1), (r_4, s_4)\}$.*

5.3 Existing Solutions

A brute-force solution for set containment join is to enumerate and verify $|\mathcal{R}||\mathcal{S}|$ pairs of records, which is cost-prohibitive. To improve the efficiency of computation, many advanced algorithms are proposed in the literature. We classify them into two categories based on their computing paradigms, namely *intersection-oriented* methods [Mam03, JP05, LFHDB15, BMGT15, KRS+16] and *union-oriented* methods [HM97, RPNK00, MGM02, MGM03, LFHDB15]. We also review several methods proposed for string similarity search [LLL08, WLF12, AAK10].

5.3.1 Intersection-Oriented Methods

Given two record collections $\mathcal{R}$ and $\mathcal{S}$, the key idea of *intersection-oriented* method is to build inverted index on $\mathcal{S}$ and then apply the **intersection** operator to calculate $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$. In this thesis, we say these algorithms are $\mathcal{S}$-driven methods because their main index structures are built on $\mathcal{S}$.

Algorithm 10 illustrates a simple *intersection-oriented* method [Mam03], namely **RI-Join**[1]. We use $I_{\mathcal{S}}(e)$ to denote the inverted list of an element e built on records in $\mathcal{S}$, which keeps IDs of the records containing the element e. Fig. 5.2 depicts the inverted index of $\mathcal{S}$ in the example of Fig. 5.1. Lines 1-2 build the inverted index of $\mathcal{S}$. Then for each record $r \in \mathcal{R}$, we can immediately identify $\mathcal{S}(r)$ (i.e., record $s \in \mathcal{S}$ with $r \subseteq s$) based on the intersection of the inverted lists for elements within r (Lines 4-6).

The dominant cost of Algorithm 10 is the intersection of the inverted lists

[1]Algorithm 10 is named **RI-Join** in this paper since there is no index on $\mathcal{R}$ and an inverted index is built on $\mathcal{S}$.

Algorithm 10: **RI-Join** ($\mathcal{R}$, $\mathcal{S}$)

Output : $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$

1 **for** each record $s \in \mathcal{S}$ **do**

2 Update inverted list $I_{\mathcal{S}}(e)$ for every $e \in s$;

3 $J := \emptyset$;

4 **for** each $r \in \mathcal{R}$ **do**

5 $C := \bigcap_{e \in r} I_{\mathcal{S}}(e)$;

6 $J := J \cup \{(r, s)\}$ for every record $s \in C$;

7 **return** J

(Line 5). We have

$$Cost(\mathcal{R} \bowtie_{\subseteq} \mathcal{S}) = \sum_{r \in \mathcal{R}} \sum_{e \in r} |I_{\mathcal{S}}(e)|. \tag{5.1}$$

Analysis. A nice property of the *intersection-oriented* approach is *verification free*. On the downside, a significant drawback is that we need to consider every element of a record for inverted index construction (Line 2). This may lead to long inverted lists and hence a large number of records accessed during the join process (Line 5).

Below are details of advanced *intersection-oriented* set containment join algorithms.

Algorithm PRETTI. Jampani *et* al. [JP05] propose a method called PRETTI to improve the performance of *intersection-oriented* method. Instead of processing each individual record in $\mathcal{R}$, a prefix tree $T_{\mathcal{R}}$ is built on $\mathcal{R}$ to share the computational cost. We define a (regular) prefix tree as follows.

Definition 5.2 (**Prefix Tree**). *Each node v in the tree (except root) is associated to an element in $\mathcal{E}$, denoted by $v.e$. We use $v.set$ to denote the set of elements associated with v and its ancestors. Similarly, we denote all elements in its ancestors*

Algorithm 11: **PRETTI**($T_{\mathcal{R}}$, $I_{\mathcal{S}}$)

Input : $T_{\mathcal{R}}$: prefix tree on $\mathcal{R}$, $I_{\mathcal{S}}$: inverted indexes on $\mathcal{S}$,

Output : $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$

```
1  for each child node v of the root of T_R do
2      processNode(v, I_S(v.e), J);
3  return J
4  procedure processNode(v, list, J)
5  list ← list ∩ I_S(v.e);
6  for each record r ∈ v.list do
7      for each record s ∈ list do
8          J ← J ∪ {(r, s)};
9  for each child node v_i of node v do
10     processNode(v_i, list, J);
```

by $v.prefix$ (i.e., $v.prefix := v.set \setminus v.e$). We also use a list, denoted by $v.list$, to keep the IDs of all records $\{x\}$ with $x = v.set$. Note that elements in each record follow a global order, and hence each record is assigned to a unique tree node.

Fig. 5.3 shows the prefix tree for the record set $\mathcal{R}$ in Fig. 5.1(a). By utilizing the prefix tree, we can share computation among records with the same prefix. For instance, the intersection for inverted lists of $I_{\mathcal{S}}(e_1)$ and $I_{\mathcal{S}}(e_2)$ only needs to be performed once when we compute the superset of r_1 and r_2.

Algorithm 11 illustrates the details of PRETTI, which traverses the prefix tree on $\mathcal{R}$ in a depth-first manner. For each node v visited, we use *list* to denote the intersection of the inverted lists of the elements in $v.prefix$, which is passed from its parent node. Based on the intersection of *list* and the inverted list of the element $v.e$ $I_{\mathcal{S}}(v.e)$ (Line 5), we obtain the list of records in $\mathcal{S}$ each of which contains all

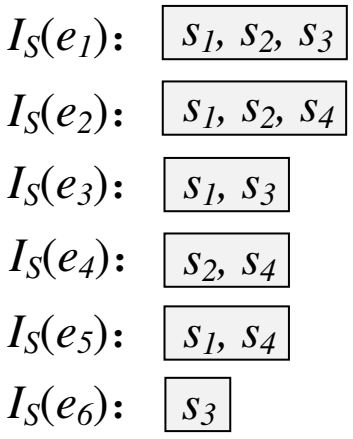


Figure 5.2: Inverted index on $\mathcal{S}$

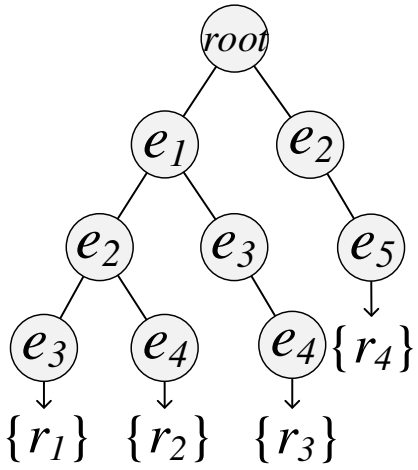


Figure 5.3: Prefix tree on $\mathcal{R}$

elements in $v.set$. Lines 6-8 generate the join results regarding the node v. Then the join will continue through its child nodes where the updated *list* will be passed (Lines 9-10).

Algorithm PRETTI+. To reduce the size of the prefix tree, Luo *et* al. [LFHDB15] introduce an extension of PRETTI, namely PRETTI+. In particular, PRETTI+ employs a compact prefix tree, called Patricia trie, to replace the prefix tree in PRETTI. This new prefix tree is the same as the previous one except that the nodes along a single path are merged into one node. The Patricia trie on the records set $\mathcal{R}$ in Fig. 5.1(a) is shown in Fig. 5.4. We omit the details of PRETTI+, which are the same as PRETTI except that we may need to merge inverted lists of multiple elements associated with a node.

Algorithm LIMIT. To avoid exploring many inverted lists for the large size records within $\mathcal{R}$, Bouros *et* al. [BMGT15] propose a new algorithm, called LIMIT. Instead of building a complete prefix tree for $\mathcal{R}$, LIMIT only builds a prefix tree with limited height k; that is, only considers the prefix of record with a fixed length. Fig. 5.5 shows a limited prefix tree with $k = 2$ for records set $\mathcal{R}$ in Fig. 5.1(a).

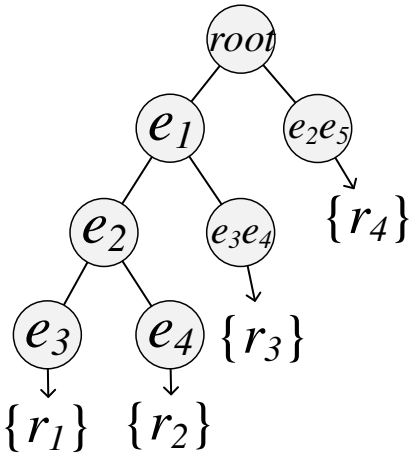


Figure 5.4: Patrica trie on $\mathcal{R}$

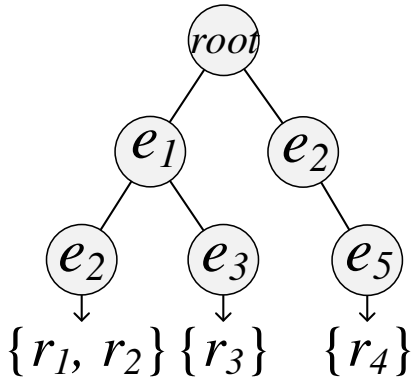


Figure 5.5: Limited tree on $\mathcal{R}$

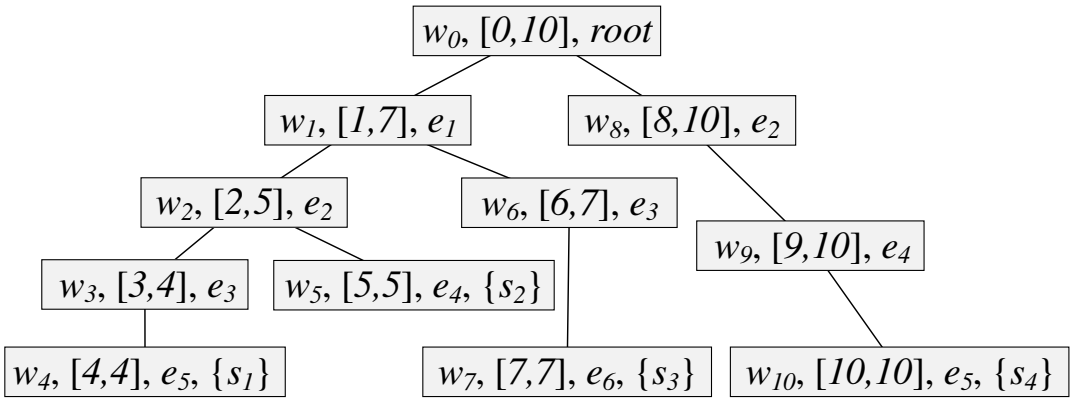


Figure 5.6: Augmented prefix tree on $\mathcal{S}$

Based on the limited prefix tree, LIMIT performs the join process following a two-phase procedure which involves *candidates generation* and *candidates verification*. In terms of algorithm implementation, LIMIT is basically the same as Algorithm 11 except the generation of join results (Lines 8 in Algorithm 11). Particularly, LIMIT handles this by considering two scenarios. If $|r| \leq k$, we output the record pair (r, s) directly since the inverted lists of *all* elements in r participate in the intersection. Otherwise, we have to verify the record pair (r, s). Although we need to verify some candidate pairs in LIMIT due to the limited tree height, this

Algorithm 12: **PIEJoin**($T_{\mathcal{R}}$, $T_{\mathcal{S}}$)

Input : $T_{\mathcal{R}}$ prefix tree on $\mathcal{R}$, $T_{\mathcal{S}}$: prefix tree on $\mathcal{S}$

Output : $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$

```
1  search(T_R.root, T_S.root, J);
2  return J
3  procedure search(v, w, J)
4  lookForOutput(v, w, J);
5  for each child node v_i of node v do
6      W ← T_S.findNodes(w, v_i.e);
7      for each child node w_j ∈ W do
8          search(v_i, w_j, J);
9  procedure lookForOutput(v, w, J)
10 if v.list ≠ ∅ then
11     list ← T_S.getRecords(w);
12     for each tuple r ∈ v.list do
13         for each tuple s ∈ list do
14             J ← J ∪ {(r, s)};
```

cost is well paid off by significantly reducing the number of inverted lists involved in the intersection. As a matter of fact, our empirical study shows that LIMIT is the most promising *intersection-oriented* method on most of the datasets evaluated.

Algorithm PIEJoin. Recently, Kunkel *e*t al. [KRS+16] propose a two-tree based method, called PIEJoin, which aims to improve the performance of *intersection-oriented* method by exploiting advanced index technique on $\mathcal{S}$. PIEJoin builds two prefix trees $T_{\mathcal{R}}$ and $T_{\mathcal{S}}$ on relations $\mathcal{R}$ and $\mathcal{S}$, respectively, together with auxiliary structures on each tree node. In particular, for $T_{\mathcal{R}}$, each node is labeled with a

preorder ID (e.g., $v_0, ..., v_9$ in Fig. 5.3), while for $T_{\mathcal{S}}$, there is a preorder interval on each node. Fig. 5.6 shows the augmented prefix tree $T_{\mathcal{S}}$ for $\mathcal{S}$ in Fig. 5.1(b).

The details of PIEJoin are illustrated in Algorithm 12, which traverses two prefix trees simultaneously. The search starts from the root of $T_{\mathcal{R}}$ and $T_{\mathcal{S}}$ (Line 1). On each tree node pair v and w, we check if there are some join pairs found (Line 4). In particular, if $v.list$ is not empty (Line 10), then we find all records in the subtree rooted at w and enumerate join pairs (Lines 11-14). After collecting results in current tree node pair, we go further by traversing the children of v. For each child v_i, we find the descendants of w such that the element contained in these nodes is $v_i.e$ (Line 6). We then recursively conduct the search process for each node pair v_i and w_j (Line 8).

Compared to the previous solutions, PIEJoin employs a tree structure on records in $\mathcal{S}$, instead of the inverted index. This alleviates the problem of the large size inverted lists for $\mathcal{S}$. However, some auxiliary structures have to be engaged to facilitate the node match at Line 6. Note that we need to find the matches within the whole subtree, which may be cost expensive. As reported in [KRS+16], the performance of PIEJoin is not competitive compared with LIMIT [BMGT15], which builds inverted index on $\mathcal{S}$, under most of the datasets evaluated.

5.3.2 Union-Oriented Methods

In general, all methods in this category use *signature-based* techniques. Let $\mathcal{L}$ denote the domain of the signature values, we use a hash function h to map a record x into a subset of signature values, denoted by $h(x)$, with $h(x) \subseteq \mathcal{L}$. For instance, in the partition-based containment join algorithm [RPNK00], a record x will be mapped into a number between 0 and $k-1$. These algorithms are also named $\mathcal{R}$-driven methods because the main index is built on records in $\mathcal{R}$.

Given two record collections $\mathcal{R}$ and $\mathcal{S}$, the key idea of *union-oriented* method is to generate **candidate records** within $\mathcal{R}$ for each record $s \in \mathcal{S}$ by the **union** of the inverted lists for the relevant signatures. Algorithm 13 illustrates a framework of simple *union-oriented* method. Lines 1-2 build inverted lists for possible signatures on $\mathcal{R}$. For each record $r \in \mathcal{R}$, Line 2 attaches its ID to the inverted list of the corresponding signature, denoted by $I_{\mathcal{R}}(\sigma)$. Then Lines 4-7 generate containment join result candidates based on the union of the inverted lists of the signatures. For a record $s \in \mathcal{S}$, we consider the inverted lists of the signatures generated by s or any of its subsets. Line 8 verifies the candidate pairs within J to remove the false positives. Note that in the implementation, we usually do not need to explicitly enumerate all subsets of s to generate M_s as shown at Line 5. Instead, M_s can be generated efficiently by exploiting the characteristics of the specific signatures used.

Algorithm 13: **A framework of simple *union-oriented* method($\mathcal{R}$, $\mathcal{S}$)**

Output : $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$

1 **for** each $r \in \mathcal{R}$ **do**

2 $\quad \sigma \leftarrow h(r)$; Update $I_{\mathcal{R}}(\sigma)$;

3 $J := \emptyset$;

4 **for** each $s \in \mathcal{S}$ **do**

5 $\quad M_s \leftarrow$ all possible signatures can be generated by s or any of its subsets;

6 $\quad C \leftarrow \bigcup_{\sigma \in M_s} I_{\mathcal{R}}(\sigma)$;

7 $\quad J := J \cup \{(r, s)\}$ for every record $r \in C$;

8 Verify candidate pairs within J;

9 **return** J

The dominant cost of Algorithm 13 is the union of the inverted lists (Line 5) ,

denoted by C_{filter}, and the verification cost (Line 8), denoted by C_{vef}. We have

$$\begin{aligned} Cost(\mathcal{R} \bowtie_{\subseteq} \mathcal{S}) &= C_{filter} + C_{vef} \\ &= \sum_{s \in \mathcal{S}} \sum_{\sigma \in M_s} |I_{\mathcal{R}}(\sigma)| + C_{vef}. \end{aligned} \tag{5.2}$$

Analysis. Compared with the *intersection-oriented* method in Algorithm 10, we need to verify the candidate pairs due to the nature of signature techniques, which usually brings false positives. Nevertheless, the advantage is that there is only one signature for each record. This leads to a smaller inverted index, and hence a smaller number of records explored during the join process (Line 6).

Below are details of the existing *union-oriented* algorithms classified based on their signature techniques.

Partition-based Techniques. In [RPNK00], a partition based method is proposed, where the signature of a record x is a number between 0 and $k-1$. For a given record $r \in \mathcal{R}$, the hash function h randomly selects an element e from r and maps e to an integer value between 0 and $k-1$ (Line 2 of Algorithm 13). We use this value as the signature of r. For a given record $s \in \mathcal{S}$, the signature subset M_s (Line 5 of Algorithm 13) consists of different signatures generated by all elements of s. Obviously, for two given collections $\mathcal{R}$ and $\mathcal{S}$, we can divide the candidate join pairs into k partitions. Fig. 5.7 shows the partitions for datasets in Fig. 5.1 where we assume that $k = 4$ and an element e_i is mapped into the value (i mod k). Several follow-up studies [MGM02, MGM03] propose more sophisticated partitioning strategies (i.e., hash function h) to reduce the number of candidates in the partition pairs.

Bitmap-based Techniques. Helmer *e*t al. [HM97] use a bitmap as the signature of a record x, and a bitmap consists of fixed b bits. Given two bitmaps b_1 and b_2, we say $b_1 \subseteq b_2$ if every 1 bit in b_1 is also set to 1 in b_2. An important property of

partition	$R_i \bowtie S_i$
0	$\{r_2\} \bowtie \{s_2, s_4\}$
1	$\{r_1\} \bowtie \{s_1, s_2, s_3, s_4\}$
2	$\{r_4\} \bowtie \{s_1, s_2, s_3, s_4\}$
3	$\{r_3\} \bowtie \{s_1, s_3\}$

Figure 5.7: Partition-based method

bitmap-based signature is that we have $h(x) \subseteq h(y)$ if $x \subseteq y$ for any two records x and y. This implies that, for a given record $s \in \mathcal{S}$ we can safely exclude a record $r \in \mathcal{R}$ from containment join if $h(r) \not\subseteq h(s)$. However, the task of enumerating all possible signatures by a record s or any of its subset (Line 5 of Algorithm 13) would be a bottleneck for the bitmap-based *union-oriented* method, since the the number of subsets of a signature is exponential to the bitmap length b. To avoid such a straightforward way of enumerating the subsets of a signature, Luo *et.* al [LFHDB15] recently propose a new algorithm, named **PTSJ**, based on a *trie-based* subsets enumeration method. In this method, the signatures of records in $\mathcal{R}$ are stored in a trie, where the leaf nodes store the record id in $\mathcal{R}$. Then, given a record $s \in \mathcal{S}$, we employ a breath-first search on the trie. For each tree node v, we store the bit values 0 and 1 in the left child and right child, respectively. If the corresponding bit value of $h(s)$ is 0, we only explore the left child. Otherwise, we will visit both children. Once we finish this traversal, the records in leaf nodes we accessed are the candidates.

Discussion. PTSJ algorithm proposed in [LFHDB15] is the state-of-the-art in-memory *union-oriented* method which significantly enhances the previous solutions in this category by advanced signature enumeration method and careful bitmap length selection. Nevertheless, our empirical study shows that PTSJ is not com-

petitive compared with other state-of-the-art *intersection-oriented* solutions. According to our analysis in Section IV-B2, PTSJ has two significant drawbacks: (*i*) does not utilize the data distribution; and (*ii*) needs to verify all candidate pairs obtained.

5.3.3 Apply Set Similarity based Methods

We are aware that existing set similarity search/join algorithms can be applied to support set containment join by setting specific thresholds. Here, we consider three representative works. It is worth mentioning that they are $\mathcal{S}$-driven methods in the sense that their main index structures are built on records from $\mathcal{S}$.

Li *e*t al. [LLL08] propose an efficient list merging algorithm, named DivideSkip, to solve the generalized T-occurrence query problem. Given a query record Q, T-occurrence problem is to find the set of record IDs that appear at least T times on the inverted lists of the elements in Q, where the inverted lists are built on $\mathcal{S}$. By setting T to the size of Q, DivideSkip can be immediately employed to process set containment search. Using a nested loop, DivideSkip can also be extended to compute set containment join.

Wang *e*t al. [WLF12] propose an adaptive framework for set similarity search, which adaptively selects the length of record prefix to build the inverted index. Since they apply the overlap similarity to handle different set similarity functions, by setting the overlap threshold T to the size of query Q, this framework can also be utilized to compute set containment join.

Agrawal *e*t al. [AAK10] study the problem of error-tolerant set containment search. To boost the query performance, they propose an frequent element set based index structure that builds inverted index on careful chosen element set. By setting the error-tolerate threshold as 1, this index structure can also be applied

to answer exact set containment query, and therefore, is also applicable to set containment join.

5.4 Our Approach

In this section, we introduce a new in-memory set containment join algorithm, namely *TT-Join*, based on two tree structures constructed on $\mathcal{R}$ and $\mathcal{S}$, respectively.

5.4.1 Motivation

Our empirical study suggests that the existing competitive in-memory set containment join algorithms follow the *intersection-oriented* computing paradigm. However, to enjoy the nice property of verification free, we need to keep the ID of a record for each of its elements in the inverted index. This is an inherent limit of the *intersection-oriented* method which may lead to a large number of records explored during the join processing, especially when the number of inverted lists involved is large. Although an augmented prefix tree has been proposed in PIEJoin [KRS$^+$16] to alleviate this issue, our empirical study suggests that the result is unsatisfactory due to the complicated data structure and expensive search cost incurred. Moreover, our analysis in Section IV-B2 also suggests that it is difficult for *intersection-oriented* methods to exploit the data distribution.

This motivates us to re-visit and design a new *union-oriented* approach. The drawbacks of existing *union-oriented* methods are two-fold: (*i*) the signature techniques used are data-independent, which cannot better exploit the distribution of the elements; (*ii*) they need to verify all candidate pairs. In this chapter, we aim to design a new *union-oriented* method which not only enhances the nice property of *union-oriented* methods (i.e., small inverted list size) but also effectively addresses

the above two issues.

In Section 5.4.2, we apply the ranked key [YGM94] technique to use the least significant element as the signature of the record in the simple *union-oriented* algorithm (Algorithm 13). Through comprehensive cost analysis, we show that the performance of the new simple *union-oriented* method can significantly outperform that of simple *intersection-oriented* method (Algorithm 10) when data becomes skewed. It is rather intuitive to further enhance the filtering capacity by using k least frequent elements. We extend the inverted indexing of the new simple *union-oriented* method to accommodate the k least frequent elements based signature. Nevertheless, we show that a simple extension of inverted index is not promising due to the large overhead incurred.

This motivates us to impose a tree structure to accommodate the k least frequent elements based signatures. In Section 5.4.3, we build a prefix tree based on the k least frequent elements of the records in $\mathcal{R}$. By doing so, we can (i) further reduce the candidate size with a small overhead; (ii) naturally apply the intersection operator to validate a large number of candidates and hence reduce the verification cost. Together with a regular prefix tree constructed on $\mathcal{S}$, we develop an efficient set containment join algorithm, namely *TT-Join*.

5.4.2 Inverted Index Based Method

In this section, we introduce a simple *union-oriented* algorithm in Section 5.4.2-A which uses the least frequent element as the signature. Section 5.4.2-B conducts cost comparison between two simple *intersection-oriented* and *union-oriented* algorithms to reveal their inherent advantages and limits. Then Section 5.4.2-C investigates an extension of the inverted index to use k least frequent elements as the signature of a record such that the number of candidate pairs can be further

reduced.

5.4.2-A Using the least frequent element (IS-Join)

As shown in Section 5.3.2, different signature techniques are employed by the existing solutions to improve the performance of simple *union-oriented* method. However, none of them consider the distribution of the elements. To take advantage of the skewness of the real-life data, we apply the ranked key [YGM94] technique to use the least significant element (i.e., least frequent element) as the signature of the record in the simple *union-oriented* algorithm (Algorithm 13). Our new simple *union-oriented* method, namely **IS-Join**[2], is immediate, based on two minor changes of Algorithm 13: (1) at Line 2, the hash function h simply returns the least frequent element as the signature; (2) at Line 5, M_s is the set of elements in s, i.e., considering $|s|$ signatures.

Algorithm correctness. For any result pair (r, s) (i.e., $r \subseteq s$), let σ be the signature of r (i.e., the least frequent element), we have $r \in I(\sigma)$. Since $r \subseteq s$, we have $\sigma \in s$ and hence $\sigma \in M_s$ at Line 5. It is immediate that $r \in C$ (Line 6). After verification at Line 8, *IS-Join* algorithm can identify the pair (r, s).

Next, we use the running example in Fig. 5.1 to show the advantage of our least frequent element based simple *union-oriented* method by comparing with the *RI-Join* (Algorithm 10).

Example 5.2. *The inverted index on $\mathcal{S}$ and the least frequent inverted index on $\mathcal{R}$ are shown in Fig. 5.2 and Fig. 5.8, respectively. According to Equation 5.1, we know that the cost for simple intersection-oriented method is* 28*, which is obtained by summing up the size of all inverted list in $I_{\mathcal{S}}$ for each record. Similarly, we have that*

[2]The new simple *union-oriented* method is named **IS-Join** because an inverted index is built on $\mathcal{R}$ and there is no index on $\mathcal{S}$.

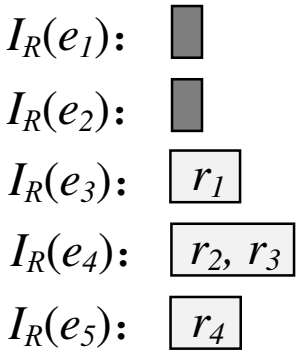


Figure 5.8: Inverted index on $\mathcal{R}$

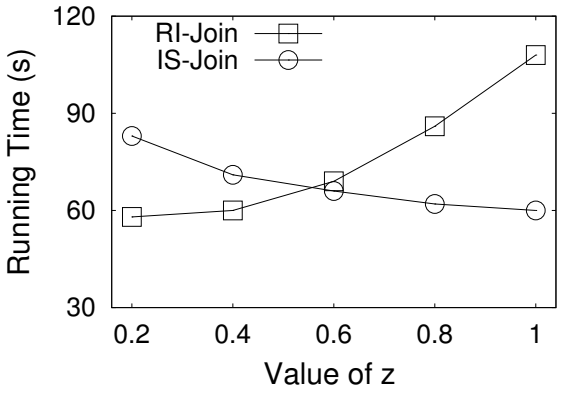


Figure 5.9: Effect of data skewness

the candidate size of union-oriented method is 8 *according to Equation 5.2, which means that the total cost is* $8 \times \mathcal{T}_{vef}$*, where* $\mathcal{T}_{vef}$ *is the cost to verify a candidate.*

The above example shows the candidate set of our *union-oriented IS-Join* algorithm is much smaller compared to that of *intersection-oriented RI-Join* algorithm. When the verification cost of *IS-Join* algorithm is not expensive, it has a good chance to outperform *RI-Join* algorithm.

5.4.2-B Cost comparison

We now theoretically compare the expected costs of *RI-Join* (i.e., a simple *intersection-oriented* method in Algorithm 10) and the *IS-Join* algorithm (i.e., a simple *union-oriented* method in Algorithm 13 where the least significant element is used as the signature), denoted by C_{RI} and C_{IS}, respectively. We use $P(e)$ to denote the frequency distribution of an element $e \in \mathcal{X}$. Let $\theta(l)$ denote the probability that a record has l elements with $l \in [1, |x|_{max}]$ where $|x|_{max}$ is the maximum cardinality of a record in $\mathcal{X}$. In the cost analysis of this paper, we assume that $\mathcal{R}$ and $\mathcal{S}$ have the same distributions in terms of element frequency and record size. Moreover, we assume $|\mathcal{R}| = |\mathcal{S}| = n$, $|r|_{avg} = |s|_{avg} = m$, and the distributions are independent.

Estimating C_{RI}. Since each element of any record in $\mathcal{S}$ leads to one entry in the inverted lists $I_{\mathcal{S}}$, we know that the expected number of entries in the inverted index is $|\mathcal{S}| \times |s|_{avg}$ where $|s|_{avg} = \sum_{l=1}^{|s|_{max}} \theta(l) \times l$ is the average size of a record in $\mathcal{S}$. Therefore, the size of the inverted list $I_{\mathcal{S}}(e)$ can be estimated as follows:

$$|I_{\mathcal{S}}(e)| = P(e) \times |\mathcal{S}| \times |s|_{avg}. \tag{5.3}$$

According to Equation 5.1, we have

$$\begin{aligned} C_{RI} &= \sum_{r \in \mathcal{R}} \sum_{e \in r} |I_{\mathcal{S}}(e)| \\ &= |\mathcal{R}| \times |r|_{avg} \times \sum_{e \in \mathcal{E}} P(e)|I_{\mathcal{S}}(e)| \\ &= |\mathcal{R}| \times |r|_{avg} \times |S| \times |s|_{avg} \times \sum_{e \in \mathcal{E}} P(e)^2 \\ &= (nm)^2 \times \sum_{e \in \mathcal{E}} P(e)^2. \end{aligned} \tag{5.4}$$

Equation 5.4 shows that, when the number of records (n) and the average size of the records (m) are fixed, *RI-Join* will achieve its best performance when all elements have the same frequency because $\sum_{e \in \mathcal{E}} P(e) = 1$. This implies that the skewness of the frequency distribution will deteriorate the performance of this simple *intersection-oriented* method, while it is well known that many real-life data are skewed.

Estimating C_{IS}. We first estimate the size of inverted list $I_{\mathcal{R}}(e)$ for an element e. Given a record $r \in \mathcal{R}$, r is in $I_{\mathcal{R}}(e)$ if and only if $e \in r$ and there is no element $e' \in r$ with lower frequency than e. Thus, the probability that r within $I_{\mathcal{R}}(e)$, denoted by $P(r \in I_{\mathcal{R}}(e))$, is

$$P(r \in I_{\mathcal{R}}(e)) = P(e) \times F(e)^{|r|-1}$$
$$= \sum_{l=1}^{|r|_{max}} \theta(l) \times l \times P(e) \times F(e)^{l-1}, \quad (5.5)$$

where $F(e) = \sum_{e' \prec e} P(e')$ is the cumulative probability before e where elements are ranked by frequency decreasing order; that is, $F(e)$ is the probability that a random chosen element has a higher frequency than e. Note that once an element e appears within the record r, it will serve as the signature with probability $F(e)^{|r|-1}$ due to the independent assumption. Thus, the expected size of list $I_{\mathcal{R}}(e)$ is as follows:

$$|I_{\mathcal{R}}(e)| = \sum_{r \in \mathcal{R}} P(r \in I_{\mathcal{R}}(e))$$
$$= |\mathcal{R}| \times \sum_{l=1}^{|r|_{max}} \theta(l) \times l \times P(e) \times F(e)^{l-1}. \quad (5.6)$$

According to Equation 5.2, we have

$$C_{IS} = \sum_{s \in \mathcal{S}} \sum_{\sigma \in M_s} |I_{\mathcal{R}}(\sigma)| + C_{vef}$$
$$= \sum_{s \in \mathcal{S}} \sum_{e \in s} |I_{\mathcal{R}}(e)| + C_{vef}$$
$$= |\mathcal{S}| \times |s|_{avg} \times \sum_{e \in \mathcal{E}} P(e) \times |I_{\mathcal{R}}(e)| + C_{vef}$$
$$\overset{(5.6)}{=} (nm)^2 \times \sum_{e \in \mathcal{E}} P(e)^2 \times F(e)^{m-1} + C_{vef}. \quad (5.7)$$

Compared with Equation 5.4, it is immediate that the number of records explored by our *union-oriented RI-Join* algorithm is smaller than that of *intersection-oriented IS-Join* algorithm since $F(e) < 1$. Our empirical study below clearly shows that this gain will eventually pay off the verification cost (C_{vef}) when the skewness of the data increases.

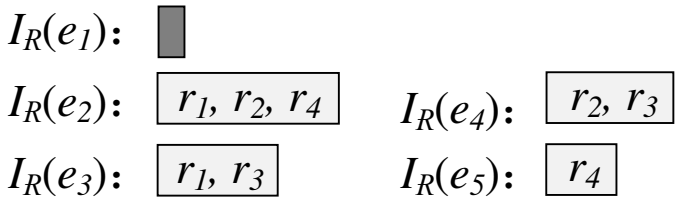


Figure 5.10: 2 least frequent elements based inverted index on $\mathcal{R}$

Empirical evaluation. To evaluate the impact of the skewness towards the performance of two algorithms, we conduct a simple experiment on synthetic datasets. In particular, we generate datasets where the frequency of the elements follow the well-known Zipfian distribution with exponent z value varying from 0.2 to 1. Note that the data skewness increases when z grows. The number of records and the average record size are set to $100,000$ and 10, respectively.

It is observed in Fig. 5.9 that *intersection-oriented IS-Join* algorithm outperforms our simple *union-oriented RI-Join* algorithm when z is small due to the extra verification cost of *RI-Join*. However, as z increases, the processing time of *IS-Join* continuously grows, while *RI-Join* can take great advantage of the skewness.

5.4.2-C Extending to k least frequent elements (kIS-Join)

According to the above cost analysis, the least frequent element is a promising signature for *union-oriented* methods. To enhance the pruning capacity, it is natural to consider k least frequent elements. Following the existing inverted index technique, now each record is mapped to k elements (signatures). Fig. 5.10 shows an example of the inverted index on $\mathcal{R}$ in Fig. 5.1(a) when $k = 2$.

Then, for a given record $s \in \mathcal{S}$, we count the number of appearances for the records in C (Line 6 in Algorithm 13). If a record $r \in C$ appears k times (i.e., all k least frequent elements of r are contained in s), r is a candidate. Otherwise, we

can prune r directly. We use **kIS-Join** to denote this algorithm which corresponds to *IS-Join* algorithm when $k = 1$.

Estimating C_{kIS}. Similar to the cost analysis for *IS-Join* algorithm, we first estimate the size of inverted list $I_{\mathcal{R}}(e)$ for an element e. Note that $I_{\mathcal{R}}$ is the inverted index based on the k least frequent elements of records in $\mathcal{R}$. Given a record $r \in \mathcal{R}$, r is in $I_{\mathcal{R}}(e)$ iff e is one of r's k least frequent elements. Thus, the probability that r is in $I_{\mathcal{R}}(e)$, denoted by $P(r \in I_{\mathcal{R}}(e))$, is:

$$\begin{aligned} P(r \in I_{\mathcal{R}}(e)) &\approx P(e) \times \sum_{i=1}^{k} F(e)^{l-i} \\ &= \sum_{l=1}^{|r|_{max}} \theta(l) \times l \times P(e) \times \sum_{i=1}^{k} F(e)^{l-i}. \end{aligned} \tag{5.8}$$

Now, the size of list $I_{\mathcal{R}}(e)$ is as follows:

$$\begin{aligned} |I_{\mathcal{R}}(e)| &= \sum_{r \in \mathcal{R}} P(r \in I_{\mathcal{R}}(e)) \\ &\overset{(5.8)}{=} |\mathcal{R}| \times \sum_{l=1}^{|r|_{max}} \theta(l) \times l \times P(e) \times \sum_{i=1}^{k} F(e)^{l-i}. \end{aligned} \tag{5.9}$$

According to Equation 5.2, we have

$$\begin{aligned} C_{kIS} &= \sum_{s \in \mathcal{S}} \sum_{\sigma \in M_s} |I_{\mathcal{R}}(\sigma)| + C_{vef} \\ &= \sum_{s \in \mathcal{S}} \sum_{e \in s} |I_{\mathcal{R}}(e)| + C_{vef} \\ &= |\mathcal{S}| \times |s|_{avg} \times \sum_{e \in \mathcal{E}} P(e) \times |I_{\mathcal{R}}(e)| + C_{vef} \\ &\overset{(5.9)}{=} (nm)^2 \times \sum_{e \in \mathcal{E}} P(e)^2 \times \sum_{i=1}^{k} F(e)^{m-i} + C_{vef}. \end{aligned} \tag{5.10}$$

By comparing Equation 5.10 and Equation 5.7, we know that the later is a special case of the former when $k = 1$. Clearly, on one hand, the pruning cost of C_{kIS} increases with k because *kIS-Join* touches more records due to the large

inverted index size. On the other hand, the verification cost C_{vef} decreases with k since a larger k can prune more non-promising records. Therefore, there is a trade-off between these two costs. Our experimental results in Section 5.6.1 show that the performance gain for C_{vef} brought by a larger k value usually cannot pay-off the increased pruning costs.

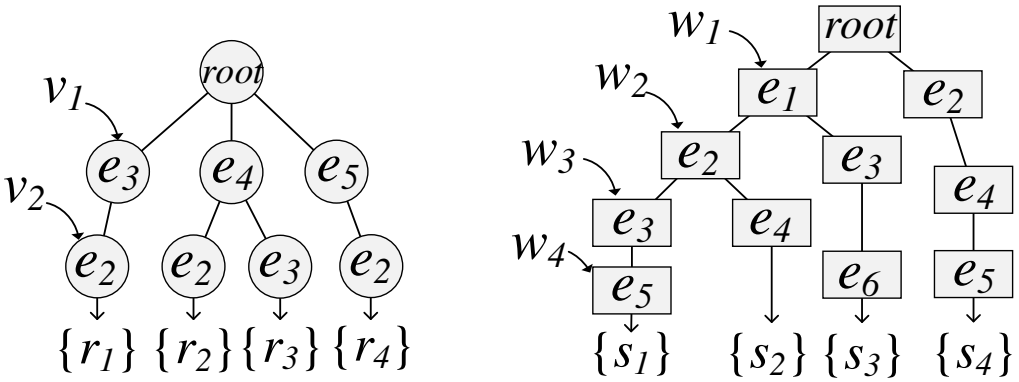
5.4.3 Tree Based Method (*TT-Join*)

It is rather intuitive that the pruning power of our simple least frequent element based *union-oriented* method can be enhanced by increasing k. However, as shown in the above analysis and empirical study, the overhead cost brought by a straightforward extension of the inverted index is expensive and the gain of the enhanced pruning capacity may not be well paid off. In this subsection, we aim to develop a new *union-oriented* algorithm which enables us to: (i) enhance the pruning capacity with small overhead; and (ii) output some join result pairs during the tree traversal without going to the verification phase. Section 5.4.3-A introduces the k-length least frequent prefix tree structure, namely *kLFP-Tree*, which is built on records in $\mathcal{R}$. Together with a prefix tree constructed on records in $\mathcal{S}$, Section 5.4.3-B presents our *TT-Join* algorithm by traversing two prefix trees simultaneously. Section 5.4.3-C conducts performance analysis on the *TT-Join* algorithm.

5.4.3-A k-length least frequent prefix tree (*kLFP-Tree*)

The *kLFP-Tree* is constructed based on the k-length least frequent prefix of each record, which is defined as follows.

Definition 5.3 (k-**length least frequent prefix**). *Given a record* $x = \{e_1, ..., e_n\}$, *we define* $\{e_n, ..., e_{n-k+1}\}$ *as its* k*-length least frequent prefix, denoted by* $LFP_k(x)$. *Note that,* $LFP_k(x)$ *is the reverse of* x *if* $|x| \leq k$.



(a) *kLFP-Tree* on $\mathcal{R}$ (b) prefix tree on $\mathcal{S}$

Figure 5.11: Tree structures for tree based method

Given a set of k-length least frequent prefixes of the records in $\mathcal{R}$, the prefix tree (*kLFP-Tree*) is built up following Definition 5.2. Specifically, for each record x, we insert the last k elements (i.e., k least frequent elements in x) into the prefix tree following the reverse order, and it takes $O(1)$ time to insert each element as a hash table is used to maintain child entries for each node in *kLFP-Tree*. Thus, the time complexity to construct *kLFP-Tree* is $O(|\mathcal{R}|k)$. With the same time complexity, we may remove a record x in *kLFP-Tree* by deleting its k least frequent elements in order. Note that there is only one replica of a record x, whose ID is kept on the corresponding node of *kLFP-Tree* based on $LFP_k(x)$.

Example 5.3. *Take the relation $\mathcal{R}$ in Fig. 5.1(a) as an example. When $k = 2$, we have $LFP_k(r_1) = \{e_3, e_2\}$, $LFP_k(r_2) = \{e_4, e_2\}$, $LFP_k(r_3) = \{e_4, e_3\}$, and $LFP_k(r_4) = \{e_5, e_2\}$. Then the corresponding kLFP-Tree is illustrated in Fig. 5.11(a).*

Algorithm 14: **TT-Join**($T_{\mathcal{R}}$, $T_{\mathcal{S}}$, k)

Input : $T_{\mathcal{R}}$: index tree on $\mathcal{R}$, $T_{\mathcal{S}}$: index tree on $\mathcal{S}$,
k : length of least frequent prefix for $\mathcal{R}$

Output : $\mathcal{R} \bowtie_{\subseteq} \mathcal{S}$

```
1  for each child node w of the root of T_S do
2      processNode(w, ∅, J);
3  return J
4  procedure processNode(w, list, J)
5  v ← findChild(T_R.root, w.e);
6  if v ≠ NULL then
7      list ← list ∪ traverse(v, w);
8  for each record s ∈ w.list do
9      for each record r ∈ list do
10         J ← J ∪ {(r, s)};
11 for each child node w_i of node w do
12     processNode(w_i, list, J);
13 procedure traverse(v, w)
14 list ← ∅;
15 for each record r ∈ v.list do
16     if |r| ≤ k then
17         list ← list ∪ {r};
18     else
19         verify(r, w.set, list);
20 for each child node v_i of node v do
21     if v_i.e ∈ w.prefix then
22         traverse(v_i, w);
23 return list
```

5.4.3-B *TT-Join* algorithm

We use $T_{\mathcal{R}}$ to denote the *kLFP-Tree* built on relation $\mathcal{R}$. To share computational cost, we also build a regular prefix tree for records in $\mathcal{S}$ following Definition 5.2, which is denoted by $T_{\mathcal{S}}$. Fig. 5.11 illustrates the example of $T_{\mathcal{R}}$ and $T_{\mathcal{S}}$ based on the records in Fig. 5.1. Note that we use a circle (resp. rectangle) to represent the node of the tree built on $\mathcal{R}$ (resp. $\mathcal{S}$), and each tree node is denoted by v_i (resp. w_i).

Algorithm 14 illustrates the details of *TT-Join* algoirthm. In general, we traverse $T_{\mathcal{S}}$ following a depth-first strategy (Lines 4-12). Lines 4-10 compute the relevant join result for each visited node w. Specifically, for the record s associated with w (i.e., $s = w.set$), we find all records in $\mathcal{R}(s)$. Recall that $\mathcal{R}(s)$ denotes the records within $\mathcal{R}$ which are a subset of s. We use R_1 to denote those records within $\mathcal{R}(s)$ without element $w.e$, and R_2 to denote the remaining records. In the procedure **processNode** (Lines 4-12), the *list* passed from the parent node corresponds to R_1 because we have $w.prefix \subset s$ and $w.prefix = s \setminus w.e$. Then Lines 5-7 identify the records in R_2. Particularly, Line 5 finds the node associated with element $w.e$ in $T_{\mathcal{R}}$. Then we only need to continue the search in its subtree because $w.e$ is the least frequent element in r. As shown in the procedure **traverse** (Lines 13-23), for each node v in $T_{\mathcal{R}}$ accessed, Line 17 can immediately **validate** a record r in $v.list$ if $|r| \leq k$ (i.e., r is reported **without verification**). Otherwise, we need to verify if $r \subseteq s$ at Line 19. At Lines 20-22, we continue to find potential records within R_2 if any of the child nodes matches an element in $w.prefix$. After all records within R_2 are identified, we use the updated *list* to keep all records within $R_1 \cup R_2$. Lines 8-10 output the join results associated with the node w accessed.

Algorithm correctness. For any record $s \in \mathcal{S}$, s must appear in one of the tree

nodes, say w, in $T_{\mathcal{S}}$. Because we traverse $T_{\mathcal{S}}$ in a depth-first manner, w must be considered during the traversal. For each reocord s in $w.list$, we can find all records $r \in \mathcal{R}$ with $r \subseteq s$. Particularly, every record r from R_1, which does not contain element $w.e$, will be passed from w's parent node because we have $r \subseteq w.set$ if $r \subseteq w.prefix$ and $w.set = w.prefix \cup w.e$. For any record $r \in R_2$, it must appear within the subtree rooted at node v with $v.e = w.e$ (Line 5) because $w.e$ is the least frequent element in r. Meanwhile, none of the record in R_1 may appear in this subtree since $w.e \notin r$ for every $r \in R_1$. For a record $r \in R_2$, we use v to denote the corresponding node of r in $T_{\mathcal{R}}$ with $r \in v.list$. Since we explore *all* child nodes v_i with $v_i.e \in w.prefix$ in the procedure **traverse**, we will eventually reach v and identify r. On the other hand, because we *only* explore child nodes v_i with $v_i.e \in w.prefix$, this implies that $v.set \subseteq w.set$ for every node v accessed in the procedure **traverse**. Consequently, all results validated at Lines 16-17 are correct. Thus, the join results on each node are complete and correct.

Example 5.4. *Consider the example in Fig. 5.1. The index trees on $\mathcal{R}$ and $\mathcal{S}$ are shown in Fig. 5.11(a) and Fig. 5.11(b), respectively. We traverse $T_{\mathcal{S}}$ in a depth-first manner starting from w_1. We immediately turn to $T_{\mathcal{R}}$ to see if there is a child node of the root of $T_{\mathcal{R}}$ matching the element of w_1 (i.e., e_1). The answer is no. We then continue the traversal processing until at w_3 where we find a child node v_1 in $T_{\mathcal{R}}$ with $w_3.e = v_1.e$. Next, we switch to traverse $T_{\mathcal{R}}$ starting from v_1 in a depth-first manner and find that v_2 matches w_2. At this point, we get a non-empty list (i.e., r_1) in v_2, which means that we get a candidate. We then conduct the verification and find that the remaining element e_1 of r_1 is in $w_3.set$. Therefore, r_1 is a subset of w_3. After that, we continue traversing $T_{\mathcal{S}}$ and reach w_4 where we would get two subsets r_1 and r_4. In particular, r_1 is passed from w_3 and r_4 is collected at w_4. Since the list of w_4 is not empty, we then generate join pairs, namely (r_1, s_1) and*

(r_4, s_1). *We find the full join results after finishing traversing* $T_{\mathcal{S}}$.

5.4.3-C Cost analysis

Next, we analyse the cost of *TT-Join*, followed by a cost comparison with *IS-Join* and *kIS-Join* introduced in Section 5.4.2.

Estimating C_{TT}. In *TT-Join*, we build the inverted index for the least frequent prefix of each record in $\mathcal{R}$, which means that the size of the inverted index is fixed at $|\mathcal{R}|$. Besides, because the inverted index is determined by the least frequent element of each record in $\mathcal{R}$, we have that the inverted index size is exactly the same as shown in Equation 5.6. On the other hand, for each least frequent prefix, we have to sequentially check whether a given record $s \in \mathcal{S}$ contains the remaining $k-1$ least frequent elements in the worst case. Therefore the overall cost of *TT-Join* is as follows:

$$\begin{aligned} C_{TT} &= \sum_{s \in \mathcal{S}} \sum_{\sigma \in M_s} |I_{\mathcal{R}}(\sigma)| + C_{check} + C_{vef} \\ &= (nm)^2 \times \sum_{e \in \mathcal{E}} P(e)^2 \times F(e)^{m-1} \\ &+ C_{check} + C_{vef}, \end{aligned} \tag{5.11}$$

where C_{check} is the overhead to check the least frequent elements.

Comparison with *IS-Join*. Equation 5.11 and Equation 5.7 indicate that, *TT-Join* and *IS-Join* have the same pruning cost. However, in terms of the verification cost, C_{vef} in Equation 5.11 is smaller than that in Equation 5.7, because *TT-Join* uses k least frequent elements as the signature of a record to enhance the pruning capacity. Therefore, with a reasonable checking cost C_{check}, *TT-Join* may benefit from increasing k.

Comparison with *kIS-Join*. Because both *kIS-Join* and *TT-Join* use the k

Algorithm 15: **Framework**

```
1 procedure map(⟨key, x⟩)
2 emit(list(⟨nid, x⟩));
3 procedure reduce(⟨nid, list(x)⟩)
4 split list(x) into two sets R and S;
5 compute set containment join between R and S;
6 output(⟨key, R ⋈⊆ S⟩);
```

least frequent elements as signature, C_{vef} in Equation 5.10 and Equation 5.11 are exactly the same. The experimental results in Section 5.6.1 show that the C_{check} is insignificant compared with the growth of the number of explored records when k increases. Therefore, compared with *kIS-Join*, *TT-Join* can achieve better trade-off by increasing k within a reasonable range (e.g., $1 \leq k \leq 5$ in our empirical study).

5.5 Distributed Processing

In this section, we aim to support better scalability by deploying *TT-Join* on the top of MapReduce framework. We first present the framework in Section 5.5.1. Our novel distribution mechanism is proposed in Section 5.5.2. Load-aware partitioning is introduced in Section 5.5.3.

5.5.1 Framework

Algorithm 15 illustrates the framework of computing set containment join on MapReduce. In MapReduce framework, each iteration consists of three phases, namely map phase, shuffle phase, and reduce phase. In the map phase as shown in

Lines 1-2, each map node (i.e., mapper) sequentially reads record x (i.e., record in $\mathcal{R}$ or $\mathcal{S}$) from the file splits on this node and emits intermediate $\langle nid, x\rangle$ pairs, where nid is the ID of a task. These intermediate $\langle nid, x\rangle$ pairs are then shuffled based on the keys (i.e., nid) and transferred to the reduce nodes (i.e., reducers), where intermediate $\langle nid, x\rangle$ pairs with the same keys are shuffled to the same reduce node. Each reduce node then receives a key-value pair in the form of $\langle nid, list(x)\rangle$, where $list(x)$ contains a list of records sharing the same nid (Line 3). After dividing the $list(x)$ into two record sets R and S, local set containment join algorithm is then applied on the reduce node to compute the join result (Lines 4-6). Note that there might be an extra job to summarize the join results from all reduce nodes.

Challenges. Following the theoretical analysis in [ASM+12], we consider three costs in a MapReduce iteration, which are map, reduce, and communication cost. The distribution strategy (i.e., Line 2 in Algorithm 15) should be able to handle load balance between the reduce nodes, since parallel computing is most important property of a MapReduce system. In the meanwhile, the communication cost should be as less as possible because the network band would become the bottleneck for a large cluster with many reduce nodes.

5.5.2 Distribution Scheme

In this section, we present our distribution scheme which is employed by the mappers to dispatch records in $\mathcal{R}$ and $\mathcal{S}$ to relevant reducers for parallel processing. We first discuss random distribution method which is most straightforward, followed by a prefix based method which is extended from the approach for processing set similarity join. We then propose a novel and efficient signature-based distribution method, which is computation load-aware and communication cost saving. For ease of explanation, we assume that there are N reduce tasks available for parallel

processing. Our goal is to devise a good distribution scheme for the mappers to dispatch records in $\mathcal{R}$ and $\mathcal{S}$ to the N reduce tasks, each of which is identified by an ID in the range of $[1, N]$. To measure the communication cost, we count the number of copies emitted to the reduce nodes for each record x, which is denoted by $C(x)$.

5.5.2-A Baseline Methods

Random based Distribution. A straightforward way to implement random distribution is as follows. For each $r \in \mathcal{R}$, we randomly dispatch r to one of the N reduce nodes, and for each $s \in \mathcal{S}$, we dispatch s to all N reduce nodes. It is evident this distribution method generates no duplication in the join results, and the communication costs are $C_{rand}(r) = 1$ and $C_{rand}(s) = N$, respectively. In the following, we present an advanced random distribution scheme, which can decrease the communication cost to $\sqrt{N}$, and at the same time, preserve the property of introducing no duplication in the join results.

We randomly divide records in $\mathcal{R}$ into $\sqrt{N}$ disjoint subsets. That is $\mathcal{R} = \cup_{1 \le i \le \sqrt{N}} \mathcal{R}_i$, where for $1 \le i \ne j \le \sqrt{N}$, $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$. Similarly, records in $\mathcal{S}$ are also randomly divided into $\sqrt{N}$ disjoint subsets where $\mathcal{S} = \cup_{1 \le i \le \sqrt{N}} \mathcal{S}_i$. Then, records in each pair $(\mathcal{R}_i, \mathcal{S}_j)$ with $1 \le i, j \le \sqrt{N}$, will be dispatched to uniquely one of the N reducers since there are N pairs in total. Apparently, the communication cost of random distribution is $\sqrt{N}$ for records in both $\mathcal{R}$ and $\mathcal{S}$; that is $C_{rand}(x) = \sqrt{N}$. Besides, there is no duplication in the reduce nodes. That is because, for any given record pair (r, s), it will only be dispatched to one reducer.

Prefix based Distribution. In [VCL10], efficient prefix-based distribution method is proposed for set similarity join. Given a record x and overlap similarity threshold T, we can compute the prefix of x, denoted by $Prefix(x)$, which

consists of the first $|x| - T + 1$ elements of x. Then, two records must share at least one common element in their prefixes if they are similar. Given a hash function h, record x is then dispatched to reduce node with ID $h(e_i)$ for each $e_i \in Prefix(x)$, where $h(e_i) = i \mod N + 1$ is widely used. This method is very efficient to process set similarity join because the prefix length is very limited. However, in our set containment join, the overlap similarity threshold T would be any value between 1 and $|x|$ since any subset of x forms a set containment relation with x. Therefore, to make the prefix based distribution strategy applicable for set containment join, we have to consider x itself as its prefix. Thus, we dispatch x to the reduce nodes corresponding to each element. Now, we estimate the communication cost as follows. For any reduce node, the probability that at least one element in x is dispatched to that node is $1 - (1 - \frac{1}{N})^{|x|}$. Therefore, the communication cost on N reduce nodes is

$$C_{pre} = N(1 - (1 - \frac{1}{N})^{|x|}). \tag{5.12}$$

Note that this method might introduce duplicates in the join results because a record pair might be dispatched to different reduce nodes.

5.5.2-B Our Signature-based Approach

Motivation. Even though the random distribution enjoys the nice property of load balance on all reduce nodes, the corresponding communication cost is high, which renders this method impractical for large scale datasets where we inevitably have to increase the value of N. Prefix-based method, on the other hand, is not able to handle dataset with large average record size, which is also verified by our experimental studies. According to Equation 5.12, the communication cost is approaching N when the record size (i.e., $|x|$) increases. The above limits of baseline methods motivate us to devise a new approach such that (i) the communication

cost is small; (*ii*) the workload on each reduce node is similar. By extending the idea of least significant element that used by *TT-Join*, we propose a signature-based distribution scheme in this section. Under the framework of this scheme, we devise efficient element domain partitioning algorithms in next section, which enable our signature-based approach to achieve the two goals (*i*) and (*ii*).

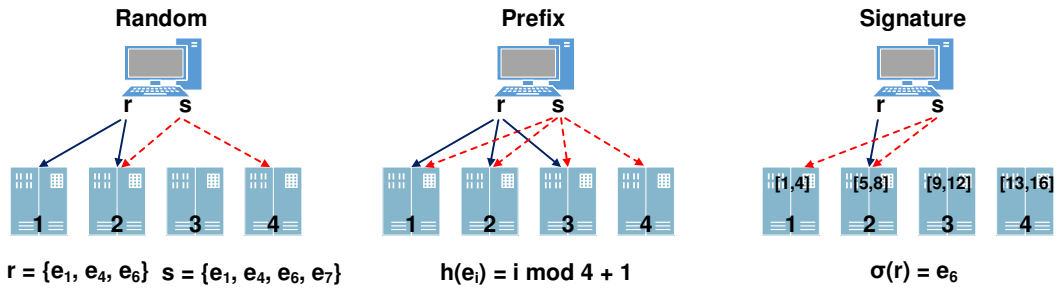


Figure 5.12: Example of different distribution schemes.

Our signature-based distribution scheme works as follows. We partition the ordered element domain $\mathcal{E} = \{e_1, e_2, ..., e_{|\mathcal{E}|}\}$ into N disjoint intervals, i.e., $[e_{l_1}, e_{h_1}]$, $[e_{l_2}, e_{h_2}]$, ..., $[e_{l_N}, e_{h_N}]$, where $l_1 = 1$, $h_N = |\mathcal{E}|$, and $l_{i+1} = h_i + 1$ for $1 \leq i \leq N - 1$. We assume that there is a one-to-one relationship between element intervals and reduce nodes. Now, given a record $r \in \mathcal{R}$, let σ be the signature (i.e., the least frequent element) of r. We find the interval where σ falls and dispatch r to the corresponding reduce node. For each record $s \in \mathcal{S}$, we dispatch s to all reduce nodes whose corresponding intervals cover at least one element of s.

Theorem 5.1. *Our signature-based distribution scheme is complete, i.e., it will not miss any join results.*

Proof: Given any result pair (r, s) (i.e., $r \subset s$), let σ be the signature of r (i.e., the least frequent element). Suppose σ falls into the i-th signature interval, which means that r will be dispatched to i-th reduce node. Since $r \subset s$, we have $\sigma \in s$.

Therefore, a copy of s will be dispatched to i-th reduce node as well. After the execution of local join algorithm in i-th reduce node, we get the pair (r, s). □

Compared to the baseline methods, a key advantage of signature-based method is that its communication cost is much lower. Specifically, $C_{sig}(r) = 1$ for $r \in \mathcal{R}$ since r is only transferred to exactly one reduce node. $C_{sig}(s) \in [1, \min(|s|, N)]$ for $s \in \mathcal{S}$ because s is only emitted to the reduce nodes that cover at least one element of s. Note that the expected value of $C_{sig}(s)$ depends on how we partition the element domain, which we will discuss in the following section. Besides, it is evident that signature-based approach does not generate duplicates in the join results because each record $r \in \mathcal{R}$ is emitted to exactly one reduce node. Also, signature-based scheme can pre-filter many unpromising candidate pairs by the signature during the distribution phase, and thus reduce the local join cost substantially.

Example 5.5. *Fig. 5.12 shows an example of all the distribution schemes. Suppose* $N = 4$, $\mathcal{E} = \{e_1, e_2, ..., e_{16}\}$, *and two records* $r = \{e_1, e_4, e_6\}$ *and* $s = \{e_1, e_4, e_6, e_7\}$. *For random scheme,* r *is dispatched to nodes* 1 *and* 2, *while* s *is distributed to nodes* 2 *and* 4. *Note that we apply a round-robin strategy to ensure that, for any record pair* r *and* s, *each of which will be dispatched to* $\sqrt{N}$ *reduce nodes, and they meet in exactly one reduce node. For prefix-based scheme, assuming* $h(e_i) = i \mod N + 1$, *and therefore* $h(e_1) = 2$, $h(e_4) = 1$, $h(e_6) = 3$, *and* $h(e_7) = 4$. *Thus,* r *is emitted to nodes* 1, 2, 3, *and* s *is emitted to all 4 nodes. For signature-based method, we assume that the element domain is evenly partitioned for ease of explanation. Therefore,* r *is dispatched to node* 2, *and* s *is dispatched to nodes* 1 *and* 2. *Clearly, our signature-based method introduce no replications, and has the lowest communication cost in this example.*

5.5.3 Load-Aware Partitioning

Our signature-based method makes use of a partition of the element domain. To find a good partition, a straightforward way is to partition $\mathcal{E}$ into N intervals evenly. However, as suggested by our experimental results, this method yields very poor performance. This is because many real-life data are skewed and therefore the records will be dispatched to the reduce nodes unevenly. In this section, we propose a judiciously-designed cost model which takes local join computation cost into consideration to guide the partition of element domain, such that the reduce nodes can take similar workload. We first devise a dynamic programming based optimal partition method, which bears a time and space complexity of $O(N|\mathcal{E}|^2)$. The high time and space complexity of this method is not suitable to dataset with large element domain size (i.e., $|\mathcal{E}|$). Consequently, we then resort to a heuristic partition algorithm, which is linear to the element domain size (i.e., $O(|\mathcal{E}|)$) for both time and space complexity.

Optimal Partition. In Section 5.4, given two collections $\mathcal{R}$ and $\mathcal{S}$, we conduct cost analysis for different join algorithms based on the element frequency distribution $P(e)$ and record length distribution $\theta(l)$. Now, we are interested in analysing the cost on each interval given a specific local join algorithm. First, we define an element partition instance as follows:

Definition 5.4. *An element partition instance, denoted by $\mathcal{P}(e_l, e_h, n)$, defines a partition which splits the element range $[e_l, e_h]$ into n disjoint intervals, where each interval is represented by $[e_{l_i}, e_{h_i}]$ for $i \in [1, n]$, assuming $l_1 = l$ and $h_n = h$.*

Note that a single interval $[e_l, e_h]$ is a partition instance with $n = 1$, i.e., $\mathcal{P}(e_l, e_h, 1)$.

Estimating $Cost(\mathcal{P}(e_l, e_h, 1))$. Given a single interval $[e_l, e_h]$, we now aim to esti-

mate the join cost on the corresponding reduce node against a given join algorithm. For ease of explanation, we use *RI-Join* algorithm to conduct the analysis because it has the simplest cost model as shown in Equation 5.4. Clearly, in order to estimate $Cost(\mathcal{P}(e_l, e_h, 1))$, we need to know the record collections received by the corresponding reduce node. By $\mathcal{R}(e_l, e_h)$ and $\mathcal{S}(e_l, e_h)$, we denote the record sets from $\mathcal{R}$ and $\mathcal{S}$, respectively. First, we consider the unit intervals; that is $e_l = e_h = e_i$. Based on the distribution scheme of our signature-based approach, $\mathcal{R}(e_i, e_i)$ contains all records with e_i as their least frequent element, where the size is shown in Equation 5.6. On the other hand, $\mathcal{S}(e_i, e_i)$ consists of the records which contain e_i, where the size is shown in Equation 5.3. According to Equation 5.4, we have:

$$
\begin{aligned}
Cost(\mathcal{P}(e_i, e_i, 1)) &= \sum_{r \in \mathcal{R}(e_i, e_i)} \sum_{e_j \in r} |I_{\mathcal{S}(e_i, e_i)}(e_j)| \\
&= |\mathcal{R}(e_i, e_i)| \times |r|_{avg} \times \sum_{j \leq i} P(e_j) |I_{\mathcal{S}(e_i, e_i)}(e_j)| \\
&\overset{(5.6)}{=} |\mathcal{R}| \times \sum_{l=1}^{|r|_{max}} \theta(l) \times l \times P(e_i) \times F(e_i)^{l-1} \\
&\times |r|_{avg} \times \sum_{j \leq i} P(e_j) |I_{\mathcal{S}(e_i, e_i)}(e_j)| \\
&\overset{(5.3)}{=} |\mathcal{R}| \times \sum_{l=1}^{|r|_{max}} \theta(l) \times l \times P(e_i) \times F(e_i)^{l-1} \\
&\times |r|_{avg} \times \sum_{j \leq i} P(e_j)^2 \times |\mathcal{S}| \times P(e_i) \times |s|_{avg}^2 \\
&= |\mathcal{R}| \times |\mathcal{S}| \times |r|_{avg}^2 \times |s|_{avg}^2 \times P(e_i)^2 \\
&\times F(e_i)^{l-1} \times \sum_{j \leq i} P(e_j)^2. \qquad (5.13)
\end{aligned}
$$

Since the inverted index size is linear to the data size received by the corresponding reduce node, it is implied that the join cost over an interval $[e_l, e_h]$ can be decomposed into the summation of the join cost over each unit interval $[e_i, e_i]$, where

$e_i \in [e_l, e_h]$. That is:

$$Cost(\mathcal{P}(e_l, e_h, 1)) = \sum_{l \le i \le h} Cost(\mathcal{P}(e_i, e_i, 1)). \tag{5.14}$$

Before presenting our partition algorithm, we first define a function f over $\mathcal{P}(e_l, e_h, n)$ to compute the maximal interval cost among n intervals as follows:

$$f(\mathcal{P}(e_l, e_h, n)) = \max_{\forall [e_{l_i}, e_{h_i}] \in \mathcal{P}(e_l, e_h, n)} Cost(\mathcal{P}(e_{l_i}, e_{h_i}, 1)). \tag{5.15}$$

Since the overall performance of the system is affected by the slowest reduce node, our goal is to find an instance $\mathcal{P}(e_1, e_{|\mathcal{E}|}, N)$ such that $f(\mathcal{P}(e_1, e_{|\mathcal{E}|}, N))$ is minimized. We denote the optimal solution that minimizes $f(\mathcal{P}(e_l, e_h, n))$ by $\mathcal{P}^*(e_l, e_h, n)$. Then, our goal is equivalent to finding $\mathcal{P}^*(e_1, e_{|\mathcal{E}|}, N)$. To solve this problem, we propose a dynamic programming algorithm based on the following key observation:

$$f(\mathcal{P}^*(e_l, e_h, n)) = \min_{l+n-2 \le i < h} \{\max\{f(\mathcal{P}^*(e_l, e_i, n-1)), f(\mathcal{P}^*(e_{i+1}, e_h, 1))\}\}. \tag{5.16}$$

This observation indicates that the optimal partition of size n can be computed by enumerating the rightmost boundary of the optimal partition of size $n-1$.

Algorithm 16 illustrates our dynamic programming based optimal partition method. Lines 1-2 computes the cost of single intervals based on Equation 5.14. Lines 3-5 iteratively computes the optimal partition for a range with n intervals based on Equation 5.16.

Time and space complexity. Based on Equation 5.14, it is easy to show that $Cost(\mathcal{P}(e_i, e_j, 1))$ can be computed in $O(1)$ time if we compute and store the value of $Cost(\mathcal{P}(e_1, e_j, 1))$ for $1 \le j \le |\mathcal{E}|$ in advance. Note that, this can be done in $O(|\mathcal{E}|)$ time. Thus, Lines 1-2 can be finished in $O(|\mathcal{E}|^2)$ time. The cost of Lines 3-5 is $O(N|\mathcal{E}|^2)$. Therefore, the time complexity of our optimal partition is $O(N|\mathcal{E}|^2)$.

Algorithm 16: **Optimal partition**

Input : $\mathcal{E} = \{e_1, e_2, ..., e_{|\mathcal{E}|}\}$: element domain to be partitioned; N: number of intervals

Output : $\mathcal{P}^*(e_1, e_{|\mathcal{E}|}, N)$: optimal partition

```
1 for 1 ≤ i ≤ j ≤ |E| do
2     Compute f(P*(e_i, e_j, 1)) by Cost(P(e_i, e_j, 1));
3 for 2 ≤ n ≤ N − 1 do
4     for n ≤ i ≤ |E| do
5         Compute f(P*(e_1, e_i, n)) based on Equation 5.16;
6 Compute f(P*(e_1, e_|E|, N)) based on Equation 5.16;
7 return P*(e_1, e_|E|, N)
```

Evidently, the space complexity is $O(N|\mathcal{E}|^2)$ as well because we have to maintain a three dimensional array for the cost of each instance $\mathcal{P}(e_l, e_h, n)$.

Heuristic Partition. The dynamic programming based method can find the optimal partition. Nevertheless, its high time and space complexities make it unsatisfactory for datasets with considerably large element domain size (e.g., $|\mathcal{E}|$ is in millions). This motivates us to devise heuristic method to partition the element domain with linear time and space complexities regarding the element domain size.

Based on the cost function shown in Equation 5.15 and the definition of optimal partition $\mathcal{P}^*(e_1, e_{|\mathcal{E}|}, N)$, it is evident that our goal is to find a partition such that the cost on each interval is as even as possible. Formally,

Theorem 5.2. *If there exists an element partition instance $\mathcal{P}(e_1, e_{|\mathcal{E}|}, N)$, such that each interval is with the same cost; that is, $\forall i, j \in [1, N], Cost(\mathcal{P}(e_{l_i}, e_{h_i}, 1)) = Cost(\mathcal{P}(e_{l_j}, e_{h_j}, 1))$, then $\mathcal{P}(e_1, e_{|\mathcal{E}|}, N)$ is an optimal partition.*

Proof: It is easy to show that $Cost(\mathcal{P}(e_l, e_h, 1))$ is monotonic with respect to

$[l, h]$; that is, if $l' \leq l$ and $h' \geq h$, then $Cost(\mathcal{P}(e_{l'}, e_{h'}, 1)) \geq Cost(\mathcal{P}(e_l, e_h, 1))$. For simplicity, assume we only have two intervals, which are $[e_{l_1}, e_{h_1}]$ and $[e_{l_2}, e_{h_2}]$, respectively. Suppose $Cost(\mathcal{P}(e_{l_1}, e_{h_1}, 1)) > Cost(\mathcal{P}(e_{l_2}, e_{h_2}, 1))$. Then, we can keep decreasing the value of h_1 and hence decreasing that of l_2 to reduce the value of $Cost(\mathcal{P}(e_{l_1}, e_{h_1}, 1))$ and increase that of $Cost(\mathcal{P}(e_{l_2}, e_{h_2}, 1))$, until we reach $Cost(\mathcal{P}(e_{l_1}, e_{h_1}, 1)) = Cost(\mathcal{P}(e_{l_2}, e_{h_2}, 1))$. At this point, we obtain the optimal partition since $Cost(\mathcal{P}(e_1, e_{|\mathcal{E}|}, N)) = \max\{Cost(\mathcal{P}(e_{l_1}, e_{h_1}, 1)), Cost(\mathcal{P}(e_{l_2}, e_{h_2}, 1))\}$. This proof generalizes to arbitrary N. □

The main idea of our heuristic method is that we sequentially generate the intervals one by one from e_1 to $e_{|\mathcal{E}|}$. In particular, to generate the i-th interval P_i, we fix the starting value l_i and keep increasing the ending value h_i until the cost of P_i is no less than the value $Cost(\mathcal{P}(e_{l_i}, e_{|\mathcal{E}|}, 1))/n$, where n is the number of intervals to partition for the rest of element range $\{e_{l_i}, ..., e_{|\mathcal{E}|}\}$. The intuition behind this method is that we use the cost of single interval on remaining element range as an estimation for each of the n intervals. Every time after splitting an interval off from it, we then re-estimate the cost for rest $n - 1$ intervals.

Algorithm 17 depicts the details of our heuristic method. Lines 4-11 iteratively generate the N intervals. Note that, before generating a new interval, we first compute the mean cost for the rest partitions, as shown in Line 2 and Line 10.

Time and space complexity. It is evident that the number of iterations (Lines 4-11) in Algorithm 17 is bounded by $|\mathcal{E}|$ since h is increased by 1 in each iteration. Meanwhile, the time complexity to compute $Cost(P(e_l, e_{|\mathcal{E}|}, 1))$ is $O(1)$ as shown before. Therefore, the time complexity of our heuristic partition method is $O(|\mathcal{E}|)$. On the other hand, the space complexity is $O(|\mathcal{E}|)$ as well because we have to store $|\mathcal{E}|$ values of $Cost(\mathcal{P}(e_1, e_j, 1))$ for $1 \leq j \leq |\mathcal{E}|$.

Algorithm 17: **Heuristic partition**

Input : $\mathcal{E} = \{e_1, e_2, ..., e_{|\mathcal{E}|}\}$: element domain to be partitioned; N: number of intervals

Output : $\mathcal{P}(e_1, e_{|\mathcal{E}|}, N)$: a partition instance

```
1  l ← h ← 1;
2  μ ← Cost(P(e_1, e_|E|, 1))/N;
3  i ← 1;
4  while i < N do
5      if Cost(P(e_l, e_h, 1)) < μ then
6          h ← h + 1;
7      else
8          Add interval [e_l, e_h] into P(e_1, e_|E|, N);
9          l ← h ← h + 1;
10         μ ← Cost(P(e_l, e_|E|, 1))/(N − i);
11         i ← i + 1;
12 Add interval [e_l, e_|E|] into P(e_1, e_|E|, N);
13 return P(e_1, e_|E|, N)
```

5.6 Experimental Studies

5.6.1 Centralized Evaluations

In this section, we empirically evaluate the performance of *TT-Join* in a single machine. All experiments are conducted on PCs with Intel Xeon 2x2.3GHz CPU and 128GB RAM running Debian Linux.

5.6.1-A Experimental Setup

Algorithms. In the experiment, we evaluate the following algorithms.

Dataset	Abbr	#Records	AvgLen	#Elements	z-value
Amazon [Data]	AMAZ	1,230,915	4.67	2,146,057	0.52
AOL [Datb]	AOL	10,054,183	3.01	3,873,246	0.68
BMS [BMGT15]	BMS	515,597	6.53	1,657	1.07
Bookcrossing [Datc]	BOOKC	340,523	3.38	105,278	0.6
Delicious [Datd]	DELIC	833,081	98.42	4,512,099	0.56
Discogs [Date]	DISCO	1,754,823	3.02	270,771	0.75
Enron [Datf]	ENRON	517,431	133.57	1,113,219	0.65
Flickr-L [BMGT15]	FLICKR-L	1,680,490	9.78	810,660	0.75
Flickr-S [LFHDB15]	FLICKR-S	3,546,729	5.36	618,970	0.63
Kosarak [BMGT15]	KOSRK	990,001	8.10	41,269	0.9
Lastfm [Datg]	LAST	1,084,620	4.07	992	0.51
Linux [Dath]	LINUX	337,509	1.78	42,045	0.81
Livejournal [Dati]	LIVEJ	3,201,203	35.08	7,489,073	0.62
Netflix [BMGT15]	NETFLIX	480,189	209.25	17,770	0.33
Orkut [LFHDB15]	ORKUT	1,853,285	57.16	15,293,693	0.13
Stack [Datj]	STACK	545,196	2.39	96,680	0.54
Sualize [Datk]	SUALZ	495,402	3.63	82,035	0.95
Teams [Datl]	TEAMS	901,166	1.52	34,461	0.39
Twitter [LFHDB15]	TWITTER	371,586	65.96	1,318	1.4
Webbase [LFHDB15]	WEBBS	168,707	463.64	15,146,263	0.04

Table 5.2: Characteristics of datasets

- **TT-Join**. Our approach proposed in Algorithm 14 in Section 5.4.3, where *kLFP-Tree* and a regular prefix tree are built on $\mathcal{R}$ and $\mathcal{S}$, respectively. By default, we set k=4 under all settings.

- **LIMIT**. *Intersection-oriented* algorithm proposed in [BMGT15] (Section 5.3.1). The optimized version of LIMIT (OPJ) is employed for performance evaluation.

- **PIEJoin**. *Intersection-oriented* algorithm proposed in [KRS$^+$16] (Section 5.3.1).

- **PRETTI+**. *Intersection-oriented* algorithm proposed in [LFHDB15] (Section 5.3.1).

- **PTSJ**. *Union-oriented* algorithm proposed in [LFHDB15] (Section 5.3.2).

- **DivideSkip**. *Adapted* algorithm proposed in [LLL08] (Section 5.3.3).
- **Adapt**. *Adapted* algorithm proposed in [WLF12] (Section 5.3.3).
- **FreqSet**. *Adapted* algorithm proposed in [AAK10] (Section 5.3.3).

Among the 8 algorithms, DivideSkip and Adapt are implemented in C++, where the source codes are obtained from the authors of [LLL08] and [WLF12] respectively. The rest 6 algorithms are all implemented in Java and the JVM maximum heap size is set to 32GB. For LIMIT and PIEJoin, we obtain the source codes from the authors of [KRS+16] since the source code of LIMIT is implemented in C++ and the authors of [KRS+16] re-implement LIMIT in Java. For PRETTI+ and PTSJ, we obtain the source code from the authors of [LFHDB15]. We implement FreqSet in Java, where FP-growth [HPY00] method is employed to compute the frequent sets. Among the 4 state-of-the-art algorithms, PIEJoin and PRETTI+ are parameter free. For LIMIT, we follow the same strategy adopted by authors of [KRS+16], where parameter tuning is carried out manually and individually for each dataset. Particularly, for datasets used in [KRS+16], we use the parameters tuned in [KRS+16], and for the rest datasets, we tune the best values individually. For PTSJ, we follow the strategy proposed by the authors, which show that a suitable signature length is between 16 and 32 times of the average length of records. In the experiments, we apply the middle value 24 for PTSJ. It is demonstrated in [KRS+16] that the frequency order of elements in records had a huge impact for LIMIT, PIEJoin, and PRETTI+. Therefore, we follow their empirical conclusion to apply infrequent sort order for LIMIT and PIEJoin, and frequent sort order for PRETTI+, which are stated optimal for the corresponding algorithms. Among the three adapted algorithms, DivideSkip and Adapt are parameter free. For FreqSet, the frequency threshold a is set to 1000.

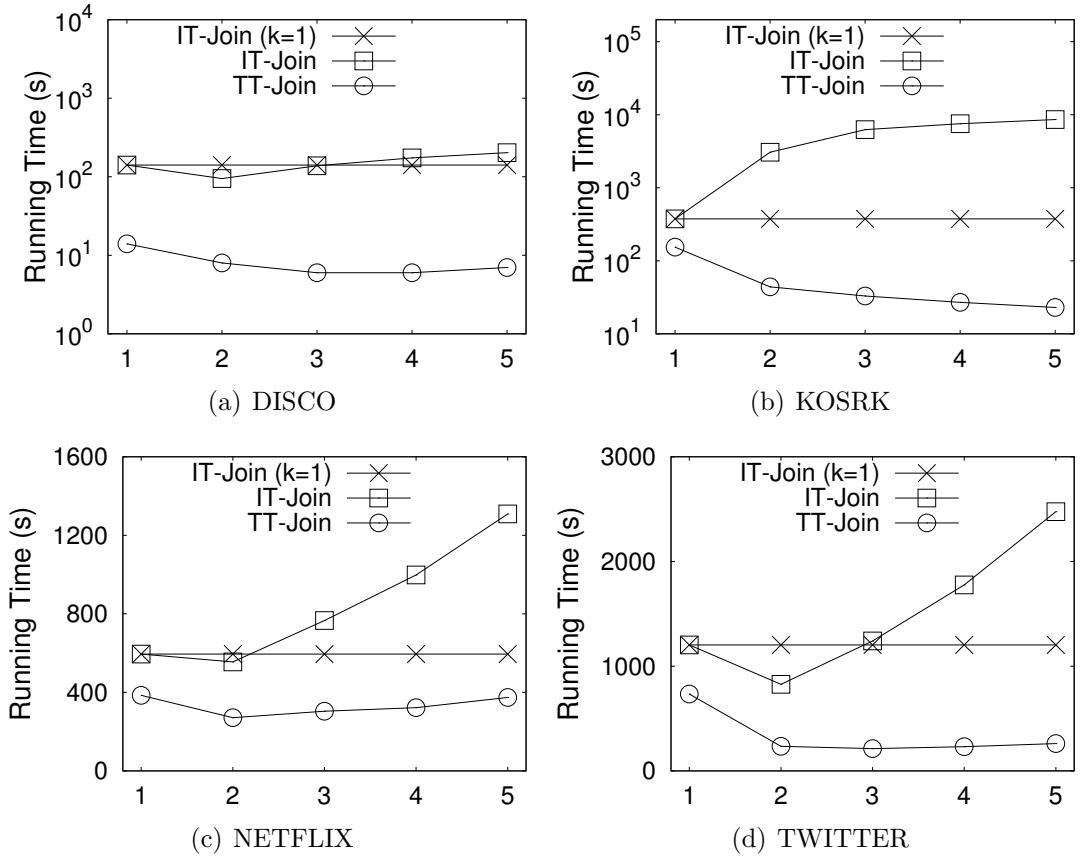


Figure 5.13: Effect of k on running time

5.6.1-B Performance Tuning

Datasets. We deploy 20 real-life datasets selected from different domains with various data properties. The detailed characteristics of the 20 datasets are shown in Table 5.2. For each dataset, we show the type of the dataset, what the record and element represent, the number of records in the dataset, the average record length, and the number of unique elements in the dataset. We also report the z-value (skewness) of the top 500 most frequent elements on each dataset by assuming that data follows Zipfian distribution. The datasets cover all datasets deployed in the state-of-the-art algorithms. In specific, Flickr-set, Orkut, Twitter, and Webbase are used in [LFHDB15] to evaluate PRETTI+ and PTSJ algorithms,

while BMS, Flickr-london, Kosarak, and Netflix are used in [BMGT15] to evaluate LIMIT algorithm. All of the eight datasets (with bold font in Table 5.2) are employed in [KRS+16] to evaluate PIEJoin. Same as the previous studies, we evaluate the self set containment join on the 20 datasets.

To better evaluate the impact of k value as well as the advantage of *kLFP-Tree* compared with the inverted index, we also implement an algorithm, namely **IT-Join**, which is an extension of *kIS-Join* algorithm where records in $\mathcal{S}$ are organized by a regular prefix tree. The traversal of the prefix tree is exactly the same as *TT-Join* (Algorithm 14) and the process of each visited node is based on *kIS-Join* algorithm in Section 5.4.2-C.

We choose four representative datasets from Table 5.2, including DISCO, KOSRK, NETFLIX, and TWITTER, which cover different types of dataset, various values of the average record size, as well as different z values. Note that, besides *TT-Join* and *IT-Join*, we also report the performance of *IT-Join* with $k = 1$ to see if the increase of k value in *TT-Join* and *IT-Join* algorithms got paid off.

Fig. 5.13 reports the running time of three algorithms on the above four datasets with k increasing from 1 to 5. It is observed that *IT-Join* can only benefit from small k values, such as 1 and 2, which implies that the performance gain from large k value can not pay-off the growth of filtering costs, i.e., the increase of the number of records explored on the inverted index. On the contrary, *TT-Join* performs much better than *IT-Join* when k increases. In particular, it can continuously benefit from the growth of k on KOSRK, while achieves the best performance when $k = 4$, $k = 2$ and $k = 3$ on DISCO, NETFLIX and TWITTER, respectively. This behaviour verifies our cost analysis in Section 5.4.2 and Section 5.4.3 that the overhead of a straightforward extension of inverted index is expensive and the gain may not be well paid off, while *TT-Join* can achieve a much better trade-off. Since

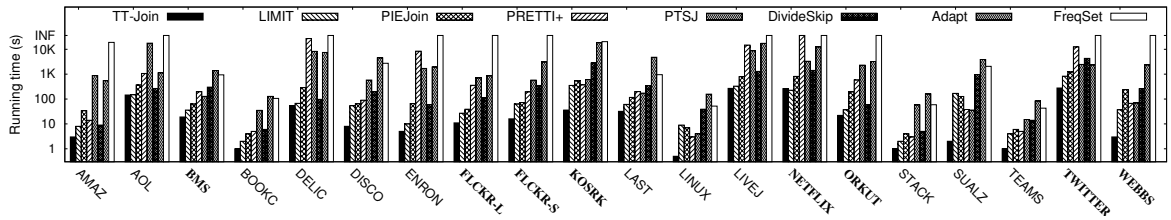


Figure 5.14: Processing Time

the performance of *IT-Join* is fully dominated by *TT-Join* under all datasets, it is excluded from the following experiments. By default, we set $k = 4$ for *TT-Join* algorithm for all datasets.

5.6.1-C Comparison with Existing Algorithms

In this subsection, we compare our *TT-Join* algorithm with four state-of-the-art algorithms LIMIT, PIEJoin, PRETTI+, and PSTJ as well as three modified algorithms DivideSkip, Adapt, and FreqSet on all 20 datasets. Recall that LIMIT, PIEJoin, and PRETTI+ are *intersection-oriented* methods and PSTJ is *union-oriented* method.

Processing Time. The experiment results in terms of processing time are reported in Fig. 5.14. Besides the set containment join time, the processing time also includes the index construction time because the indexes of all algorithms are generated on the fly. It is reported that our *TT-Join* algorithm outperforms all state-of-the-art algorithms on all datasets, except that it is slightly outperformed by LIMIT on NETFLIX. Among the existing algorithms, LIMIT achieves the best performance except on LINUX and SUALZ. The performance of PIEJoin is quite stable on all datasets, but it is always suppressed by LIMIT except on LINUX and SUALZ. The processing time of PRETTI+ and PTSJ is quite sensitive to the record length [LFHDB15]. It is observed that PRETTI+ favors datasets with small record size, such as AMAZ, DISCO, LINUX, SUALZ, and TEAMS. However,

PRETTI+ is extremely slow on datasets with relatively large record size, such as DELIC, ENRON, LIVEJ, NETFLIX, and TWITTER. It takes more than 10 hours on NETFLIX in which the average record size is 209. As reported in [LFHDB15], PTSJ, on the other hand, cannot efficiently handle datasets with small record size. For example, it takes hours for PTSJ to process AOL, while *TT-Join* spends less than 2 minutes. Generally, PTSJ has the worst overall performance. The reasons are two-fold. First, PTSJ is a bitmap-signature based method, which is data-independent and does not make use of the distribution of the elements. Second, it considers records in $\mathcal{S}$ individually, which means there is no computation share between records, even for identical records. The results show that DivideSkip significantly outperform other two adapted algorithms. Interestingly, DivideSkip even beats two state-of-the-art algorithms PRETTI+ and PTSJ on several datasets, such as AOL, DELIC, ENRON, FLICKR-L, LIVEJ, and ORKUT. The reason is that DivideSkip uses the same index strategy as PRETTI+, but DivideSkip can take advantage of the careful processing of long and short inverted lists in different ways. It is reported that Adapt and FreqSet are not competitive under all datasets. In particular, FreqSet fails to return results on half of the 20 datasets (we set allowed running time to be 10 hours).

As reported in Fig. 5.14, *TT-Join* has the best overall performance on 20 real-life datasets and significantly outperforms other competitors on the majority of the datasets. This is because *TT-Join* not only enhanced the nice properties of the *union-oriented* approach, e.g., exploited the skewness of the data and had less number of records explored, but also can directly validate a significant number of pairs, which are verification free. In particular, *TT-Join* beats other algorithms by at least around one order of magnitude on datasets with large z-values, such as DISCO, KOSRK, LINUX, SUALZ, and TWITTER. This is because *union-oriented*

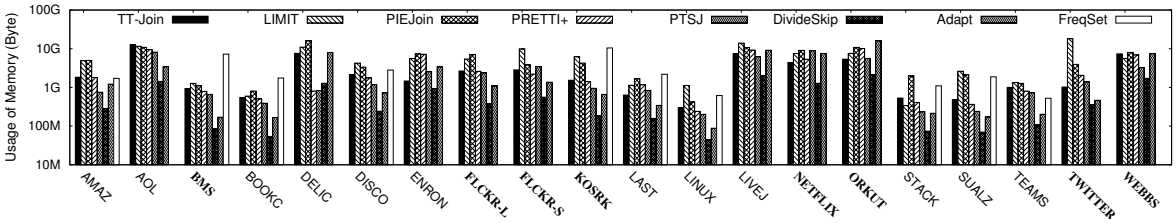
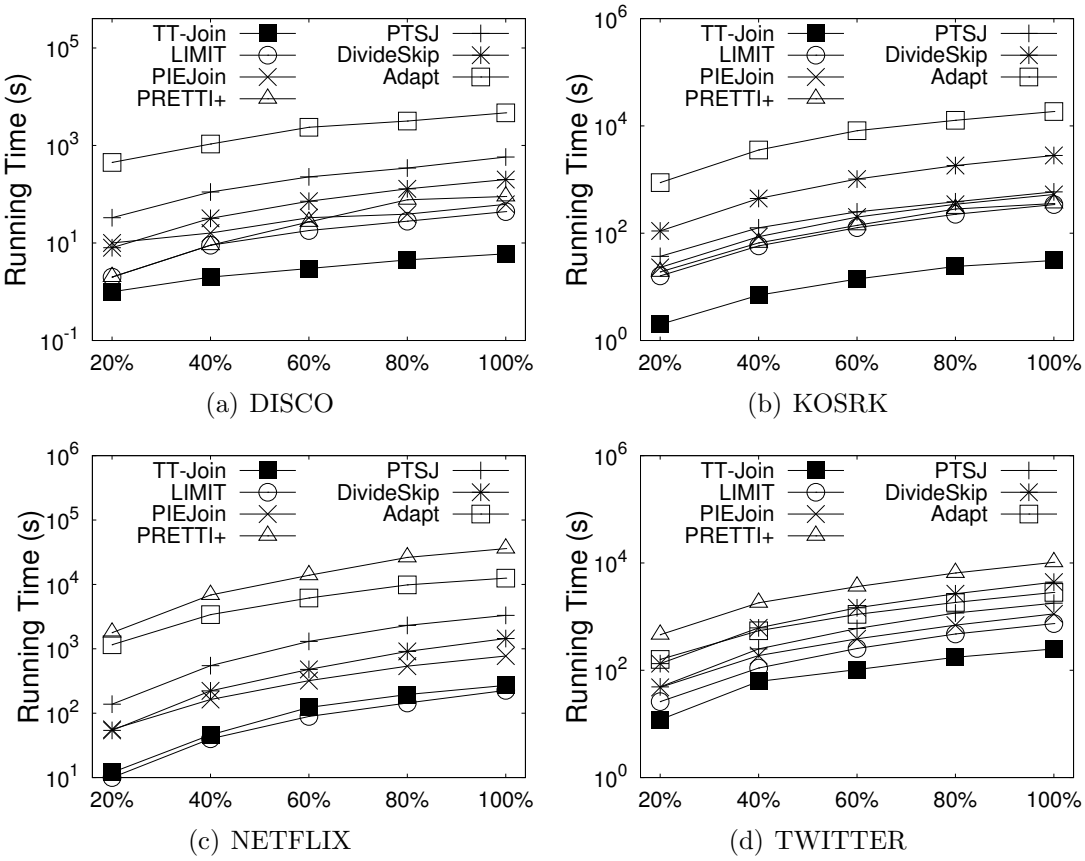


Figure 5.15: Memory Usage

TT-Join can effectively exploit the skewness of the data distribution. It is very interesting that *TT-Join* can also significantly outperform other competitors on ORKUT and WEBBS although they are not skewed, with z values 0.13 and 0.04, respectively. For instance, *TT-Join* outperforms other algorithms on WEBBS by one order of magnitude. We observe that there are a large number of distinct elements in ORKUT and WEBBS, and the average size of the records is large, which favour the least frequent element based signature technique. For some datasets with moderate or small z-value, such as AMAZ, LAST, and TEAMS, *TT-Join* can also achieve a superior performance, at least 2 times faster than the second ranked algorithm. The reason is that the *kLFP-Tree* enables us to increase the filtering capacity with small overhead and validate a significant number of join results without explicitly invoking the verification during the join processing. NETFLIX is the only dataset in which *TT-Join* is slightly outperformed by other competitors. We observe that it is not skewed ($z = 0.33$) and the number of distinct element ($|\mathcal{E}| = 17,770$) is small compared to the dataset size ($n = 480,189$ and $m = 209$), both of which are not in favour of *TT-Join*.

Memory Usage. Fig. 5.15 reports the memory usage of 8 algorithms. Same as [LFHDB15], the used memory is measured by the difference between the total memory and free memory of JVM after indexes are constructed for algorithms implemented in Java. For algorithms implemented in C++, we measure the maximum amount of used memory. It is observed that, DivideSkip consumes the small-



(a) DISCO (b) KOSRK

(c) NETFLIX (d) TWITTER

Figure 5.16: Vary number of records

est amount of memory under all datasets. Under most of the datasets, PTSJ and Adapt consume the second smallest amount of memory because PTSJ only builds Patricia trie index on $\mathcal{R}$ while Adapt only builds inverted list on $\mathcal{S}$. They are followed by *TT-Join* and PRETTI+. The memory usage of LIMIT and PIEJoin are similar and relatively larger than that of the other algorithms. This is because both of them use complicated index structures. Particularly, besides the prefix trees on both $\mathcal{R}$ and $\mathcal{S}$, PIEJoin also needs some auxiliary data structures to speed up the join processing.

Scalability Evaluation. In the last set of experiments, we evaluate the scalability of 7 algorithms on 4 representative datasets. FreqSet is excluded from the evaluation because it fails to give response within allowed time on most of the ex-

perimental settings. For each dataset, we randomly sample 20%, 40%, 60%, 80%, 100% of records from the original dataset, and conduct experiment on each sampled dataset. Fig. 5.16 shows that the running time of the algorithms grow steadily as the number of records increases on all datasets, and the performance ranks of the algorithms remain the same under most of the settings.

5.6.2 Distributed Evaluations

In this section, we evaluate the performance of our distributed set containment join algorithm. We implement algorithms with both Hadoop 2.7.2 and Spark 2.0.0. By default, we run our experiments on Spark. All experiments are conducted on a 10-node (one namenode/master and nine datanodes/slavers) cluster. Each node in the cluster is a Debian 6.0.10 server that has 3.4GHz Intel Xeon 8 cores CPU, 16GB RAM, and gigabit Ethernet interconnect. For Hadoop, we allocate a JVM heap space of 4MB for each mapper and reducer, and we allow at most 3 reducers running concurrently in each machine. We use the default block size 64MB in HDFS, and set the data replication factor of HDFS to 1. For Spark, we use standalone model and allocate 14GB memory for each executor. On each executor, we allocate at most 3 cores. In the experiments, we evaluate two metrics, namely, running time and communication/shuffle cost.

Algorithms. We compare the following three distribution schemes.

- **SIGNATURE.** Our signature-based distribution approach proposed in Section 5.5.2, where, by default, we apply the heuristic partition strategy proposed in Section 5.5.3.
- **RANDOM.** Advanced random distribution approach proposed in Section 5.5.2.

Datasets	TWEETS	MEMES
#Records	$74.6M$	$41.6M$
#Elements	$953K$	$2.4M$
Average record size	6.3	14
Size in GB	2.7	2.7

Table 5.3: Datasets statistics

- **PREFIX.** Prefix-based distribution approach proposed in Section 5.5.2.

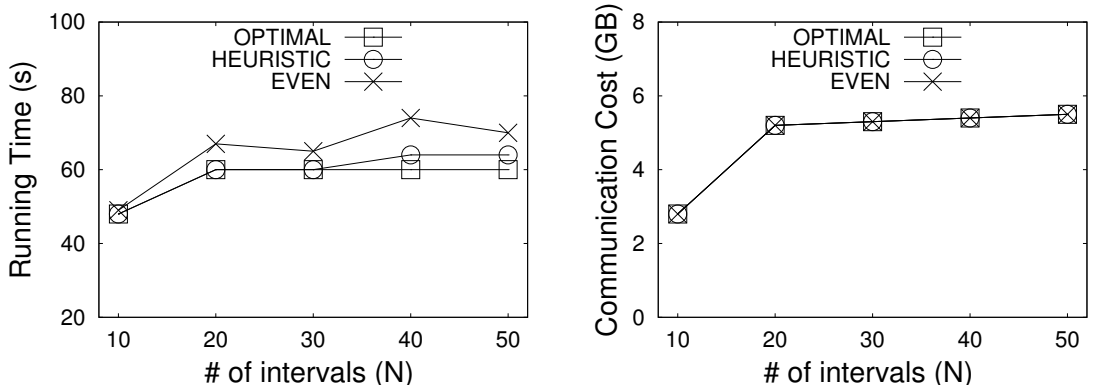
In the following experiments, we employ *TT-Join* as the local join algorithm, since it achieves the best overall performance as demonstrated in our centralized evaluation (Section 5.6.1). It is worth mentioning that PREFIX might introduce duplicates in the join results because a record pair might be dispatched to different reduce nodes. The running time in the experiments does not include the time for eliminating the duplication.

Datasets. Two datasets are deployed to evaluate the algorithms. TWEETS is a real-life dataset collected from Twitter, containing 74.6M tweets with an average number of terms being 6.3. MEMES is obtained from Memetracker [LBK09], which tracks the quotes and phrases that appear most frequently over time across the entire online news spectrum. The quotes and phrases used in this paper are collected in April 2009, where we consider each meme as a record. The statistics of two datasets are summarized in Table 5.3.

5.6.2-A Performance Tuning

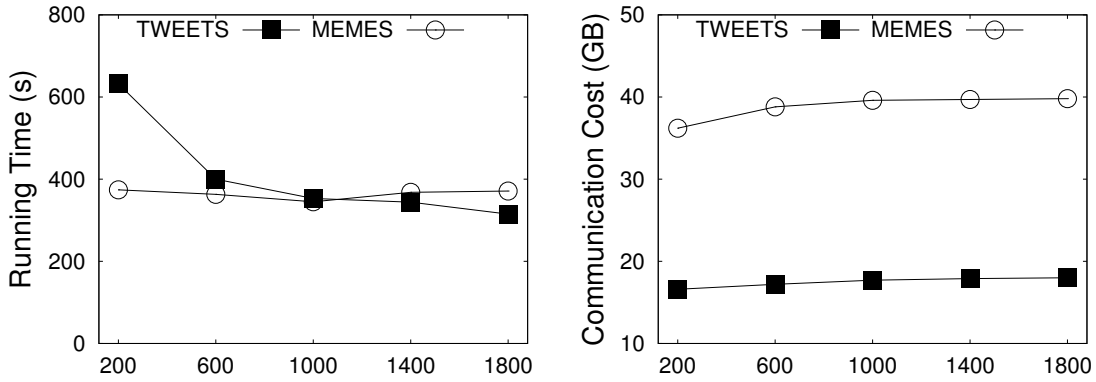
We start the experiments by tuning the performance of our approach SIGNATURE.

Compare partition strategies. In Section 5.5.3, we propose two element domain partition methods. Recall that the optimal partition has a time complexity of $O(N|\mathcal{E}|^2)$ which is impractical on datasets with large element domain size. We



(a) Running Time (b) Communication Cost

Figure 5.17: Compare partition strategies



(a) Running Time (b) Communication Cost

Figure 5.18: Vary number of intervals (N)

therefore choose NETFLIX in Table 5.2, where the element domain size is $17,770$, to conduct the experiments. Note that, we also implement the even partition strategy, where the element domain is evenly partitioned into N intervals. As reported in Fig. 5.17(a), our heuristic partition method, denoted by HEURISTIC, can achieve comparable performance as optimal partition method, denoted by OPTIMAL, in terms of running time, while even partition method, denoted by EVEN, is always beaten by other methods. That is because both OPTIMAL and HEURISTIC take the computation cost into consideration such that reduce nodes can take similar workload. It is interesting that all three methods have the similar communication

cost under all settings as shown in Fig. 5.17(b). The reason is that the communication cost is mainly determined by the distribution scheme and the number of intervals N. By default, we use HEURISTIC as the element domain partition method.

Effect of number of partitions. In this experiment, we evaluate the effect of number of intervals N to our approach SIGNATURE against the two datasets in Table 5.3, namely TWEETS and MEMES, where the number of intervals is varied from 200 to 1800. Fig. 5.18(a) reports that, SIGNATURE can continuously benefit from growth of N on TWEETS regarding the running time, while it runs the fastest when $N = 1000$ on MEMES. On the other hand, it is shown in Fig. 5.18(b) that the communication cost increases gradually when N increases. The reason is obvious because as N grows, we need to distribute records to more reduce nodes after map phase. Taking both running time and communication cost into consideration, we set $N = 1000$ for SIGNATURE for all datasets in the following experiments.

5.6.2-B Performance Evaluation

Scalability test. In this set of experiments, we evaluate the scalability of 3 approaches. For each datasets, we randomly sample 20%, 40%, 60%, 80%, 100% of records from the original dataset, and conduct experiment on each sampled dataset. It is worth mentioning that we also tune the best number of partitions for RANDOM and PREFIX. In particular, for RANDOM, the numbers of partitions are set to 12, 14, 16, 18, 20 on both $\mathcal{R}$ and $\mathcal{S}$ under the 5 settings, while the number of intervals for SIGNATURE and PREFIX are set to 200, 400, 600, 800, 1000, correspondingly.

It is reported in Fig. 5.19 that our approach SIGNATURE performs the best under all settings with respect to both running time and communication cost. We

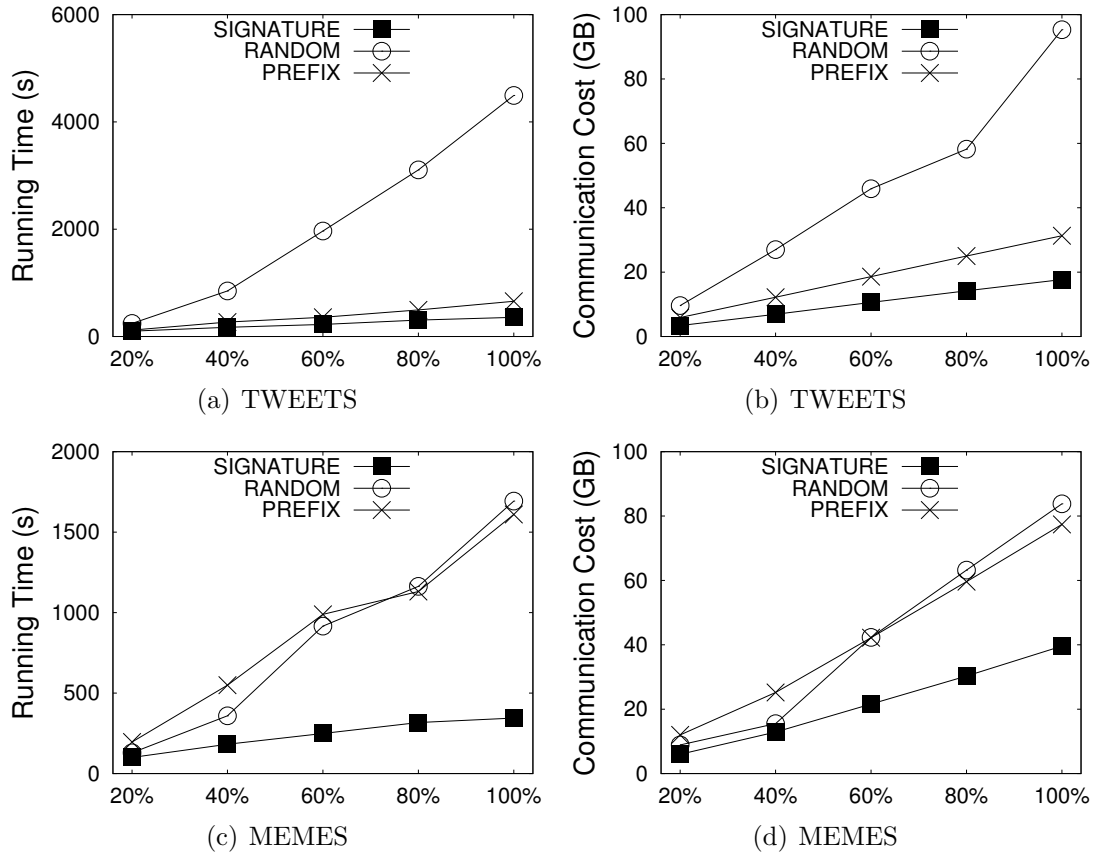
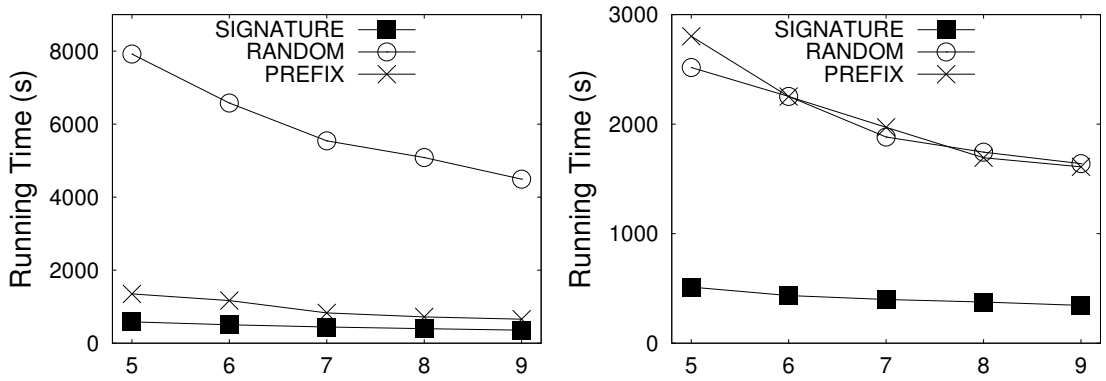


Figure 5.19: Vary number of records on Spark

also observe that SIGNATURE can scale much better than RANDOM does on TWEETS, and both RANDOM and PREFIX do on MEMES. Taking TWEETS for instance (Fig. 5.19(a)), when dataset size is 20%, SIGNATURE and RANDOM have comparable performance, and finish in 200 and 244 seconds, respectively. However, when dataset size grows to 100%, these numbers become 357 and 4493, which means that SIGNATURE is more than one order of magnitude faster than RANDOM. We have similar conclusion in terms of communication cost. Interestingly, it is also observed from Fig. 5.19 that PREFIX has very difference performance on the two datasets. In particular, PREFIX achieves much better performance on TWEETS (Fig. 5.19(a) and Fig. 5.19(b)) than it does on MEMES (Fig. 5.19(c) and Fig. 5.19(d)). Taking the running time for example, it is only slightly beaten

by SIGNATURE on TWEETS, while significantly outperformed by SIGNATURE on MEMES, where it is even marginally beaten by RANDOM when dataset size is less than 80%. The reason is that PREFIX is not suitable for dataset with large record size because it uses the entire record as prefix to build inverted index. Since the average record sizes of TWEETS and MEMES are 6.3 and 14, respectively, this explains why PREFIX performs better on TWEETS.

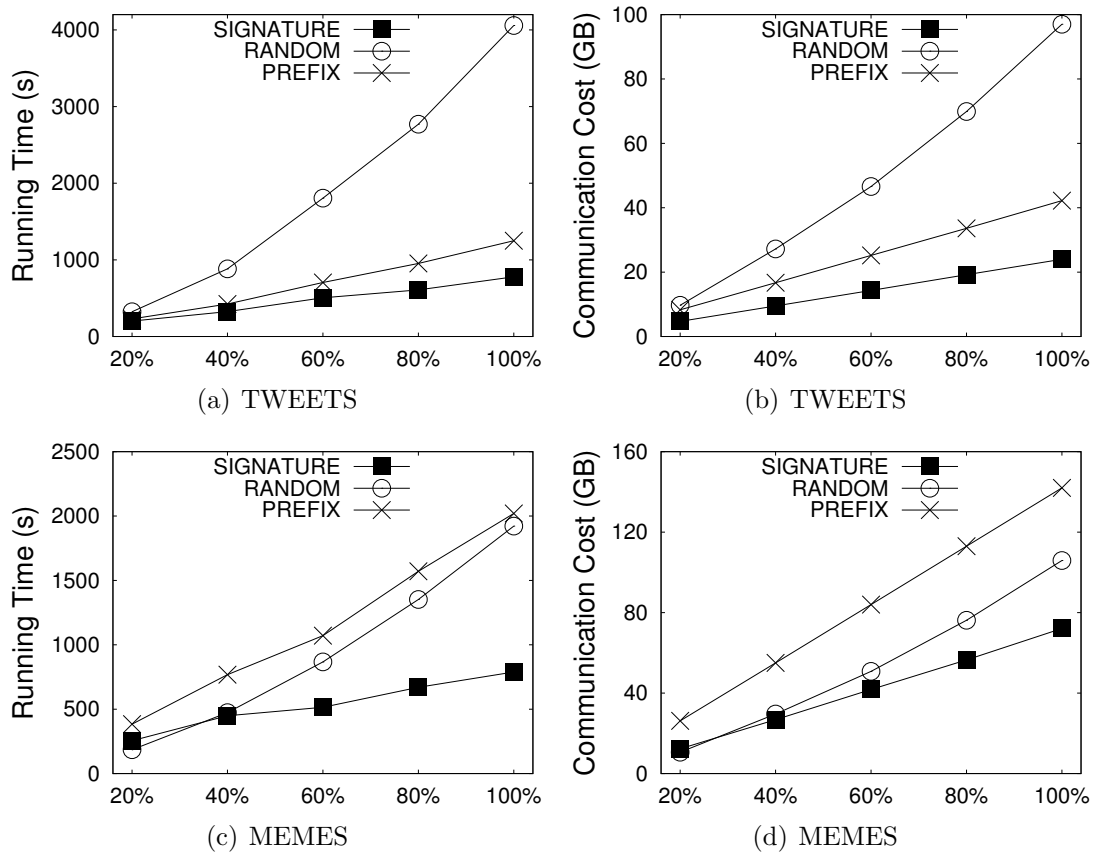


(a) TWEETS (b) MEMES

Figure 5.20: Vary number of slave nodes on Spark

Effect of number of slave nodes. In this experiment, we evaluate the effect of number of slave nodes by varying it from 5 to 9. Note that, we only show the running time for this experiment since the shuffle cost are the same for an algorithm under different number of slave nodes. The experiment results are shown in Fig. 5.20. When the number of slave nodes increases, the running time of all algorithms decreases steadily, and it drops more sharply when the number of slave nodes is small. Fig. 5.20(a) and Fig. 5.20(b) show that SIGNATURE is more than one order of magnitude faster than RANDOM on TWEETS, and 5 times faster than both RANDOM and PREFIX on MEMES under all settings, respectively.

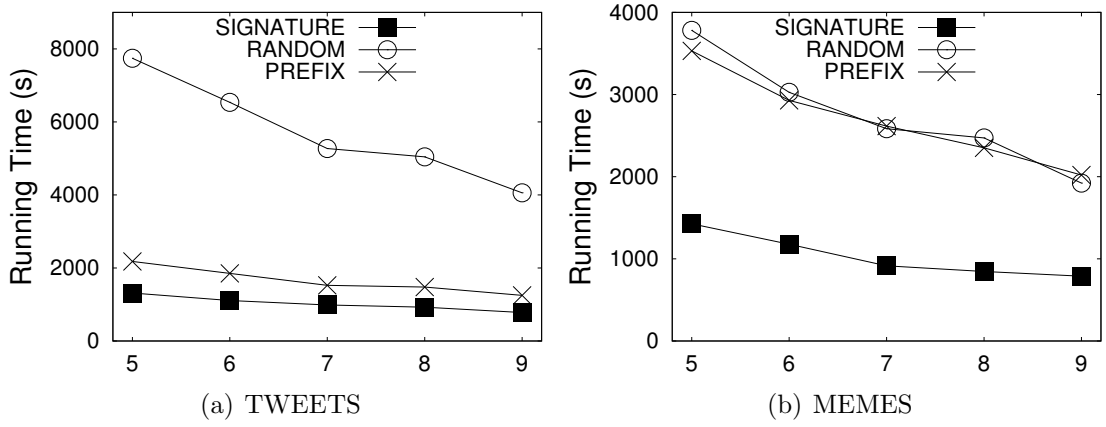
Evaluations on Hadoop. Fig. 5.21 and Fig. 5.22 report the performance of the algorithms running on Hadoop, which show similar behaviors as they do on Spark



(a) TWEETS (b) TWEETS

(c) MEMES (d) MEMES

Figure 5.21: Vary number of records on Hadoop



(a) TWEETS (b) MEMES

Figure 5.22: Vary number of slave nodes on Hadoop

shown in Fig. 5.19 and Fig. 5.20. That is, the processing time increases when the number of records increases, while decreases with the growth number of slave

nodes.

Interestingly, it is observed that the performance gap between SIGNATURE and other algorithms narrows on Hadoop. Take Fig. 5.19(a) and Fig. 5.21(a) for example. At the setting where the number of records is 100%, it takes 357 and 4493 seconds for SIGNATURE and RANDOM to finish on Spark. However, the numbers become 779 and 4058 on Hadoop. The reason is that SIGNATURE applies a two-stage strategy to compute the set containment join. In the first stage, we compute the element distribution, and based on that, we can find a good element domain partition. In the second stage, we distribute the records to reduce nodes according to the well partitioned element intervals and compute set containment join locally on each reduce node. On the other hand, both RANDOM and PREFIX only consist of one stage. Since Spark stores data in memory, it is more efficient for processing multiple-stage jobs. Therefore, SIGNATURE performs better on Spark than it does on Hadoop.

5.7 Conclusion

In this chapter, we study the problem of set containment join. Several in-memory set containment join algorithms have been developed in the literature. Based on the computing paradigms, we classify them into two categories, namely *intersection-oriented* and *union-oriented* methods. Through a comprehensive analysis, we show the advantages and limits of the algorithms in each category. Then we propose a new *union-oriented* method, namely *TT-Join*, which can take advantage of both *union-oriented* and *intersection-oriented* approaches. Extensive experiments on 20 real-life set-valued datasets from a variety of applications demonstrate the superior performance of *TT-Join* compared with the state-of-the-art techniques. Further-

more, to support large scale of datasets, we extend our techniques to distributed systems on top of MapReduce framework. With the help of careful designed load-aware distribution mechanisms, our distributed join algorithm can scale out well.

Chapter 6

Final Remarks

In this chapter, we summarize our research in this thesis and provide the possible future research directions. Specifically, Section 6.1 concludes the major contributions of this thesis. Section 6.2 introduces several possible orientations for future work.

6.1 Conclusions

As one of the most important information mining tasks, influence analysis has attracted tremendous attention in both industry and academic communities. In this thesis, we study three important influence related problems on three types of datasets, namely product and user preference data, spatial object data, and set-valued data. Below are the details.

Influence based Cost Optimization on User Preference. We advocate the problem of influence based cost optimization on user preference. By taking advantage of the k-level computing techniques and the monotonicity and convexity of the cost function, we reduce the solution space to a finite number of points. By exploiting the nice geometric properties of our problem in 2-dimensional spaces,

we develop an efficient traverse based algorithm for 2-dimensional spaces with time complexity $O(n)$. For general multi-dimensional spaces, we develop a space partition based algorithm. To further boost the computation performance, we devise a novel sampling based approach that can significantly outperform the exact algorithm with high accuracy.

Categorical Top-k Spatial Influence Query. We study the problem of categorical top-k spatial influence query, which is NP-hard regarding the number of types in the query. In the thesis, we follow the filtering-and-refinement framework base on R-tree style spatial indexes. Efficient and effective pruning techniques are developed to avoid the costly verification as much as possible. To tackle the computation hardness in the verification phase, we develop two algorithms: one is an efficient exact algorithm and the other is an approximate algorithm with performance guarantee.

Efficient Set Containment Join. For the set-valued data, we study the problem of set containment join which is fundamental. After a careful existing work review, we classify the existing solutions into two categories, namely *intersection-oriented* and *union-oriented* methods. Through a comprehensive analysis, we show the advantages and limits of the algorithms in each category. Then we propose a new *union-oriented* method, namely *TT-Join*, which can take advantage of both *union-oriented* and *intersection-oriented* approaches. Furthermore, to support large scale of datasets, we extend our techniques to distributed systems on top of MapReduce framework.

6.2 Directions for Future Work

In this section, we propose several possible directions for future work.

6.2.1 Cost Constraint based Influence Maximization

Given an influence target (i.e., k), our influence based cost optimization is to find a new object in the solution space such that its influence is at least k and the cost is minimized. Oppositely, it is also interesting to find a new object in the solution space such that its influence is maximized and the corresponding cost is under a given value. This query can help the manufacturing companies to maximize the profit of their manufacturing budget. Currently, there is no existing work studying cost constraint based influence maximization. Thus effective and efficient algorithms are needed to answer this query.

6.2.2 General Distance Cost Function based Spatial Influence Query

In Chapter 4, we have investigated the problem of categorical top-k spatial influence query, where the distance cost function is the sum of pairwise distance of objects in the functional unit. However, sometime it maybe inevitable to adopt other distance cost functions, such as diameter, minimum disk etc. Thus, it is of great importance to develop new pruning and indexing techniques to support different distance cost functions.

6.2.3 Spatio-textual Set Containment Join

In Chapter 5, we have studied the problem of set containment join for set-valued data. In other contexts, a record may also associate with location information. For instance, a checkin post on Foursquare contains both spatial and textual information. For such spatio-textual data, it is interesting to investigate the spatio-textual set containment join. Specifically, given a distance threshold D, we aim to find the

record pairs such that the distance between two records is less than D, and their textual sets form a set containment relation. To the best of our knowledge, there is no existing work investigating this problem. Thus it is necessary to devise efficient indexing and querying techniques to resolve this problem.

Bibliography

[AACS98] Pankaj K Agarwal, Boris Aronov, Timothy M Chan, and Micha Sharir. On levels in arrangements of lines, segments, planes, and triangles. *Discrete & Computational Geometry*, 19(3):315–331, 1998.

[AAK10] Parag Agrawal, Arvind Arasu, and Raghav Kaushik. On indexing error-tolerant set containment. In *SIGMOD*, pages 927–938, 2010.

[ADBMS98] Pankaj K Agarwal, Mark De Berg, Jirí Matousek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM journal on computing*, 27(3):654–667, 1998.

[ADV12] Anastasios Arvanitis, Antonios Deligiannakis, and Yannis Vassiliou. Efficient influence-based processing of market research queries. In *CIKM*, pages 1193–1202, 2012.

[AGK06] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *PVLDB*, pages 918–929, 2006.

[APV07] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Best position algorithms for top-k queries. In *PVLDB*, pages 495–506, 2007.

[ASM+12] Foto N Afrati, Anish Das Sarma, David Menestrina, Aditya

Parameswaran, and Jeffrey D Ullman. Fuzzy joins using mapreduce. In *ICDE*, pages 498–509, 2012.

[BGM12] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. Spatio-textual similarity joins. pages 1–12, 2012.

[BMGT15] Panagiotis Bouros, Nikos Mamoulis, Shen Ge, and Manolis Terrovitis. Set containment join revisited. *Knowledge and Information Systems*, pages 1–28, 2015.

[Bro97] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[CBC+00] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R Smith. The onion technique: indexing for linear optimization queries. In *SIGMOD*, pages 391–402, 2000.

[CC87] Yves Chabrillac and J-P Crouzeix. Continuity and differentiability properties of monotone real functions of several real variables. In *Nonlinear analysis and optimization*, pages 1–16. Springer, 1987.

[CCJO11] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. Collective spatial keyword queries. In *SIGMOD*, pages 689–700, 2011.

[CCJW13] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. Spatial keyword query processing: An experimental evaluation. In *PVLDB*, pages 217–228, 2013.

[CCKS07] Surajit Chaudhuri, Kenneth Church, Arnd Christian König, and Liying Sui. Heavy-tailed distributions and multi-keyword queries. In *SIGIR*, pages 663–670, 2007.

[CCSC16] Farhana M Choudhury, J Shane Culpepper, Timos Sellis, and Xin Cao. Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. In *PVLDB*, pages 456–467, 2016.

[CDbL+05] S. Cabello, J. M. Diaz-banez, S. Langerman, C. Seara, and I. Ventura. Reverse facility location problem. In *CCCG*, 2005.

[CJW09] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *PVLDB*, pages 337–348, 2009.

[CLH+15] Lei Chen, Xin Lin, Haibo Hu, Christian S Jensen, and Jianliang Xu. Answering why-not questions on spatial keyword top-k queries. In *ICDE*, pages 279–290, 2015.

[CLXJ17] Lei Chen, Yafei Li, Jianliang Xu, and Christian S Jensen. Direction-aware why-not spatial keyword top-k queries. In *ICDE*, pages 107–110, 2017.

[CLZZ11] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*, pages 577–588, 2011.

[CP07] Yun Chen and Jignesh M. Patel. Efficient evaluation of all-nearest-neighbor queries. In *ICDE*, pages 1056–1065, 2007.

[CP15] Aniket Chakrabarti and Srinivasan Parthasarathy. Sequential hypothesis tests for adaptive locality sensitive hashing. In *WWW*, pages 162–172, 2015.

[CPL16] Dong-Wan Choi, Jian Pei, and Xuemin Lin. Finding the minimum spatial keyword cover. In *ICDE*, pages 685–696, 2016.

[CS88] Kenneth L Clarkson and Peter W Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *SCG*, pages 12–17. ACM, 1988.

[CS89] Kenneth L Clarkson and Peter W Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(1):387–421, 1989.

[CSLZ14] Muhammad Aamir Cheema, Zhitao Shen, Xuemin Lin, and Wenjie Zhang. A unified framework for efficiently processing ranking related queries. In *EDBT*, pages 427–438, 2014.

[CXL+16] Lei Chen, Jianliang Xu, Xin Lin, Christian S Jensen, and Haibo Hu. Answering why-not spatial keyword top-k queries via keyword adaption. In *ICDE*, pages 697–708, 2016.

[Data] `http://liu.cs.uic.edu/download/data/`.

[Datb] `http://www.cim.mcgill.ca/~dudek/206/Logs/AOL-user-ct-collection`.

[Datc] `http://www.informatik.uni-freiburg.de/~cziegler/BX/`.

[Datd] `http://dai-labor.de/IRML/datasets`.

[Date] `http://www.discogs.com/`.

[Datf] `http://www.cs.cmu.edu/~enron`.

[Datg] `http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html`.

[Dath] `http://konect.uni-koblenz.de/networks/lkml_person-thread`.

[Dati] `http://socialnetworks.mpi-sws.org/data-imc2007.html`.

[Datj] http://www.clearbits.net/torrents/1881-dec-2011.

[Datk] `http://vi.sualize.us/`.

[Datl] `http://wiki.dbpedia.org/Downloads`.

[Dey98] Tamal K Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[DGKS07] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Nikos Sarkas. Ad-hoc top-k query answering for data streams. In *PVLDB*, pages 183–194, 2007.

[DLH+14] Dong Deng, Guoliang Li, Shuang Hao, Jiannan Wang, and Jianhua Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *ICDE*, pages 340–351, 2014.

[DLWF15] Dong Deng, Guoliang Li, He Wen, and Jianhua Feng. An efficient partition based method for exact set similarity joins. In *PVLDB*, pages 360–371, 2015.

[DZX05] Y. Du, D. Zhang, and T. Xia. The optimal-location qeury. In *SSTD*, pages 163–180, 2005.

[EOS86] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341–363, 1986.

[FHR08] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[FX10] Rudolf Fleischer and Xiaoming Xu. Computing minimum diameter color-spanning sets. In *International Workshop on Frontiers in Algorithmics*, pages 285–292. Springer, 2010.

[GCC15] Tao Guo, Xin Cao, and Gao Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418, 2015.

[GJ79] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[GLC+15] Yunjun Gao, Qing Liu, Gang Chen, Baihua Zheng, and Linlin Zhou. Answering why-not questions on reverse top-k queries. In *PVLDB*, pages 738–749, 2015.

[GMC+13] Shen Ge, Nikos Mamoulis, David Wai-lok Cheung, et al. Efficient all top-k computation-a unified solution for all top-k, reverse top-k and top-m influential queries. *TKDE*, 25(5):1015–1027, 2013.

[GMC+15] Shen Ge, Nikos Mamoulis, David WL Cheung, et al. Dominance relationship analysis with budget constraints. *Knowledge and information systems*, pages 409–440, 2015.

[GVDN15] Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Finding the most diverse products using preference queries. In *EDBT*, pages 205–216, 2015.

[GZCL09] Yunjun Gao, Baihua Zheng, Gencai Chen, and Qing Li. Optimal-location-selection query processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1162–1177, 2009.

[GZZC16] Yunjun Gao, Jingwen Zhao, Baihua Zheng, and Gang Chen. Efficient collective spatial keyword query processing on road networks. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):469–480, 2016.

[HKC+12] Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cédric Du Mouza, Michel Scholl, and Nicolas Travers. Subscription indexes for web syndication systems. In *EDBT*, pages 312–323, 2012.

[HKP01] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, pages 259–270, 2001.

[HLTF12] Weihuang Huang, Guoliang Li, Kian-Lee Tan, and Jianhua Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.

[HM97] Sven Helmer and Guido Moerkotte. Evaluation of main memory join algorithms for joins with set comparison predicates. In *VLDB*, pages 386–395, 1997.

[HP04] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB Journal*, 13(1):49–70, 2004.

[HPY00] Jiawei Han, Jian Pei, and Yan Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[HS99] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.

[HWQ+11] Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, and Zhen He. Top-k most influential locations selection. In *CIKM*, pages 2377–2380, 2011.

[IBS08] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[IF13] Ahmed Ibrahim and George HL Fletcher. Efficient processing of containment queries on nested sets. In *EDBT*, pages 227–238, 2013.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[JP05] Ravindranath Jampani and Vikram Pudi. Using prefix-trees for efficiently computing set joins. In *DASFAA*, pages 761–772, 2005.

[KLC14] Jia-Ling Koh, Chen-Yi Lin, and Arbee L Chen. Finding k most favorite products based on reverse top-t queries. *VLDB Journal*, 23(4):541–564, 2014.

[KM00] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[KRS+16] Anja Kunkel, Astrid Rheinländer, Christopher Schiefer, Sven Helmer, Panagiotis Bouros+3, and Ulf Leser. Piejoin: Towards parallel set containment joins. In *SSDBM*, page 11, 2016.

[LBK09] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *SIGKDD*, pages 497–506, 2009.

[LFHDB15] Yongming Luo, George HL Fletcher, Jan Hidders, and Paul De Bra. Efficient and scalable trie-based algorithms for computing set containment relations. In *ICDE*, pages 303–314, 2015.

[LFX12] Guoliang Li, Jianhua Feng, and Jing Xu. Desks: Direction-aware spatial keyword search. In *ICDE*, pages 474–485, 2012.

[LJ12] Hua Lu and Christian S Jensen. Upgrading uncompetitive products economically. In *ICDE*, pages 977–988, 2012.

[LKC13] Chen-Yi Lin, Jia-Ling Koh, and Arbee LP Chen. Determining k-most demanding products with maximum expected number of total customers. *TKDE*, 25(8):1732–1747, 2013.

[LLC11] Jiaheng Lu, Ying Lu, and Gao Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

[LLL08] Chen Li, Jiaheng Lu, and Yiming Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.

[LOTW06] Cuiping Li, Beng Chin Ooi, Anthony KH Tung, and Shan Wang. Dada: a data cube for dominant relationship analysis. In *SIGMOD*, pages 659–670, 2006.

[LWWF13] Cheng Long, Raymond Chi-Wing Wong, Ke Wang, and Ada Wai-Chee Fu. Collective spatial keyword queries: A distance owner-driven approach. In *SIGMOD*, pages 689–700, 2013.

[MA14] Willi Mann and Nikolaus Augsten. Pel: Position-enhanced length filter for set similarity joins. In *Grundlagen von Datenbanken*, pages 89–94, 2014.

[MAB16] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. An empirical evaluation of set similarity join techniques. In *PVLDB*, pages 636–647, 2016.

[Mam03] Nikos Mamoulis. Efficient processing of joins on set-valued attributes. In *SIGMOD*, pages 157–168, 2003.

[MBP06] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, pages 635–646, 2006.

[MF12] Ahmed Metwally and Christos Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. pages 704–715, 2012.

[MGM02] Sergey Melnik and Hector Garcia-Molina. Divide-and-conquer algorithm for computing set containment joins. In *EDBT*, pages 427–444, 2002.

[MGM03] Sergey Melnik and Hector Garcia Molina. Adaptive algorithms for set containment joins. *TODS*, 28(1):56–99, 2003.

[MP13] Kyriakos Mouratidis and HweeHwa Pang. Computing immutable regions for subspace top-k queries. In *PVLDB*, pages 73–84, 2013.

[Mul91] Ketan Mulmuley. On levels in arrangements and voronoi diagrams. *Discrete & Computational Geometry*, 6(1):307–338, 1991.

[MZP15] Kyriakos Mouratidis, Jilian Zhang, and HweeHwa Pang. Maximum rank query. pages 1554–1565, 2015.

[PS85] Franco P Preparata and Michael I Shamos. Computational geometry: an introduction. 1985.

[PW15] Peng Peng and Raymong Chi-Wing Wong. k-hit query: Top-k query with probabilistic utility function. In *SIGMOD*, pages 577–592, 2015.

[QGJ15] Li Qian, Jinyang Gao, and HV Jagadish. Learning user preferences by adaptive pairwise comparison. In *PVLDB*, pages 1322–1333, 2015.

[QZK+12] Jianzhong Qi, Rui Zhang, Lars Kulik, Dan Lin, and Yuan Xue. The min-dist location selection query. In *ICDE*, pages 366–377, 2012.

[RH11] Leonardo Andrade Ribeiro and Theo Härder. Generalizing prefix filtering to improve set similarity joins. *Information Systems*, 36(1):62–78, 2011.

[RJGJN11] João B Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases*, pages 205–222. Springer, 2011.

[RJVDN10] Joao B Rocha-Junior, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Efficient processing of top-k spatial preference queries. In *PVLDB*, pages 93–104, 2010.

[RKV95] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[RPNK00] Karthikeyan Ramasamy, Jignesh M Patel, Jeffrey F Naughton, and Raghav Kaushik. Set containment joins: The good, the bad and the ugly. In *VLDB*, pages 351–362, 2000.

[SL15] Anshumali Shrivastava and Ping Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, pages 981–991, 2015.

[SP12] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. pages 430–441, 2012.

[TBV+11] Manolis Terrovitis, Panagiotis Bouros, Panos Vassiliadis, Timos Sellis, and Nikos Mamoulis. Efficient answering of set containment queries for skewed item distributions. In *EDBT*, pages 225–236, 2011.

[THPP07] Yufei Tao, Vagelis Hristidis, Dimitris Papadias, and Yannis Papakonstantinou. Branch-and-bound processing of ranked queries. *Information Systems*, 32(3):424–445, 2007.

[TPVS06] Manolis Terrovitis, Spyros Passas, Panos Vassiliadis, and Timos Sellis. A combination of trie-trees and inverted files for the indexing of set-valued attributes. In *CIKM*, pages 728–737, 2006.

[TXP07] Yufei Tao, Xiaokui Xiao, and Jian Pei. Efficient skyline and top-k retrieval in subspaces. *IEEE Transactions on Knowledge and Data Engineering*, 19(8), 2007.

[VCL10] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD*, pages 495–506, 2010.

[VDKN10] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.

[VDNK10] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Identifying the most influential data objects with reverse top-k queries. In *PVLDB*, pages 364–372, 2010.

[VDNK13] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *SIGMOD*, pages 481–492, 2013.

[WCJ12] Dingming Wu, Gao Cong, and Christian S Jensen. A framework for efficient spatial web object retrieval. *VLDB Journal*, 21(6):797–822, 2012.

[WCZL15] Shenlu Wang, Muhammad Aamir Cheema, Ying Zhang, and Xuemin Lin. Selecting representative objects considering coverage and diversity. In *GeoRich*, pages 31–38, 2015.

[WLF12] Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*, pages 85–96, 2012.

[WÖY+09] Raymond Chi-Wing Wong, M Tamer Özsu, Philip S Yu, Ada Wai-Chee Fu, and Lian Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. pages 1126–1137, 2009.

[WQL+17] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. Leveraging set relations in exact set similarity join. In *PVLDB*, pages 925–936, 2017.

[WWI+09] Qian Wan, Raymond Chi-Wing Wong, Ihab F Ilyas, M Tamer Özsu, and Yu Peng. Creating competitive products. In *PVLDB*, pages 898–909, 2009.

[WWP11] Qian Wan, Raymond Chi-Wing Wong, and Yu Peng. Finding top-k profitable products. In *ICDE*, pages 1055–1066, 2011.

[WYJ13] Dingming Wu, Man Lung Yiu, and Christian S Jensen. Moving spatial keyword queries: Formulation, methods, and analysis. *ACM Transactions on Database Systems (TODS)*, 38(1):7, 2013.

[WYJC11] Dingming Wu, Man Lung Yiu, Christian S Jensen, and Gao Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[XLXJ17] Xike Xie, Xin Lin, Jianliang Xu, and Christian S Jensen. Reverse keyword-based location search. In *ICDE*, pages 375–386, 2017.

[XWLY08] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.

[XZKD05] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. On computing top-t most influential spatial sites. In *PVLDB*, pages 946–957, 2005.

[YAY12] Albert Yu, Pankaj K Agarwal, and Jun Yang. Processing a large number of continuous preference top-k queries. In *SIGMOD*, pages 397–408, 2012.

[YDMV07] Man Lung Yiu, Xiangyuan Dai, Nikos Mamoulis, and Michail Vaitis. Top-k spatial preference queries. In *ICDE*, pages 1076–1085, 2007.

[YGM94] Tak W Yan and Héctor García-Molina. Index structures for selective dissemination of information under the boolean model. *TODS*, 19(2):332–364, 1994.

[YYY+03] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, and Yuguo Chen. Efficient maintenance of materialized top-k views. In *ICDE*, pages 189–200, 2003.

[ZCM+09] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.

[ZCS+12] Xiaolong Zhang, Ke Chen, Lidan Shou, Gang Chen, Yuan Gao, and Kian-Lee Tan. Efficient processing of probabilistic set-containment queries on uncertain set-valued data. *Information Sciences*, 196:97–117, 2012.

[ZCT14] Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*, pages 355–364, 2014.

[ZGC+17] Jingwen Zhao, Yunjun Gao, Gang Chen, Christian S Jensen, Rui Chen, and Deng Cai. Reverse top-k geo-social keyword queries in road networks. In *ICDE*, pages 387–398, 2017.

[ZJK14] Zhao Zhang, Cheqing Jin, and Qiangqiang Kang. Reverse k-ranks query. In *PVLDB*, pages 785–796, 2014.

[ZLG11] Jiaqi Zhai, Yin Lou, and Johannes Gehrke. Atlas: a probabilistic algorithm for high dimensional similarity search. In *SIGMOD*, pages 997–1008, 2011.

[ZMP14] Jilian Zhang, Kyriakos Mouratidis, and HweeHwa Pang. Global immutable region computation. In *SIGMOD*, pages 1151–1162, 2014.

[ZMR98] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *TODS*, 23(4):453–490, 1998.

[ZNPM16] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. Lsh ensemble: Internet scale domain search. In *PVLDB*, pages 1185–1196, 2016.

[ZSZ+15] Kai Zheng, Han Su, Bolong Zheng, Shuo Shang, Jiajie Xu, Jiajun Liu, and Xiaofang Zhou. Interactive top-k spatial keyword queries. In *ICDE*, pages 423–434, 2015.

[ZZZ+14] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, and Xiaoyang Wang. Diversified spatial keyword search on road networks. In *EDBT*, pages 367–378, 2014.

[ZZZL12] Liming Zhan, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Finding top k most influential spatial facilities over uncertain objects. In *CIKM*, 2012.

[ZZZL13] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*, pages 1706–1721, 2013.

[ZZZL16] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1706–1721, 2016.