ISBN 978-82-326-1066-2 (printed version)
ISBN 978-82-326-1067-9 (electronic version)
ISSN 1503-8181

NTNU – Trondheim
Norwegian University of
Science and Technology

Orestis Gkorgkas

Doctoral Thesis

Doctoral theses at NTNU, 2015:205

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer andInformation Science

NTNU

Doctoral theses at NTNU, 2015:205

Orestis Gkorgkas

Database Content Exploration and Exploratory Analysis of User Queries

NTNU – Trondheim
Norwegian University of
Science and Technology

Orestis Gkorgkas

Database Content Exploration and Exploratory Analysis of User Queries

Thesis for the degree of Philosophiae Doctor

Trondheim, August 2015

Norwegian University of Science and Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

ISBN 978-82-326-1066-2 (printed version)
ISBN 978-82-326-1067-9 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2015:205

Printed by Skipnes Kommunikasjon as

Abstract

Content providers, such as enterprises and organizations who publish their content on the Internet, aim at making their content visible and easily accessible to the users. The vast amount of data contained in databases impedes their efforts, as users often find it challenging to navigate through the available data and find the items that best suit their needs. It is therefore necessary for content providers to motivate users to explore the available data and assist them in finding items that are interesting to them. State-of-the-art approaches such as top-k queries are not appropriate for data exploration as they require the users to be aware of the database structure and the content they are exploring.

In this thesis, we study the problem of enhancing the visibility of database content through exploratory search and analysis. We propose exploratory algorithms that return to the user a small number of results, which at the same time provide a wide overview of the available content. In addition, we present algorithms that identify items that are appealing to users and can be exploited for offering users an insight of the available items and motivating them to explore the database. In particular, the main contributions of the thesis are:

- We develop a framework for organizing and summarizing keyword search results based on their textual content and temporal data.
- We introduce a new type of query, the *eXploratory Top-k Join* (XTJ_k) query, which creates object combinations that are better suited to user preferences than single objects, and we present algorithms for the efficient processing of XTJ_k queries.
- We introduce the *continuous influential query*, which returns objects that are continuously attractive to a large number of users for long periods, and we present algorithms for the efficient retrieval of continuous influential objects.
- We model the diversity of database objects based on user preferences, and we propose efficient algorithms for selecting products that are attractive to a wide range of users with diverse preferences.
- We describe the *Best-terms* problem which is the problem of increasing the rank of a spatio-textual object through the enhancement of its textual description. We show that the problem is NP-hard and we present approximate algorithms that retrieve high quality results.

The proposed approaches have been evaluated through extensive experimental evaluation. The experiments were conducted using both synthetic and real datasets and demonstrate the efficiency of the proposed methods.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of philosophiae doctor.

The doctoral program has been conducted at the Department of Computer and Information Science, NTNU, Trondheim. The main supervisor was Kjetil Nørvåg. Jon Atle Gulla and George Tsatsaronis were assigned co-supervisors.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor Kjetil Nørvåg for his guidance and advice. Without his help, I would not have succeeded in completing my studies. I would like also like to thank my co-supervisors Jon Atle Gulla and George Tsatsaronis, who along with Kjetil gave me the opportunity to pursue my Ph.D. studies at NTNU.

I am particularly grateful to Akrivi Vlachou and Christos Doulkeridis. They have been extremely helpful in terms of developing new ideas, implementation and writing. Our co-operation has been a valuable learning experience for me. I am also thankful to Kostas Stefanidis for his help during the first years of my studies.

I would like to thank my colleagues for the nice working environment they have always created. I have particularly enjoyed our inspiring discussions and the relaxing breaks.

I am very grateful to Tassos for his friendship. He has always been present when I needed him. Thanks to Arne, Olga, Satu and George for their encouragement, throughout these years. Special thanks to Vasilis and Chrissa.

It is hard to express my gratitude for my parents Rania and George for their sacrifices and effort to provide me the best possible education. Finally, I would like to thank my brother Alexis and my uncle Nikos for their constant love, support and encouragement.

Contents

I Introduction and Background **3**

1 Introduction **5**

1.1 Motivation 5

1.1.1 Exploratory search 6

1.1.2 Exploratory analysis of user preferences 8

1.2 Research questions 9

1.3 Research method 10

1.4 Contribution 12

1.5 Publications 14

1.6 Additional publications 15

1.7 Structure of the thesis 16

2 Background **19**

2.1 Preference queries 19

2.1.1 Top-k queries on object combinations 23

2.1.2 Performance issues in top-k queries 23

2.2 Reverse top-k queries 25

2.3 Keyword search in structured data 28

2.3.1 Keyword search framework 28

II Exploratory Search **31**

3 Grouping and Summarizing Keyword Search Results **33**

3.1 Introduction 33

3.2 Related work 35

3.3 Framework 37

3.3.1 Keyword search 37
3.3.2 Keyword search result vector representation 38
3.3.3 Finding groups of keyword search results 38
3.3.4 Summaries of keyword query results 41
3.4 Evaluation 44
3.5 Conclusion 48

4 Exploratory Product Search Using Top-k Join Queries 49
4.1 Introduction 49
4.2 Related work 53
4.3 Problem definition 55
4.3.1 Object combinations 56
4.3.2 Ranking combinations 57
4.3.3 Theoretical properties 58
4.4 Pull-bound framework 59
4.4.1 XTJ_k framework 59
4.4.2 Modified HRJN* algorithm 60
4.5 Exploratory rank-join (XRJN) 63
4.5.1 Bounding scheme 63
4.5.2 Algorithm and correctness 66
4.5.3 Instance optimality 67
4.5.4 Lazy upper bound evaluation 69
4.5.5 Cost and complexity analysis 69
4.5.6 The XRJN* pulling strategy 70
4.6 Providing more results 72
4.7 Experimental evaluation 75
4.7.1 Pulling technique evaluation 77
4.7.2 Sensitivity analysis 78
4.8 Conclusion 86

III Exploratory Analysis of User Queries 87

5 Discovering Influential Data Objects over Time 89
5.1 Introduction 89
5.2 Related work 91
5.3 Preliminaries 93
5.3.1 Time-invariant case 93
5.3.2 Temporal model 94

5.4 Problem definition 95
5.5 Sequential interval scan 96
5.6 Algorithms with early termination 98
5.6.1 Early termination interval scan 100
5.6.2 Early termination best-first interval 102
5.7 Experimental evaluation 104
5.7.1 Performance of query processing 106
5.8 Conclusion 115

6 Finding the Most Diverse Products using Preference Queries 117
6.1 Introduction 118
6.2 Related work 120
6.3 Problem definition 122
6.4 Algorithms with centroid computation 125
6.4.1 Retrieving the candidate centroids 126
6.4.2 Diverse product selection algorithm 127
6.5 Selective top-k algorithm 129
6.5.1 Centroid approximation 129
6.5.2 Algorithmic description 130
6.6 Maintenance 133
6.7 Supporting other set-based similarity functions 134
6.8 Experimental evaluation 135
6.8.1 Quality of diversity 137
6.8.2 Scalability analysis 138
6.8.3 Sensitivity analysis 140
6.8.4 Results on real data 145
6.9 Conclusion 146

7 Maximizing Influence of Spatio-Textual Objects 149
7.1 Introduction 150
7.2 Related work 151
7.3 Preliminaries 153
7.3.1 Top-k spatial keyword queries 153
7.3.2 IR-tree 154
7.4 Problem definition 155
7.5 The best term first (BTF) algorithm 156
7.5.1 Algorithm description 156
7.5.2 Complexity analysis 158
7.6 Graph based term selection 158

7.6.1 Graph construction algorithm 159
7.6.2 Best subgraph selection algorithm 162
7.6.3 Complexity analysis 163
7.7 Experimental evaluation 164
7.8 Conclusion 169

IV Closing Remarks 171

8 Conclusion 173
8.1 Contributions 173
8.2 Future work 174
8.3 Outlook 176

References 179

Part I

Introduction and Background

In Chapter 1, we discuss the motivation, present the main contributions, and we give an outline of the remainder of the thesis. In Chapter 2, we present the related work that is the basis of this thesis.

Chapter 1

Introduction

Databases nowadays contain large volumes of data, and they are accessed by numerous users on a daily basis. The large volume of data poses challenges to both users accessing the databases and the companies or organizations managing them. Users on one hand, find it challenging to navigate through the data and explore the database content in order to find the information they are searching for. Enterprises on the other hand, need to make their content visible and accessible to the users and identify which objects in their database (e.g. products) have a significant impact on the user basis and use this information for promoting their products. In this thesis, we study the problem of enhancing the visibility of database content through exploration and query analysis techniques.

This chapter is organized as follows: In Section 1.1, we present the motivation behind our research, while Section 1.2 presents the research questions. In Section 1.3, we describe the methodology we followed during our research. In Section 1.4, we present the main contributions of this thesis and in Sections 1.5 and 1.6 we list the papers resulted from this Ph.D. study. Finally, in Section 1.7, we describe the structure of the thesis.

1.1 Motivation

Most companies today invest significant resources on making their content visible on the Web and enabling users to browse the offered products and services. Often, companies provide a plethora of different alternatives, which overwhelm the user and make it extremely difficult for her to find the products she is interested in. The original target of increasing the visibility of the available products is thus hindered by the abundance of products contained in the database. It is therefore necessary to develop data exploration techniques that will enable users to explore large databases and provide them with a wide, yet coherent overview of objects that fit their preferences. In that way, the available solutions will be more visible and easily accessible to the users.

In addition to helping users explore the database content, companies need to motivate users to visit and browse their database by presenting them attractive products. Analysis of user preferences can contribute in identifying products with large influence on the user base. The identification of such products can help a company plan its promotion and advertising strategy that will lead to the attraction of new customers.

In this thesis, we study the problem of enhancing the visibility of the objects contained in a database through exploratory search and analysis. To that direction, we propose search algorithms for providing users an informative overview of the database content. We also propose analysis techniques for identifying objects that are attractive to the users. In the following, we discuss the existing challenges in exploratory search and preference query analysis.

1.1.1 Exploratory search

When users are searching in a database, they are usually unaware of the exact database content. Quite commonly, they do not have a concrete idea of the objects' properties they are searching for but only certain preferences about them. Consequently, they need to *explore* the database contents to find the objects that best fit their preferences. For instance, if someone wishes to buy a laptop, one may have a general idea about the desired characteristics, but an exact description of the laptop is difficult to be strictly determined. Traditional database queries are hard constraint queries, which return either exact matches or nothing. In addition, hard constraint queries are in general quite complex, and in order to produce useful results, they require the user to be aware of the database content. They also often require the knowledge of a specific query language and the structure of the queried database [55]. Moreover, hard constraints are quite likely to produce very small or extremely large result sets that provide little insight of the available data. As a result, users are led to pose repeatedly new queries until they retrieve a satisfying result set [64]. Therefore, they are inappropriate for exploratory search as they pose significant difficulties to users searching the database.

Users experience frustration when they are not able to easily find the information they need. In an attempt to make database content easily accessible, several approaches have been proposed [23, 25, 34, 58, 79, 63, 125], which allow users to express their needs by posing preference queries using either sets of keywords or by indicating their interest on the objects' attributes they are searching for. The query result is typically a list of objects, usually ranked according to a function that measures the relevance or the performance of each object with respect to the query. This type of queries are known in literature as *preference* or *top-k* queries.

A key aspect that preference queries fail to capture in its entirety is the fact that users performing exploratory search are generally unfamiliar with the domain of the data they are searching, and they are possibly unclear about their wishes [116]. A flat list of results provides little insight to the user about the available information. In addition, the relaxation of constraints induced by preference queries introduces ambiguity to the search, as each keyword query could be associated with a large number of database queries. As a result queries can produce a large number of redundant results, which the user has to filter out. For example, the keyword query "Charlie Chaplin" on a movie database of an entertainment provider such as Netflix, could be referring to films where Charlie Chaplin participated as an actor or to movies and documentaries about Charlie Chaplin's life. Similar problems exist in cases where the number of the available choices are too many for the user to process. For example, if a user is interested in buying a laptop and she is interested in the price and certain upgradeable characteristics (e.g., battery or disk size), it is possible that a combination of an economic laptop with a number of accessory components is more preferable than a single expensive laptop. Presenting, however, a large number of possibly similar combinations between laptops and accessory products may confuse the user and prevent her from easily acquiring an overview of the available solutions.

It becomes apparent that preference queries transfer the problem of understanding the structure and the content of a data collection from the query formation to the query results. Most applications address such problems by either presenting to users a large number of diverse results, or by limiting the scope of search to a specific field, i.e., by implicitly adding constraints. In both cases, the user acquires little insight of the available data, either because she is confused by the overwhelming size of the data, or because the provided results offer limited alternatives. Producers and service providers are directly affected by this phenomenon, as the visibility of their products is compromised due to omission of products or due to the inclusion of redundant results. It is therefore essential for providers to balance between presenting a coherent set of results and offering a wide overview of the available alternatives, in order to increase the visibility of the offered products.

1.1.2 Exploratory analysis of user preferences

Apart from providing users a wide overview of the available products, it is vital for a company to be able to explore the connections between user preferences and products. Analysis of user queries and the database content can provide information regarding products that are attractive to users and the characteristics of the users who are attracted to them. Such information is of great importance in market analysis and product promotion. Two characteristic cases are the identification of products that are appealing to large number of users over long periods of time, and products that are attractive to a wide range of users.

User preferences vary over time and it is essential for a product manufacturer or a service provider to be able to identify products that are constantly highly ranked in the search results of large number of users. This information can be exploited for the more efficient promotion of products. For instance, different promotion tactics can be followed for products that the users are continuously interested in and for products that attract the users only for short periods of time or they do not attract users at all.

Marketing analysis can also benefit from identifying products or groups of products that are attractive to a wide range of users and potentially new customers. When a user is visiting the website of a company, she has little information about the available alternatives. A small number of possibly interesting products would give her an insight of the database content, intriguing and helping her to explore the database to find the desired item. Usually, only few items can be presented in the front page of a web-shop or in an advertisement campaign and these items should cover the preferences of different users in order to attract as many customers as possible. It is therefore essential in such settings to present a set of objects that can be appealing to a wide range of users with diverse preferences.

Analysis of user preferences can also be used by a company to improve its own visibility in the market. An increasing number of enterprises advertise their products through third party e-commerce platforms such as Booking.com[1]. These platforms serve as an intermediate between product manufacturers or service providers and customers. The advertised companies are required to provide descriptions of their products and services that are used by the catalog

[1]http://www.booking.com

service to return to the users results (e.g., companies or services) relevant to their preference queries. Quite often, the provided services and the user preferences are connected to a specific location. For instance, a user could be looking for a restaurant offering a specific type of food close to the city center or a hotel with a gym that is also near a beach. In such settings, the visibility of a provider is affected not only by the preferences of the user, but also by the services offered by competing providers in the area. Analysis of user preferences can provide information for services or products that a company should include in its description in order to increase its visibility among the users using the e-commerce platform.

1.2 Research questions

The primary research question studied in this thesis is the enhancement of the visibility of the objects stored in large databases through exploratory search and analysis. The research topic is quite broad and it is necessary to analyze it in more specific questions. The questions researched in this thesis are the following:

RQ 1 Keyword queries are inherently ambiguous and current keyword search techniques in structured data return a large number of results, which differ in content and structure. Users need to acquire a summarized overview of the content and the structure of the returned results. The provided summary should be significantly smaller than the result set, yet informative about the underlying content. The generation of the summarized result set should not induce a large processing overhead when compared to the generation of the original results.

How can we summarize keyword search results on structured data and provide users a coherent overview of the information relevant to their query?

RQ 2 A large number of items in a database can be combined with one another and produce combinations that fit better to user preferences than single items. However, users are often not able to get a wide overview of the available alternatives, due to the repeated appearance of the same items in numerous combinations. User should be presented a small number of combinations that cover at the same time a wide range of items and be guided to a combination fitting their preferences. As the number of the possible combinations can be extremely high, it is important to avoid generating similar combinations that are unlikely to be attractive to the user.

How can we efficiently explore a large collection of combinable objects?

RQ 3 User preferences are likely to change over time and the products that are attractive to users may be different for different time periods. As the number of products and the users accessing on-line product databases are constantly increasing, it is essential to develop efficient algorithms for discovering products that are constantly important for a large number of users.

How can we efficiently identify objects that are constantly highly ranked by a large number of users over a specific time-period?

RQ 4 Users can often be intimidated by the size of large data collections and be reluctant to explore a product collection of which they have little or no insight. Presenting attractive products to the users helps them explore the product database. Similar to RQ 3, efficiency and scalability are critical factors in the discovery of products that are attractive to a wide user base.

How can we efficiently identify groups of products that attract customers with diverse preferences?

RQ 5 The visibility of a product or a service is affected both by the terms used for describing them and the descriptions of competing products or services. Careful selection of features and terms in the description of an object is important for maximizing its visibility in a competitive environment.

How can a description of a service or product be enriched to improve its visibility?

1.3 Research method

Research methodologies vary significantly with respect to research area. In computer science, research follows mainly two paradigms: a) the behavioral science, and b) the design science [52]. Behavioral science, originating from natural science research, studies human and animal behavior. Under the scope of information technology, it involves developing of theories describing the interaction among people or organizations and information systems, and predicts or explains the impact of information technology on individuals or groups. Design science differs substantially from behavioral and in general, natural science. While natural science studies objects or phenomena in nature or society, design science studies artificial objects and phenomena designed to meet certain goals [107]. In other words, natural science tries to understand reality while design-science is a problem-solving paradigm trying to create artifacts that serve certain purposes [52, 78].

In this thesis, we consider the design science paradigm to be the most appropriate for the purposes of our research, as the construction and the evaluation of artifacts in the form of algorithms and prototypes is an inherent part of our study of the aforementioned research questions. The research cycle for each of the questions follows the design science cycle [107] and consists of the following steps:

- **Awareness of the problem.** The first step includes the identification of a problem, its description in natural language, and search for existing solutions through an extensive literature review.
- **Suggestion.** The problem is strictly defined and existing solutions are studied in order to analyze the weaknesses of current approaches and identify areas of improvement. Based on the problem properties and the analysis of the current approaches, possible improvements are being suggested, and a hypothesis is formulated regarding the performance effect of the proposed improvements.
- **Development.** Based on the suggested improvements, new algorithms are designed and implemented, which solve the problem as it was formulated in the Suggestion step.
- **Evaluation.** During the evaluation procedure, we compare the proposed algorithms against the state-of-the-art or the baseline in terms of efficiency and effectiveness. We use real datasets that are applicable to the respective topic. In cases where real data are unavailable or they are not adequate for a thorough evaluation, we employ synthetic datasets widely used among the research community.

  In several cases, the performance of the evaluated algorithms is affected in terms of effectiveness or efficiency by a large number of parameters.

In order to be able to study the effect of each parameter, we perform a series of experiments during the evaluation. In each experiment, one parameter is considered to be varying while all other parameters have constant values. The main parameters taken into consideration are *data cardinality*, i.e., the number of objects in a database, *weight cardinality*, i.e., the number of user preferences, *dimensionality*, i.e., the number of dimensions for objects and user preferences, *size of result set*, and *data distribution* in cases of synthetic datasets. The efficiency of the algorithms is evaluated with respect to I/O, i.e., the amount of data accessed from the disk, and *processing time* which is the total time needed to obtain a complete result for a query.

- **Transferring knowledge.** The results are published in good venues.

1.4 Contribution

In this section, we describe the main contributions in terms of the research questions posed in Section 1.2.

Creating summarized results of keyword search on structured data. Keyword-based search is the most popular way to explore data in databases due its simplicity and its intuitive nature. At the same time keyword queries are ambiguous and when they are applied on structured or semi-structured data the results may vary significantly in structure making it difficult for the user to understand the results. Towards this direction, we propose a framework for organizing the results into groups based on their content similarity and their temporal characteristics. To further assist users to comprehend the results and identify the results of their interest we provide summaries of the results as hints for query refinement. The summaries are expressed as a set of important terms in the result set. Our experimental results indicate that users are more satisfied when results are organized with respect to content and time than when results are simply ordered with respect to relevance.

Our framework answers to RQ 1 and is described in Chapter 3.

Exploratory top-k join queries. Quite often, many of the products that users are searching for, can be combined with accessories which extend the products' functionality or characteristics. To the user's discomfort, not all products and accessories are combinable with one another and not all combinations are appropriate for the user's needs. To address this issue, we model this problem as an exploratory search problem and we propose a new type of query, the *eXploratory Top-k Join* (XTJ_k) query, which returns to users a ranked list of product combinations that fit best the given preferences. To maximize the overview of the available solutions, we distinguish the products into main products (the products that the user is primarily interested in) and into accessories. Users are presented with a list of main products combined with the optimal set of accessories according to their preferences. Users have also the ability to explore alternative combinations for the products they are interested in. We introduce an efficient algorithm which exploits the properties of XTJ_k queries and allows the early termination of query processing. The algorithm's efficiency is proved theoretically and verified through a detailed experimental process.

XTJ_k queries answer to RQ 2 and are described in Chapter 4.

Identifying objects that are continuously visible over a time period. In applications such as market analysis, it is of great interest to product manufacturers to be able to identify the products with the highest impact. However, user preferences vary over time and monitoring of the products' popularity is essential for discovering products which are consistently visible to the users. We introduce the *continuity score*, which captures the impact of a product over a period of time and the respective *continuous influential query*, which selects products with high continuity score. We propose algorithms for efficient processing of continuous influential queries, which provide early termination and support incremental retrieval of a series of continuous influential objects. The algorithms' performance is evaluated through a detailed experimental study.

Continuous influential queries answer to RQ 3 and are described in Chapter 5.

Identifying sets of diverse objects using top-k queries. Discovering the most diverse objects among an object collection has numerous applications including market analysis and product promotion. Different from current approaches where the selection of diverse objects is based solely on their properties, we take into account the user preferences and we aim at finding the objects which attract the most diverse sets of users. Since the problem is NP-hard, we employ a greedy algorithm that considers the entire set of user preferences and a more efficient approximate algorithm which does not require the evaluation of all preferences. We compare the proposed algorithms in terms of efficiency and quality of the selected items through a thorough experimental evaluation.

Our approach computes efficiently a set of diverse objects based on user preferences and answers to RQ 4. It is described in detail in Chapter 6.

Feature-based object description optimization. Modern applications such as Google Maps and TripAdvisor enable users to formulate spatio-textual queries, using a set of keywords that describe their preferences and a desired location. The result set of a spatio-textual query consists of a set of objects, which are typically service provides such as hotels that are relevant to the user preferences and close to the desired location. In this context, the visibility of a service provider is determined by both the user preferences and the location of competing providers offering the same services. It becomes thus essential for a service provider to identify the keywords that will allow the described service to appear in the results of as many users as possible increasing this way the visibility of the provider.

We formulate the problem of increasing the visibility of a spatio-textual object by enriching its textual description and we prove that it is NP-hard. We present an approximate greedy algorithm of linear complexity with respect to the number of user preferences and we introduce a novel algorithm for keyword selection that improves the performance of query processing. We demonstrate the efficiency of the proposed algorithms through a detailed experimental evaluation using real data.

Our approach computes efficiently a set of terms which improves the visibility of a spatio-textual object and answers to the RQ 5. It is described in detail in Chapter 7.

1.5 Publications

In this section we present the scientific papers that resulted from this Ph.D. study. For each paper, we refer to the corresponding chapter in which the content of the paper is included.

Paper 1. A framework for grouping and summarizing keyword search results [41]. Orestis Gkorgkas, Kostas Stefanidis, and Kjetil Nørvåg. In *Pro-*

ceedings of ADBIS, 2013.
The content of this paper is included in Chapter 3.

Paper 2. Efficient processing of exploratory top-k joins [44]. Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. In *Proceedings of SSDBM*, 2014.
The content of this paper is included in Chapter 4.

Paper 3. Exploratory product search using top-k join queries [42]. Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. *Under submission.*
The content of this paper is included in Chapter 4.

Paper 4. Discovering influential data objects over time [43]. Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. In *Proceedings of SSTD*, 2013.
The content of this paper is included in Chapter 5.

Paper 5. Finding the most diverse products using preference queries [45]. Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. In *Proceedings of EDBT*, 2015.
The content of this paper is included in Chapter 6.

Paper 6. Maximizing Influence of spatio-textual objects through keyword selection [46]. Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. In *Proceedings of SSTD*, 2015.
The content of this paper is included in Chapter 7.

1.6 Additional publications

In the course of this Ph.D. the following papers were published, but they are not included in this thesis because they are not directly connected to its research topic.

- Efficient processing of top-k spatial keyword queries [91]. João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. In *Proceedings of SSTD*, 2011.

- Efficient community detection using power graph analysis [105]. George Tsatsaronis, Matthias Reimann, Iraklis Varlamis, Orestis Gkorgkas, and Kjetil Nørvåg. In *Proceedings of LSDS-IR '11*, 2011.

1.7 Structure of the thesis

This thesis is organized into 4 parts. The first part gives an introduction to our research topic and presents the technical background of the thesis. The second and third parts present our research on increasing the visibility of database content through exploratory search and exploratory analysis. In particular, the second part focuses on exploratory search and the third part on exploratory analysis. In the fourth part, we conclude the thesis and present directions for future research. In the following, a detailed outline of the thesis is presented.

Part I Introduction and Background

Chapter 1 presents the motivation of our research and the research questions studied. We also describe the methodology followed and the contributions of the thesis.

Chapter 2 describes the fundamentals regarding preference and keyword queries on which this thesis is based.

Part II Exploratory Search

Chapter 3 presents a framework for summarizing and grouping keyword search results in structured data based on their content and time-related information.

Chapter 4 studies the problem of creating combinations of objects which suit better to user preferences than single objects. It describes a new type of query, the eXploratory Top-k Join query, and presents algorithms for efficiently processing such types of queries.

Part III Exploratory Analysis

Chapter 5 describes the problem of identifying objects which are continuously influential over a specific time-period. It introduces the notion of the continuous influential objects and presents efficient algorithms for the identification of continuous influential objects.

Chapter 6 studies the problem of identifying objects which attract users with diverse preferences. We determine the diversity among objects based on the user queries on which they rank highly and we describe efficient and scalable algorithms for identifying a set of r diverse objects.

Chapter 7 focuses on the problem of increasing the visibility of a spatio-textual object by enhancing its textual description through term selection. We show that the problem is NP-hard and we propose two approximate algorithms.

Part IV Conclusions

Chapter 8 presents the conclusions of our research and possible directions for future research.

Chapter 2

Background

In this chapter, we describe the basic properties of preference and keyword queries, and discuss briefly common processing approaches.

The chapter is organized as follows. In Section 2.1, we describe preference queries in database systems, while in Section 2.2, we present the notion of reverse top-k queries. In Section 2.3, we describe approaches for processing keyword queries in structured data.

2.1 Preference queries

Preference is defined[1] as "*a greater interest in or desire for somebody/something than somebody/something else.*" In other words, preferences can be viewed as *soft constraints*, which require a best possible match and not an exact match as required by hard constraints posed by traditional database queries [64]. Preference queries have been extensively studied and many models have been suggested. The suggested models can be divided in two categories, the qualitative models and the quantitative ones [23]. Qualitative models typically represent preferences as binary relations. For instance, let us consider the example database of Table 2.1 and a user wishing to buy a car. Given the attribute *color* with domain $\text{dom(color)} = \{\text{black}, \text{white}, \text{red}, \text{blue}\}$, the preferences of a user could be expressed as a set of rules such as $P_1 = \{\text{black} > \text{white}, \text{white} > \text{blue}, \text{red} > \text{blue}\}$ denoting that black is more preferable than white, and red is more preferable than blue. Similarly, considering the domain $\text{dom(fuel)} =$

[1]Oxford Advanced Learner's Dictionary

Cars		
id	**color**	**fuel**
c_1	white	diesel
c_2	black	diesel
c_3	red	gasoline
c_4	blue	electricity

Table 2.1: Qualitative preference queries example

$\{\text{diesel}, \text{gasoline}, \text{electricity}\}$, the same user could have the set of preferences $P_2 = \{\text{electricity} > \text{diesel}, \text{diesel} > \text{gasoline}\}$. Preference rules can also be applied on other preference rules indicating their importance. A rule of the form $\mathrm{P}_1 > \mathrm{P}_2$ would indicate the importance of the fuel type over the color. According to those preferences, the best match for the user would be car c_4 while the second best would be c_2.

In quantitative models, object properties are typically represented by a set of real values. Each value represents the importance of a specific attribute or the performance of the described object regarding the attribute [3, 54]. A hotel database for instance, could contain a set of entries where each entry consists of a set of values describing among other features the overall quality, value for money, cleanliness, price per night and distance to the city center. Similarly, in an article database, a value could be connected to a subject (e.g., politics or sports) or a keyword (e.g., elections) and the weight would indicate the relation between the article and the respective subject or keyword. A possible way for a user to express her preferences, is to determine a set of constraints for each attribute of interest. A user might pose a query for all hotels with a price lower than 60 USD per night, an overall quality above 80%, and a distance to the center shorter than 1 km. The result set of such a query could have a size varying from being totally empty to being overwhelmingly large. The user would have to perform multiple queries in order to be able to have an insight about the data and to adjust the constraints in order to produce a satisfying result set [64]. Even with adjusted constraints, the result set could be hard to evaluate as there is no ordering imposed to the objects of the result set and consequently, the user has to process the entire result set in order to identify the object suited best to her needs.

Top-k queries [25, 34, 40] alleviate this problem by imposing a score value to each object and selecting the k objects with the best score. Instead of a set of hard constraints, each user query is associated with a vector of weights where each dimension of the vector represents the importance of a specific attribute to the user and each object is associated with a score based on that vector. In this way, the result set of a user query returns always a predefined number of results.

In more detail, let S be a set of objects where each object o has a set of real valued attributes $\mathcal{A} = \{a_1, \ldots, a_d\}$ and is associated with a point in $\mathbb{R}^d$ where the value $o[i]$ of dimension i represents the weight of o for the respective attribute a_i. Similarly let $w \in \mathbb{R}^d$ be a vector expressing a user preference where the value $w[i]$ of each dimension i represents the importance to the user of attribute a_i. Each object o is assigned a score using a scoring function $f_w : \mathbb{R}^d \rightarrow \mathbb{R}$, which is defined based on the user preference vector w. Often, the function employed is monotonically increasing, i.e., $f_w(o_1) \leq f_w(o_2)$ if it holds that $o_1[i] \leq o_2[i], 1 \leq i \leq d$. In other words, assuming lower scores are preferable, if o_1 is no worse than o_2 for all dimensions, the score of o_1 will be no worse than that of o_2. When for two objects o_1 and o_2 it holds that $o_1[i] \leq o_2[i], 1 \leq i \leq d$ and $o_1[j] < o_2[j]$ for at least one dimension j we say that o_1 *dominates* o_2. A direct consequence of the usage of a monotonic function is the fact that if o_1 dominates o_2 then $f_w(o_1) < f_w(o_2)$ for any preference w.

Given a preference vector w, a top-k query selects the k best objects in S according to their score as calculated by the scoring function f_w. A typical top-k query can be expressed in SQL as follows:

```
SELECT *
FROM HOTELS
WHERE HOTELS.CITY='Barcelona'
RANK BY 0.4*price+0.6*distance_to_center ASC
STOP AFTER k;
```

In the above example we have a 2-dimensional query where the user preference vector is equal to $w = (0.4, 0.6)$ and the scoring function is the linear function $f_w = \sum_{i=1}^{2} w[i]o[i]$. More formally, a top-$k$ query can be defined as follows.

Definition 2.1. ***Top-k query [110].*** *Given a set of objects S, a positive integer k and a user-defined weighting vector w, the result set $TOP_k(w)$ of a top-k query is a ranked set of objects such that $TOP_k(w) \subseteq S$, $|TOP_k(w)| = k$ and $\forall o_1, o_2 : o_1 \in TOP_k(w)$, $o_2 \in S - TOP_k(w)$ it holds that $f_w(o_1) \leq f_w(o_2)$.*

Returning to the aforementioned example, let us consider a set of hotels S where each hotel is described by its price and distance to the center. The hotels

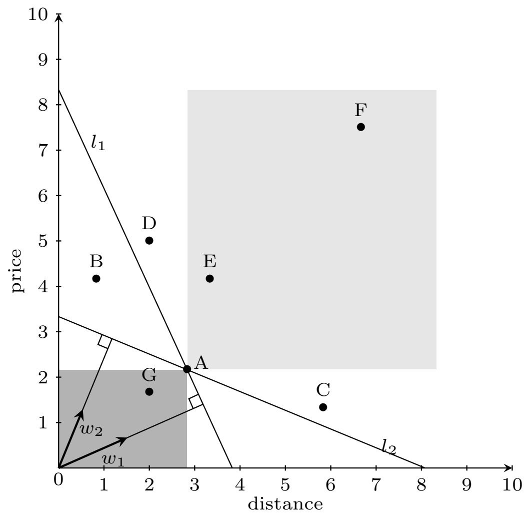


Figure 2.1: Top-k example: lower is better

can be represented by a set of points in $\mathbb{R}^2$ and they are shown in Figure 2.1. The light gray area indicates the area dominated by hotel A, while the dark gray area indicates the area dominating A. For any monotonic function, any object in the light gray area has a worse (higher) score than A, while any object in the dark area has a better (lower) score than A. Any other object may have either a better or worse score depending on the scoring function defined by a preference vector w.

Given a hotel o_1, a scoring function f and a preference vector w, the equation $f_w(o) = f_w(o_1)$ defines a locus in $\mathbb{R}^2$ of all points o which have a score equal to that of o_1. Considering the linear scoring function $f_w = \sum_{i=1}^{2} w[i]o[i]$, lines l_1 and l_2 in Figure 2.1 illustrate the loci defined by the equations $f_{w_1}(o) = f_{w_1}(A)$ and $f_{w_2}(o) = f_{w_2}(A)$ of two preference vectors w_1 and w_2 respectively. Each locus divides the space in two half-spaces. Any object in the lower half-space has a score lower than A while any object in the upper half space has a score higher than A. Considering lower scores to be preferable, we can easily see that A is in the $TOP_3(w_1)$ set as it has a worse (larger) score than G and B but better than any other point, and for w_2 it belongs in the $TOP_2(w_2)$ because its score is the lowest of all points with the exception of G.

2.1.1 Top-k queries on object combinations

It is quite common for user queries to involve not only single objects but combinations of objects as well. Travelers who explore a travel agency looking for cheap combinations of flights and hotels for a vacation trip are a usual case of users looking for a combination of services. This type of query is usually called *rank-join* query in literature [37, 57, 96] and an SQL-like example of a rank-join query is the following:

```
SELECT *
FROM HOTELS, FLIGHTS
WHERE HOTELS.CITY=FLIGHTS.DEST_CITY
RANK BY 0.33*HOTELS.price+0.33*FLIGHTS.price+0.33*HOTELS.rating
STOP AFTER k;
```

Considering a relational database, a rank join query is defined by a joining condition over a set of relations and a ranking function f_w. The result set of a rank-join query is a set of k tuples where each tuple τ of the result set consists of exactly one tuple from each relation participating in the join defined by the query.

Definition 2.2. ***Rank-join query [57].*** *Given a join of n relations $\mathcal{R} = R_1 \bowtie_{\sigma_1} R_2 \dots \bowtie_{\sigma_{n-1}} R_n$ a scoring function f_w and an integer k, the result set of rank-join query is a set of tuples RJ such that $|RJ| \leq k$ and for each tuple $\tau \in RJ$ it holds that $\tau = t_1 \bowtie_{\sigma_1} t_2 \dots \bowtie_{\sigma_{n-1}} t_n$, $\tau \in \mathcal{R}$, $\forall \tau_1, \tau_2 : \tau_1 \in RJ, \tau_2 \in \mathcal{R} - RJ$ it holds that $f_w(\tau_1) \leq f_w(\tau_2)$.*

2.1.2 Performance issues in top-k queries

Top-k queries are used widely by databases that are visited by numerous users [57, 58, 79]. Therefore, modern systems have to satisfy strict performance and scalability requirements. Each system has different requirements but in general, the requirements for a top-k search system are the following:

- Low I/O cost: During a top-k search the amount of data read from the disk should be minimized.
- Low response time: The response time of a query should be minimized. The response time includes the I/O cost and the CPU processing time.

- Early return of the first result: The first results of a top-k query should be returned as fast as possible. Early returns on top-k search allow users to examine the first results while the rest of the results are being calculated and they also allow the efficient pipelining of multiple top-k queries.

The main challenge in top-k search is the fact that each top-k query determines a different ordering of the ranked objects. A naive way to answer a top-k query would be to calculate the score of all objects and choose the best k objects according to their score. Such an approach induces prohibitive cost both in terms of I/O and CPU processing time, as all objects have to be accessed and evaluated. Moreover, the first results cannot be returned until all objects have been evaluated. There have been several approaches for answering top-k queries that focus on minimizing the number of accessed objects and the early return of the first results [40, 58, 77]. Most approaches identify the k best objects according to the scoring function by using either a set of ranked lists over the queried set of objects [40, 56, 122] or an R-tree-like multi-dimensional index [12, 103].

Algorithm 1 describes the generic framework followed by most approaches in literature using ranked lists. The algorithm takes as input a set of objects S, a scoring function f_w defined by the preference vector w, and an integer k denoting the number of desired results. It also requires as input a set of ranked lists of the objects in S. Each list contains all objects in S ranked according to a specific attribute a_i or according to a preference vector w. In each iteration of the loop, the algorithm initially reads an object o from a list L_i and updates its score $f_w(o)$ according to the function f_w. In the following steps, it updates the TOP_k set of the k best objects retrieved, and calculates and maximum (worst) score needed for any object to be included in the TOP_k set and the minimum (best) possible score f_{min} of any seen or unseen object not in the TOP_k set. When f_{min} becomes greater or equal to $thres$ (line 11), the algorithm can safely stop as it is certain that any unseen object cannot be included in the TOP_k set.

The performance of the algorithms following this generic framework depends on two factors. The first factor is the number of accessed objects until the termination condition is satisfied and the second factor is the computational complexity of the algorithm used for the estimation of the f_{min} value. As a consequence, a significant part of the research in top-k queries is focused on finding an efficient f_{min} estimation algorithm which will minimize the number of accessed objects (access depth) and will also induce a low computational cost. The efficiency of top-k search algorithms is evaluated based on the *access depth*, i.e., the maximum number of objects accessed from each list and the total

Algorithm 1 Framework for list-based top-k query processing techniques

```
Input: $S$: set of objects
       $\mathcal{L} = \{L_1, \ldots, L_r\}$: a set of ranked lists over $S$
       $f_w$: scoring function
       $k$:number of returned results
Output: $TOP_k$
 1: $f_{min} \leftarrow -\infty$
 2: $thres \leftarrow +\infty$
 3: $V \leftarrow \emptyset$ //visited objects
 4: $TOP_k \leftarrow \emptyset$
 5: while $f_{min}(V) < thres$ do
 6:     $i \leftarrow$ choose a list to read from
 7:     $o \leftarrow$ read next object from $L_i$
 8:     update the score of $o$
 9:     $V \leftarrow V \bigcup \{o\}$
10:     update $TOP_k$
11:     $thres \leftarrow \max(TOP_k)$
12:     update $f_{min}(V)$
13: end while
14: return $TOP_k$
```

processing time, factors which both reflect the efficiency of the f_{min} estimation value.

Algorithms using R-tree-like structures start by accessing the root of the tree and continue by expanding the node with the best possible score. All visited nodes are maintained in a priority queue and the algorithm stops when k objects have been retrieved. Algorithm 2 describes the framework of index-based techniques [103]. The efficiency of index-based algorithms depends mainly on the pruning abilities of the used index. A significant advantage of index-based techniques is that they are I/O optimal [103] and that it is not necessary to precompute a set of ranked lists in order to process a top-k query.

2.2 Reverse top-k queries

Top-k queries help users identify objects that are most suitable to their preferences by presenting to them a ranked set of k objects and thus prevent them from examining a large number of results. Clearly, the visibility of an object

Algorithm 2 Framework for index-based top-k query processing techniques

```
Input: $I$: multi-dimensional index
       $f_w$ :scoring function
       $k$: number of returned results
Output: $TOP_k$
 1: $TOP_k \leftarrow \emptyset$
 2: PQ$\leftarrow \emptyset$ //Priority Queue
 3: n$\leftarrow$ I.root()
 4: PQ.push(n,n.bestScore($f_w$))
 5: while PQ$\neq \emptyset$ do
 6:   $n \leftarrow$PQ.pop()
 7:   if n is a leaf entry then
 8:     $TOP_k \leftarrow \mathrm{TOP}_k \bigcup \{\mathrm{n}\}$
 9:     if $|TOP_k| = k$ then
10:       return $TOP_k$
11:     end if
12:   end if
13:   for all child nodes e of n do
14:     PQ.push(e,e.bestScore($f_w$))
15:   end for
16: end while
17: return $TOP_k$
```

o depends on the number of users for which o is in their query results. This information is of particular interest for companies wishing to identify which of their products are visible to the users and which products are shadowed by other competitive products. Enterprises are also interested in identifying the group of users to which a product is visible. Such information would help a company to estimate and improve the visibility of current and new products and select the appropriate user group for promoting a specific item.

Reverse Top-k queries [110] aim to identify the set of users for which an object is visible. The *monochromatic* Reverse Top-k (mRTOP$_k$) query describes the preferences of the users to whom an object is visible. More precisely, given a query object q, the result of an mRTOP$_k$ query is the locus of preferences w for which q is in the $TOP_k(w)$ set.

Definition 2.3. ***Monochromatic Reverse Top-k query [110].*** *Given an object q, a positive number k, and a set of $d-$dimensional objects S, the re-*

sult of a monochromatic *Reverse Top-k query, $mRTOP_k(q)$ is the locus of $d-$dimensional vectors w for which it holds that $\exists o \in TOP_k(o)$ such that $f_w(q) \leq f_w(o)$.*

Returning to Figure 2.1, mRTOP$_2(A)$ is the locus of vectors for which it holds that $0.278 \leq w[1]/w[2] \leq 1$. For any other preference vector w, A is not going to be in the $TOP_2(w)$ set because either B or C is going to have a better score than A.

The mRTOP$_k$ query indicates the area of users who can see q in their TOP_k sets. It is however, not certain that a product with a non-empty locus will be visible by any users as there might not be any users who have such preferences. The *bichromatic* Reverse Top-k (bRTOP$_k$ or simply $RTOP_k$) query avoids this problem by calculating the exact set of users who can see q in their result sets of their TOP_k queries. A bRTOP$_k$ query takes as input a query object q, a set of objects S, and a set of preferences W and it calculates the set of users to whom q is visible.

Definition 2.4. ***Bichromatic Reverse Top-k query [110].*** *Given an object q, a positive number k, and two datasets S and W, where S represents data objects and W is a set of weighting vectors, a weighting vector $w \in W$ belongs to the reverse top-k result set (bRTOP$_k(q)$) of q, if and only if $\exists o \in TOP_k(w)$ such that $f_w(q) \leq f_w(o)$.*

Returning to the example of Figure 2.1, the result set of a bRTOP$_2(G)$ query for point G is the set $\{w_1, w_2\}$ as G is in the top-2 set of both w_1 and w_2, while the bRTOP$_2(A)$ set for point A is the set $\{w_2\}$, as A is in the top-2 set of w_2 but is not in the top-2 set of the w_1 vector.

The cardinality of the bRTOP$_k(o)$ set of an object o is referred in literature as influence score [112] of o and is denoted as $f^I_k(o)$. The influence score of an object o, is indicative of the visibility of an object, as it shows the number of users who see o in their result sets. Objects that have a high rank in the result sets of many users are more likely to be preferred over objects that are ranked lower, and they are possibly not visible to the users. To that direction, Vlachou et al. [112] introduced the top-m influential query, which given a set of objects S and of preferences W, returns the set of m objects with the highest influence score.

Definition 2.5. ***Influence Score [112].*** *Given a positive integer k, a dataset S and a set of preferences W, the influence score $f^I_k(o)$ of a data object o is defined as the cardinality of the reverse top-k query result set of object o: $f^I_k = |b\text{RTOP}_k(o)|$.*

Definition 2.6. ***Top-m Most Influential Data Objects [112].*** *Given a positive integer k, a dataset S and a set of preferences W, the result set $ITOP_k^m$ of the top-m influential query is a ranked set of objects such that $ITOP_k^m \subseteq S$, $|ITOP_k^m| = m$ and $\forall o_i, o_j o_i \in ITOP_k^m o_j \in S - ITOP_k^m$ it holds that $f_k^I(o_i) \geq f_k^I(o_j)$.*

2.3 Keyword search in structured data

Keyword search is one of the most popular querying methods because of its simplicity, intuitiveness, and the abstraction it offers with respect to the structure of the data. While in text databases information is organized in documents, in relational databases information is scattered in tuples of different relations. As a result, tuples from different relations have frequently to be combined in order for the desired results to be produced. There have been many approaches for searching in relational databases and combining tuples of multiple relations [4, 8, 11, 51, 53, 55, 62, 86, 88]. In the following, we are going to present the generic framework that most of the existing approaches adopt.

2.3.1 Keyword search framework

Let D be a database containing a set of relations $\mathcal{R} = \{R_1, \ldots R_n\}$, and $\mathcal{G}_S^d$ be a directed *schema graph* that captures the primary-foreign key relationships in the database schema. Each node of the graph corresponds to a relation R_i while two nodes R_i, R_j are connected with an edge $E_\kappa = (R_i, R_j)$ if and only if there is a primary-foreign key relationship $R_i \rightarrow R_j$. We denote as $\mathcal{G}_S^u$ the undirected version of $\mathcal{G}_S^d$. Given a keyword query $Q = \{q_1, \ldots, q_n\}$ of m terms, graph $\mathcal{G}_S^u$ is used to create *Joining Trees of Tuples* (JTTs) which are trees of tuples connected through primary-foreign key relationships.

Definition 2.7. ***Joining Tree of Tuples (JTT).*** *Given an undirected schema graph $\mathcal{G}_S^u$, a Joining Tree of Tuples (JTT) is a tree of tuples T, such that for each pair of adjacent tuples t_i, t_j, $t_i \in R_i, t_j \in R_j$ there is an edge (R_i, R_j) in $\mathcal{G}_S^u$ and it holds that $(t_i \bowtie t_j) \in (R_i \bowtie R_j)$.*

Total JTT: A JTT is called *total* with respect to a query Q if it contains all terms of Q.
Minimal JTT: A JTT T is called *minimal* with respect to a query Q if and only if no tuple can be removed from T and T to remain total.

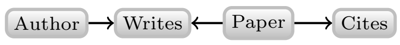
Author	
aid	name
a_1	John Smith
a_2	Peter Jones
a_3	James Hicks
a_4	George Backs

Paper	
pid	title
p_1	Keyword Queries in Databases
p_2	Database Queries
p_3	Exploratory Search in Databases
p_4	Keyword Search in Relational Data

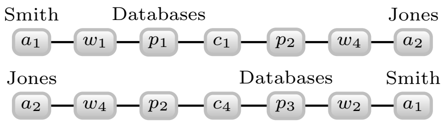
Writes		
wid	aid	pid
w_1	a_1	p_1
w_2	a_1	p_3
w_3	a_2	p_4
w_4	a_2	p_2
w_5	a_3	p_4

Cites		
wid	pid	cites_pid
c_1	p_1	p_2
c_2	p_1	p_4
c_3	p_4	p_3
c_4	p_2	p_3

(a) Database



(b) Schema graph


Smith
Databases
Jones
a_1 w_1 p_1 c_1 p_2 w_4 a_2
Jones
Databases
Smith
a_2 w_4 p_2 c_4 p_3 w_2 a_1

(c) JTT

Figure 2.2: Sample Database

Typically the result of a keyword query Q is a set of total and minimal JTTs with respect to the query, a strategy that considers valid results to be JTTs containing all terms of the query and no excessive tuples.

As an example, consider the database in Figure 2.2. The database contains 4 relations where each table has as primary key an id field. Figure 2.2(b) shows the G_S^d graph describing the primary-foreign key relationships between the relations. Given a keyword query $Q = \{\text{Smith, Jones, Databases}\}$, Figure 2.2(c) shows two valid results of the query. The first result shows a paper written by Smith which contains the term "Databases" and cites a paper of Jones, while the second result shows a paper written by Jones which cites a paper of Smith containing the term "Databases".

JTT generation methods. In literature there are two basic paradigms for generating JTTs, namely the *instance-based* approaches and the *schema-based* approaches [86]. In instance-based approaches, a graph $G_I = (\mathcal{V}_I, \mathcal{E}_I)$ of the database instance is constructed and maintained. The node set of the graph consists of the tuples of the database, and two tuples $t_i \in R_i, t_j \in R_j$ are

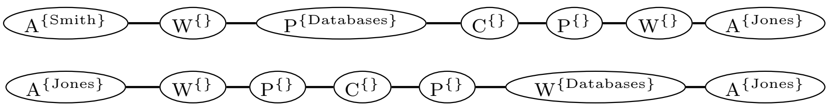


Figure 2.3: JTS example

connected by an edge if and only if there is an edge (R_i, R_j) in $\mathcal{G}_S^u$ and $(t_i \bowtie t_j) \in (R_i \bowtie R_j)$. The edges of G_I can be weighted or unweighted depending on the approach. Given a keyword query Q, most instance-based algorithms locate tuples that contain at least one term of the query and for each tuple they create a JTT with a single node. The JTTs are then expanded until they become total or until they exceed a predefined size. In the former case, they are added to the result set, while in the latter, they are discarded as not valid results.

Schema-based approaches use the directed schema graph G_S^d in order to create *Joining Trees of TupleSets* (JTSs) [55] which are translated into SQL queries. A JTS is essentially a tree that describes the relational algebra that joins a sequence of relations in order to produce a set of minimal and total JTTs [88]. Given a keyword query Q, each node $R_i^{\{X\}}$ of a JTS is described by a relation R_i and a subset X of Q, and corresponds to the relational algebra expression $\sigma_{X \wedge \neg\{Q-X\}} R_i$. Each edge of a JTS corresponds to a join between the tuplesets of two nodes. Figure 2.3 shows the JTSs for generating the JTTs illustrated at Figure 2.2(c). Note that an empty set of keywords corresponds to a tupleset that includes all tuples that do not contain any term of the query. In that way node $A^{\{Smith\}}$ corresponds to the set of tuples in *Authors* relation that contain the term "Smith" but not the terms "Jones" and "Databases", while node $P^{\{\}}$ defines the set of tuples in *Papers* relation that do not contain any of the terms "Jones", "Smith" or "Databases". JTSs are translated into SQL queries, which generate the JTTs that constitute the result set of the keyword query. Both schema-based and instance-based approaches generate equivalent result sets but they follow different ranking strategies.

Result ranking. Typically, results are ranked based on their size. In general, JTTs consisting of few nodes are considered to be more relevant to a query Q than JTTs consisting of more nodes. Due to their translation to SQL queries schema-based techniques have limited ranking capabilities while instance-based techniques support various methods of ranking. Most of them use the inner and outer degree of the nodes of the instance graph G_I [8, 11].

Part II

Exploratory Search

In this part, we present algorithms which provide users means for exploratory search enabling them to acquire a wide overview of the database content. In Chapter 3, we propose a framework for grouping summarizing keyword search results on relational data, while in Chapter 4, we introduce the eXploratory Top-k Join query which presents to the users combinations of objects fitted to their preferences.

Chapter 3

A Framework for Grouping and Summarizing Keyword Search Results

Keyword queries are inherently ambiguous, and when applied in structured data, they produce large numbers of results, which vary both in content and structure. In this chapter, we describe a framework for organizing keyword search results into groups, which contain results of similar content and similar temporal characteristics. Each group is described by a summary that provides the user with information about the content of the group assisting the user to comprehend the query results.

3.1 Introduction

Keyword-based search is extremely popular as a means for exploring information of interest without using complicated queries or being aware of the underlying structure of the data. Existing approaches for keyword search in relational databases use either the database schema (e.g., [4, 55]), or the given database instance (e.g., [11]) to retrieve tuples containing the keywords of a posed query. For example, consider the movie database instance depicted in Figure 3.1. For the keyword query $Q = \{$*comedy, J. Davis*$\}$, the results are the *comedy* movies *Deconstructing Harry* and *Celebrity* both with *J. Davis.*

Movies

idm	title	genre	year	director
m_1	Annie Hall	drama	1977	W.Allen
m_2	Interiors	drama	1978	W.Allen
m_3	Manhattan	drama	1979	W.Allen
m_4	Broadway Danny Rose	comedy	1984	W.Allen
m_5	The Purple Rose of Cairo	comedy	1985	W.Allen
m_6	Hannah and her Sisters	comedy	1986	W.Allen
m_7	Deconstruting Harry	comedy	1997	W.Allen
m_8	Celebrity	comedy	1998	W.Allen

Plays

idm	ida
m_1	a_1
m_2	a_1
m_3	a_1
m_4	a_2
m_5	a_2
m_6	a_2
m_7	a_3
m_8	a_3

Actors

ida	name	gender	dob
a_1	D. Keaton	female	1946
a_2	M. Farrow	female	1945
a_3	J. Davis	female	1955

Figure 3.1: Database instance

Given the huge volume of available data, keyword-based searches typically return result sets of size which is prohibitive for the users to process. Thus, a significant part of the database content is practically not visible to the users. Previous approaches mostly focus on ranking the results of keyword queries to help users retrieve a small piece of them. Such approaches include, among others, adapting IR-style document relevance ranking strategies (e.g., [53]) and exploiting the link structure of the database (e.g., [11]). Still, this flat ranked list of data items provides a narrow overview of the available information as it limits severely the amount of data presented to the users and does not make it easy for them to explore and discover important items relevant to their needs.

We consider an alternative presentation of the results of queries expressed through sets of keywords. In particular, we add some structure to the ranked lists of query results. Our goal is to help users receive a broader view of the query results, give the users the opportunity to learn about data items that they are not aware of, and increase in this way the visibility of the database content.

Towards this direction, we organize the keyword query results into groups, trying to have groups that exhibit internal cohesion and external isolation. This way, it is easier for the users to scan the results of their queries. Our primary focus is on producing informative, expressive and meaningful groups containing results with similar content that refer to similar temporal characteristics. For example, assume the database instance of Figure 3.1 and the keyword query $Q = \{$*W. Allen, female*$\}$. Intuitively, for this query, we can construct three groups of results; the first group refers to the movies *Annie Hall*, *Interiors* and *Manhattan*, the second group refers to the movies *Broadway Danny Rose*, *The Purple Rose of Cairo* and *Hannah and her Sisters* and the third one to the movies *Deconstructing Harry* and *Celebrity.* Each group contains movies with the same actress (*content similarity*) that are produced at the same time period (*temporal similarity*).

To help users refine their queries, we provide them with summaries over the groups of their query results. The summary of a group presents the most important keywords associated with the specific group of results. For instance, for the above constructed groups, we may have the summaries {*drama, D. Keaton*}, {*comedy, M. Farrow*} and {*comedy, J. Davis*}.

Finally, we evaluate the effectiveness of our approach. Our results indicate that users are more satisfied when the results are presented in groups and summarized.

In a nutshell, we make the following contributions:

- We introduce a framework that offers a different way for presenting the results of keyword-based searches.
- We exploit the content of results along with their temporal characteristics to produce groups of results with similar content referring to the same time periods. Summaries for the groups of results are presented to users as hints for query refinement.
- We present the results of a user study comparing our framework to a standard keyword search technique.

The rest of the chapter is organized as follows. Section 3.2 describes the related work, while Section 3.3 describes our framework for grouping and summarizing the results of keyword-based searches. In Section 3.4, we present our evaluation findings and finally, Section 3.5 presents the drawn conclusions.

3.2 Related work

In this section, we provide an overview of the related research literature.

Keyword Search. Keyword search in relational databases has been the focus of much current research. Schema-based approaches (e.g., [4, 55]) use the schema graph to generate join expressions and evaluate them to produce tuple trees. Instance-based approaches (e.g., [11]) represent the database as a graph in which there is a node for each tuple. Results are provided directly by using a Steiner tree algorithm. Based on [11], several more complex approaches have been proposed (e.g., [51, 62]). There have also been proposals for providing ranked keyword retrieval, which include incorporating IR-style relevance ranking [53], authority-based ranking [8], automated ranking based on workload and data statistics of query answers [17] and preference-based ranking [99, 101].

Our approach is different, in that we propose grouping keyword search results to help users receive the general picture of the results of their queries. A comparison between a flat ranked list of results and a clustering web search interface shows that the users of the clustering approach view more documents and spend less time per document [126]. However, the relevance of the viewed documents is unknown. [87] presents an approach for clustering keyword search results based on common structure patterns without taking into account the aspect of time. Recently, [5] introduced a prototype and framework for interactive clustering of query results. This technique is applied in document collections, while our work focuses on structured data.

Tag clouds. Summaries of keyword queries results resemble the notion of *tag clouds*. A tag cloud is a visual representation for text data. Tags are usually single words, alphabetically listed and in different font size and color to show their importance[1]. Tag clouds have appeared on several Web sites, such as Flickr and del.icio.us. With regard to our approach for summaries, *data clouds* [67] are the most relevant. This work proposes algorithms that try to discover good, not necessarily popular, keywords within the query results. Our approach follows a pure IR technique to locate important, in terms of popularity, keywords. From a database perspective, [35] introduces the notion of *object summary* for summarizing the data in a relational database about a particular *data subject*, or keyword. An object summary is a tree with a tuple containing the keyword as the root node and its neighboring tuples containing additional information as child nodes.

Faceted search. Finally, our work presents some similarities with faceted search (e.g., [9, 94]). Faceted search is an exploration technique that provides a form of navigational search. In particular, users are presented with query results classified into multiple categories and can refine the results by selecting different conditions. Our approach is different in that we do not tackle refinement. [30, 100] present a different way for database exploration by recommending to users items that are not part of the results of their query but appear to be highly related to them. Such items are computed based on the most interesting sets of attribute values that appear in the results of the original query. The interestingness of a set is defined based on its frequency in the results and the database.

[1]http://en.wikipedia.org/wiki/Tag_cloud

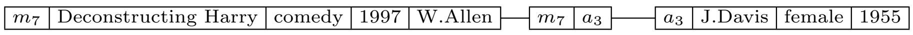

m_7 | Deconstructing Harry | comedy | 1997 | W.Allen
m_7 | a_3
a_3 | J.Davis | female | 1955

Figure 3.2: JTT result for the query $Q = \{\text{comedy, J.Davis}\}$

3.3 Framework

Let $\mathcal{D}$ be a database with a set of n relations $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. We assume that some relations in $\mathcal{R}$ include, in addition to their base attributes, a time-related attribute B that represents the time that a tuple was inserted in the database. We consider W to be the potentially infinite set of all keywords and a keyword query Q to consist of a set of keywords, i.e., $Q \subseteq W$. Our goal is twofold; first, we focus on organizing into groups the results of a keyword query based on their content similarity and the similarity on the values of their time-related attributes and then, we highlight the important keywords in the produced groups of results.

3.3.1 Keyword search

Typically, the result of a keyword query is defined with regards to joining trees of tuples (JTTs), which are trees of tuples connected through primary to foreign key dependencies. For example, the JTT of Figure 3.2 represents a JTT for the keyword query $Q = \{$*comedy, J. Davis*$\}$. The size of a JTT is equal to the number of its tuples. In this case, the aforementioned JTT has a size equal to 3. We consider as valid results JTTs which are both total and minimal, i.e., it is required for a valid JTT to contain all terms of the query and no sub-tree of a valid JTT to be total as well.

As described in Section 2.3 each JTT in the result set corresponds to a JTS, namely a tree at schema level. The above JTT corresponds to the schema level tree $Movies^{\{comedy\}} - Play^{\{\}} - Actors^{\{J.Davis\}}$, where each R_i^X consists of the tuples of R_i that contain all keywords of X and no other keyword of Q. Several algorithms in the research literature aim at constructing JTSs for a query Q as an intermediate step of the computation of the final results (e.g. [4, 55]). We adopt the approach of [55] in which all JTSs with size up to l are constructed. In particular, given a query Q, all possible tuple sets R_i^X are computed, where $R_i^X = \{t | t \in R_i \wedge \forall a_x \in X, t \text{ contains } a_x \wedge \forall a_y \in Q - X, t \text{ does not contain } a_y\}$. After selecting a random query keyword a_z, all tuple sets R_i^X for which $a_z \in X$ are located. These are the initial JTSs with only one node. Then, these trees are expanded either by adding a tuple set that contains at least another query keyword or a tuple set for which $X = \{\}$ (free tuple set). These trees can be further expanded. JTSs that contain all query keywords are returned, while JTSs of the form $R_i^X - R_j^{\{\}} - R_i^Y$, where an edge $R_j \rightarrow R_i$ exists in the schema graph, are pruned, since JTTs produced by them have more than one occurrence of the same tuple for every instance of the database.

3.3.2 Keyword search result vector representation

Our effort focuses on grouping results based on their content and some temporal information associated with them. Regarding the content of a JTT, we may think of a JTT as the equivalent of a "document". Then, the textual content of a JTT T can be represented by a term-vector u_T. For a query Q with result $Res(Q)$, let $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The importance score $x_{i,j}$ of a keyword a_i in $\mathcal{A}$ for the JTT T_j of $Res(Q)$, is defined with respect to the TF-IDF model [27]. Specifically, for each a_i in $\mathcal{A}$ for T_j, $x_{i,j}$ is equal to: $x_{i,j} = tf_{i,j} * log(N/df_i)$, where $tf_{i,j}$ is the number of occurrences of a_i in the JTT T_j, df_i is the number of tuples in $\mathcal{D}$ that contain a_i and N is the cardinality of the result set $Res(Q)$. Formally, a JTT-vector for a specific JTT is:

Definition 3.1. ***JTT-vector.*** *Let Q be a keyword query with query result Res(Q) and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The JTT-vector of a JTT T_j in Res(Q) is a vector $u_{T_j} = \{(a_1, x_{1,j}), \dots, (a_m, x_{m,j})\}$, where $a_i \in \mathcal{A}$, $|\mathcal{A}| = m$, and $x_{i,j}$ is the importance score of a_i for T_j, $1 \leq i \leq m$.*

Many times, two JTTs may contain very similar information. Next, we will exploit similarities between JTTs in order to construct groups of similar results.

3.3.3 Finding groups of keyword search results

We consider that each database relation includes in its schema a time-related attribute B. In case a relation does not contain time-related data, we consider a virtual time-related attribute $B = 0$ for all tuples in the relation. Then, for a tuple t_i of a relation R_j, $1 \leq j \leq n$, we refer to the value $t_i[R_j.B]$ as the *age* of t_i. Naturally, time-related attributes of the database relations may vary. For instance, for a relation containing *movie titles*, the production year is a time-related attribute associated with the age of each tuple and for a relation with *actors* the respective attribute is the *date of birth*. For two tuples t_i, t_x of the relations R_j, R_y, we say that t_i is more recent than t_x, if and only if, $t_i[R_j.B] > t_x[R_y.B]$, $1 \leq j, y \leq n$.

Given a joining tree of tuples T, we define its *age* with respect to the age of the tuples appearing in the tree. In particular, the age of T is determined by the age of the most recent of its tuples. The motivation behind this, is that before the insertion of the most recent tuple the tree did not exist. For example, let $Movies.B$ be the attribute *year* of the relation $Movies$, $Actors.B$ be the attribute *dob* (date of birth) of the relation $Actors$ while each tuple t_i of the relation $Play$ has value $t_i[Play.B]$ equal to 0. Then, the age of the JTT of Figure 3.2 is 1997 which is the most recent date in the JTT and the production year of the movie. Formally:

Definition 3.2. ***Age of JTT.*** *Given a JTT T with tuples* $t_1 \in R_{j_1}, \ldots, t_p \in R_{j_p}$, $1 \leq j_1, j_p \leq n$, *the age of T,* age_T, *is:*

$$age_T = \max_{1 \leq i \leq p} \{t_i[R_{j_i}.B]\}$$

Our goal here is to detect groups of JTTs. Each group contains JTTs that: (i) have similar content, and (ii) are continuous in time, which means that their *age* values increase. A straightforward way for quantifying the similarity between two JTTs, is to use a cosine-based definition of similarity, which measures the similarity between their corresponding vectors.

Definition 3.3. ***Cosine JTT Similarity.*** *Given two JTTs* T_1 *and* T_2 *with vectors* $u_{T_1} = \{(a_1, x_{1,1}), \ldots, (a_m, x_{m,1})\}$ *and* $u_{T_2} = \{(a_1, x_{1,2}), \ldots, (a_m, x_{m,2})\}$, *respectively, the cosine JTT similarity between* T_1 *and* T_2 *is:*

$$sim_c(T_1, T_2) = \frac{u_{T_1} \cdot u_{T_2}}{||u_{T_1}||||u_{T_2}||} = \frac{\sum_{i=1}^{m} x_{i,1} \times x_{i,2}}{\sqrt{\sum_{i=1}^{m}(x_{i,1})^2} \times \sqrt{\sum_{i=1}^{m}(x_{i,2})^2}}$$

Given the similarity between JTTs, we focus on the grouping process. A group of JTTs is expressed as a set of JTTs. The JTTs of a group G_j define a time interval described by two time instances $G_j.s$ and $G_j.e$; $G_j.s$ denotes the starting point of the interval and corresponds to the age of the oldest JTT in the group, while $G_j.e$ denotes the ending point of the interval and corresponds to the age of the most recent JTT. For example, for a group G_j consisting of the JTTs (i) (m_1, *Annie Hall, drama, 1977, W. Allen*) – (m_1, a_1) – (a_1, *D. Keaton, female, 1946*), (ii) (m_2, *Interiors, drama, 1978, W. Allen*) – (m_2, a_1) – (a_1, *D. Keaton, female, 1946*), and (iii) (m_3, *Manhattan, drama, 1979, W. Allen*) – (m_3, a_1) – (a_1, *D. Keaton, female, 1946*), $G_j.s$ = 1977 and $G_j.e$ = 1979.

Similarly to the JTT-vector, we define the Group-vector which describes the content of a group. In particular, the Group-vector of a group G_j is an aggregation of all vectors of the JTTs belonging to G_j. For a query Q with result $Res(Q)$, let $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$ and G_j be a group of JTTs in $Res(Q)$. The importance score $s_{i,j}$ of a keyword a_i in $\mathcal{A}$ for the group G_j is equal to: $s_{i,j} = aggr_{T_w \in G_j}(x_{i,w})$, where *aggr* is an aggregation function such as *average*, *sum*, *maximum* or *minimum* of the values $x_{i,w}$ of the JTTs of G_j.

Definition 3.4. ***Group-vector.*** *Let G_j be a group of JTTs belonging to the query result $Res(Q)$ of a query Q and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The Group-vector of G_j is a vector $u_{G_j} = \{(a_1, s_{1,j}), \ldots, (a_m, s_{m,j})\}$, where $a_i \in \mathcal{A}$, $|\mathcal{A}| = m$, and $s_{i,j}$ is the importance score of a_i for G_j, $1 \leq i \leq m$.*

Given the query result $Res(Q)$ of a query Q, our aim is to partition the JTTs of $Res(Q)$ into non-overlapping groups. Our definition for non-overlapping groups takes into account both time and content overlaps. Specifically, two groups are: (i) non-overlapping with respect to time, if their time intervals are disjoint, and (ii) non-overlapping with respect to content, if they do not contain common JTTs.

Definition 3.5. ***Non-overlapping Groups.*** *Let G_i, G_j be two groups of JTTs with time-intervals $[G_i.s, G_i.e]$, $[G_j.s, G_j.e]$. G_i, G_j are non-overlapping groups, if and only if: (i) ($G_i.s > G_j.e$ and $G_i.s > G_j.s$) or ($G_j.s > G_i.e$ and $G_j.s > G_i.s$), and (ii) $G_i \cap G_j = \emptyset$.*

To partition the joining trees of tuples into non-overlapping groups, we employ a bottom-up hierarchical agglomerative clustering method. Initially, the *JTT Partitioning Algorithm* (Algorithm 3) places each JTT in a cluster of its own. Then, at each iteration, it merges the two most similar clusters. The similarity between two clusters is defined as the minimum similarity between any two JTTs that belong to these clusters (*max linkage*). That is, for two clusters, or groups, G_1, G_2: $sim(G_1, G_2) = \min_{T_i \in G_1, T_j \in G_2}\{sim_c(T_i, T_j)\}$.

Clearly, two clusters, or groups, G_1, G_2 can be merged if they are non-overlapping groups. But this is not enough. For constructing groups with JTTs with growing age values there is also a need to ensure that, for the groups G_1, G_2, there is no other group G_3 with time interval between the time intervals of G_1 and G_2. We refer to such groups as merge-able groups. Formally:

Definition 3.6. ***Merge-able Groups.*** *Let G_i, G_j be two groups of JTTs with time-intervals $[G_i.s, G_i.e]$, $[G_j.s, G_j.e]$. G_i, G_j are merge-able groups, if and only if: (i) G_i, G_j are non-overlapping groups, and (ii) $\nexists G_p$ with time interval*

$[G_p.s, G_p.e]$*, such that, the groups* G_i*,* G_p *and* G_p*,* G_j *are non-overlapping, and* *(*$G_p.s > G_i.e$ *and* $G_j.s > G_p.e$*) or (*$G_p.s > G_j.e$ *and* $G_i.s > G_p.e$*).*

Thus, in overall, we proceed in merging two groups only if the groups are merge-able. The algorithm stops either when a single cluster containing all the JTTs of $Res(Q)$ has already produced or when no more clusters can be merged. As a final step, the algorithm selects to return the clusters of the iteration that present the maximum clustering quality. The clustering quality C_i, computed after merging the two clusters of a specific iteration i, is:

$$C_i = \sum_{j=1}^{K_i} \sum_{\forall T_p \in G_j} u_{T_p} \cdot u_{G_j} \tag{3.1}$$

where K_i is the number of clusters after the merging operation of iteration i. The selected iteration is the one that constructs K^* clusters, such that:

$$K^* = argmax_i(C_i - \lambda K_i) \tag{3.2}$$

where λ is a penalty for each additional cluster.

Algorithm 3 presents a high-level description of the *JTT Partitioning Algorithm.* As a final note, consider that alternatively we can pre-specify the number of clusters K^* and directly select to return the clusters of the iteration that produces the K^* ones. Instead, in this work, to ensure high clustering quality, we opt for following the above described procedure, even if the resulting processing cost is high.

We illustrate our approach with the following example. Assume the keyword query $Q = \{$*W. Allen, female*$\}$. For the database instance of Figure 3.1, the result set $Res(Q)$ consists of the JTTs of Figure 3.3 while the ages of the trees are the highlighted dates. Applying the *JTT Partitioning Algorithm* results in producing three groups G_1, G_2 and G_3 with trees $\{T_1, T_2, T_3\}$, $\{T_4, T_5, T_6\}$ and $\{T_7, T_8\}$.

3.3.4 Summaries of keyword query results

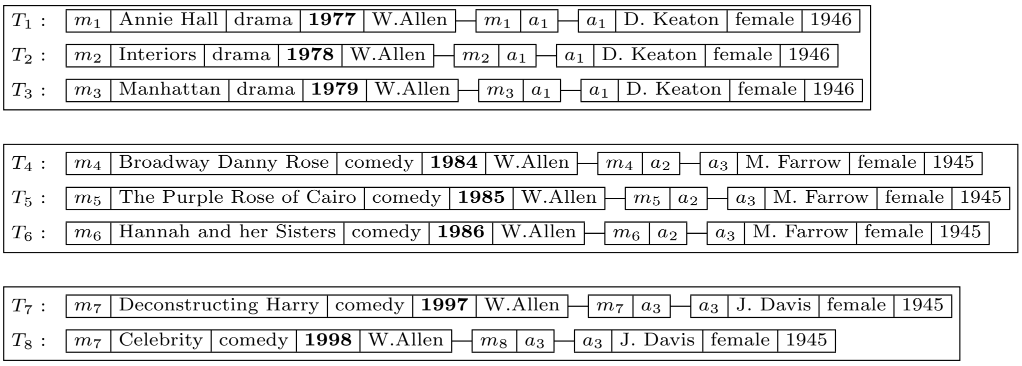
In this section, we describe the notion of group summaries that put in a nutshell the results within groups of keyword searches. In general, group summaries provide hints for query refinement and can lead to discoveries of interesting results that a user may be unaware of.

Let $Res(Q)$ be the query results of a query Q and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. Let also $G_1, \ldots, G_z$ be the groups of JTTs

Algorithm 3 JTT Partitioning Algorithm

Input: A set of JTTs.
Output: A set of groups of JTTs.

```
 1: Create a group for each JTT
 2: Repeat
 3:   $i = 1$
 4:   Locate the two merge-able groups with the maximum similarity
 5:     **If** there are no merge-able groups or only one group exists **then**
 6:       End loop
 7:     **Else**
 8:       Merge the two groups
 9:       Compute $K_i$, $C_i$
10:       $i$++
11: Select the partitioning that constructs $K^*$ groups
```



T_1 : m_1 | Annie Hall | drama | **1977** | W.Allen — m_1 | a_1 — a_1 | D. Keaton | female | 1946
T_2 : m_2 | Interiors | drama | **1978** | W.Allen — m_2 | a_1 — a_1 | D. Keaton | female | 1946
T_3 : m_3 | Manhattan | drama | **1979** | W.Allen — m_3 | a_1 — a_1 | D. Keaton | female | 1946
T_4 : m_4 | Broadway Danny Rose | comedy | **1984** | W.Allen — m_4 | a_2 — a_3 | M. Farrow | female | 1945
T_5 : m_5 | The Purple Rose of Cairo | comedy | **1985** | W.Allen — m_5 | a_2 — a_3 | M. Farrow | female | 1945
T_6 : m_6 | Hannah and her Sisters | comedy | **1986** | W.Allen — m_6 | a_2 — a_3 | M. Farrow | female | 1945
T_7 : m_7 | Deconstructing Harry | comedy | **1997** | W.Allen — m_7 | a_3 — a_3 | J. Davis | female | 1945
T_8 : m_7 | Celebrity | comedy | **1998** | W.Allen — m_8 | a_3 — a_3 | J. Davis | female | 1945

Figure 3.3: Grouping example

produced for $Res(Q)$. Our goal is to compute an importance score $s_{i,j}$ for each keyword a_i in $\mathcal{A}$ for each group G_j, $1 \leq j \leq z$. Then, for each group G_j, the top-k keywords, that is, the k keywords with the highest importance scores are used as a summary of the JTTs in G_j. Formally:

Definition 3.7. ***Group Summary.*** *Let G_j be a group of JTTs belonging to the query result $Res(Q)$ of a query Q and $\mathcal{A}$ be the set of keywords appearing in the JTTs of $Res(Q)$. The group summary S_{G_j}, $S_{G_j} \subseteq \mathcal{A}$, of G_j is a set of k keywords, such that, $s_{i,j} \geq s_{p,j}$, $\forall a_i \in S_{G_j}$, $a_p \in \mathcal{A} - S_{G_j}$.*

For example, for the keyword query $Q = \{$*W. Allen, female*$\}$, the group summaries of the produced groups G_1, G_2 and G_3, for $k = 2$, are $S_{G_1} = \{$*drama, D. Keaton*$\}$, $S_{G_2} = \{$*comedy, M. Farrow*$\}$ and $S_{G_3} = \{$*comedy, J. Davis*$\}$.

To provide users with more detailed summaries that include some information about the schema of the results, we extend the notion of group summaries to take into account the relations that a keyword belongs to. Specifically, for each group G_j, instead of reporting the set of the k keywords with the highest importance scores, we report these keywords along with their associated relations. This way, users obtain an overview about the possible origination and meaning of the keywords. We refer to these summaries as enhanced group summaries. Formally:

Definition 3.8. ***Enhanced Group Summary.*** *Let G_j be a group of JTTs and S_{G_j} be the corresponding group summary of G_j with keywords $a_1, \ldots, a_k$. The enhanced group summary $\mathcal{E}_{G_j}$ of G_j is a set of k pairs of the form (a_i, P_i), such that, there is one pair $\forall a_i \in S_{G_j}$ and P_i is the set of relations that contain a_i for the JTTs of G_j.*

Returning to our previous example, the enhanced group summaries of G_1, G_2 and G_3 are represented as $\mathcal{E}_{G_1} = \{($*drama*, $\{$*Movies*$\})$, $($*D. Keaton*, $\{$*Actors*$\})\}$, $\mathcal{E}_{G_2} = \{($*comedy*, $\{$*Movies*$\})$, $($*M. Farrow*, $\{$*Actors*$\})\}$ and $\mathcal{E}_{G_3} = \{($*comedy*, $\{$*Movies*$\})$, $($*J. Davis*, $\{$*Actors*$\})\}$, respectively.

Based on the summaries of the produced groups of results, we define the summary of the query result as a whole, as follows:

Definition 3.9. ***Query Result Summary.*** *Let $G_1, \ldots, G_z$ be the groups of JTTs produced for the query result $Res(Q)$ of a query Q. The query result summary $\mathcal{S}_Q$ is a set of z group summaries, $\mathcal{S}_Q = \{\mathcal{S}_{G_1}, \ldots, \mathcal{S}_{G_z}\}$, such that, $\mathcal{S}_{G_j}$ is either the group summary S_{G_j} or the enhanced group summary $\mathcal{E}_{G_j}$ of G_j, $1 \leq j \leq z$.*

That is, for $Q = \{$*W. Allen, female*$\}$, the query result summary $\mathcal{S}_Q$ taking into account the group summaries is $\{\{$*drama, D. Keaton*$\}$, $\{$*comedy, M. Farrow*$\}$, $\{$*comedy, J. Davis*$\}\}$, while for the enhanced group summaries we have the summary $\{($*drama*, $\{$*Movies*$\})$, $($*D. Keaton*, $\{$*Actors*$\})$, $($*comedy*, $\{$*Movies*$\})$, $($*M. Farrow*, $\{$*Actors*$\})$, $($*comedy*, $\{$*Movies*$\})$, $($*J. Davis*, $\{$*Actors*$\})\}$.

We could also consider other versions for summaries. For instance, assume that the importance of each keyword is computed separately for each relation. Then, we may report important keywords with respect to their relation-specific scores or keywords for relations of high user interest.

Summary-based exploratory keyword queries. Besides presenting summaries to the users and offering, this way, a side means for further exploration, we can also use the summaries to directly discover interesting pieces of data that are potentially related to the users information needs. Specifically, to locate such related information, special-purpose queries, called *summary-based exploratory keyword queries*, can be constructed. The focus of these queries is on retrieving results highly correlated with the results of the original users queries.

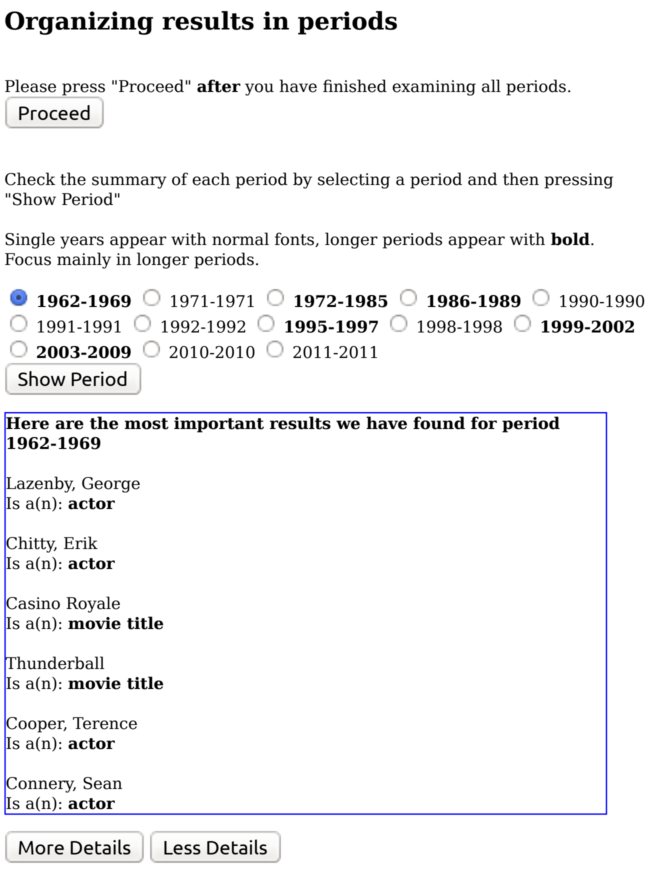
We employ the keywords of summaries to emerge new interesting results. An exploratory keyword query for a query Q consists of a set of keywords, that is a subset of the keywords in a group summary G_j of Q that frequently appear together in the JTTs of G_j. There are also other ways for constructing exploratory queries that qualify different properties. For example, sets of keywords that frequently appear in the result and, at the same time, rarely appear in the database ensure high surprise, or unexpectedness, as a measure of interestingness, as surprise used in the data mining literature (e.g., [98]). Recently, exploratory queries are used for exploration in relational databases through recommendations [30].

3.4 Evaluation

To demonstrate the effectiveness of grouping and summarizing keyword search results, we conducted an empirical evaluation of our approach using a real movie dataset[2] with 30 volunteers with a moderate interest in movies. The schema of our database is shown in Figure 3.5 while the size of the database is 1.1 GB.

We run our experiments for queries of different sizes, i.e., number of keywords, and keywords of a different selectivity. We presented the results to the participants using two methods, 1) without any grouping (baseline method) and 2) with groups produced by our approach (grouped method). In the baseline method, for each query, we presented an enhanced group summary of the whole result set, considering the whole result set as one group. To help the users understand the context of the significant terms we presented them also the attribute value in which a significant term appeared in. We give also the participants the ability to examine the set of produced JTTs. The results, i.e., the JTTs, are ranked based on their size that corresponds to the relevance of the trees to the query. In the grouped method, the participants are initially presented the groups of JTTs which were formed on the same results that were presented in the baseline. The groups are indicated to the participants by the

[2]http://www.imdb.com/interfaces



Organizing results in periods

Please press "Proceed" **after** you have finished examining all periods.

Proceed

Check the summary of each period by selecting a period and then pressing "Show Period"

Single years appear with normal fonts, longer periods appear with **bold**.
Focus mainly in longer periods.

● **1962-1969** ○ 1971-1971 ○ **1972-1985** ○ **1986-1989** ○ 1990-1990
○ 1991-1991 ○ 1992-1992 ○ **1995-1997** ○ 1998-1998 ○ **1999-2002**
○ **2003-2009** ○ 2010-2010 ○ 2011-2011

Show Period

Here are the most important results we have found for period 1962-1969

Lazenby, George
Is a(n): **actor**

Chitty, Erik
Is a(n): **actor**

Casino Royale
Is a(n): **movie title**

Thunderball
Is a(n): **movie title**

Cooper, Terence
Is a(n): **actor**

Connery, Sean
Is a(n): **actor**

Figure 3.4: Sample of results for the query "James Bond male actors"

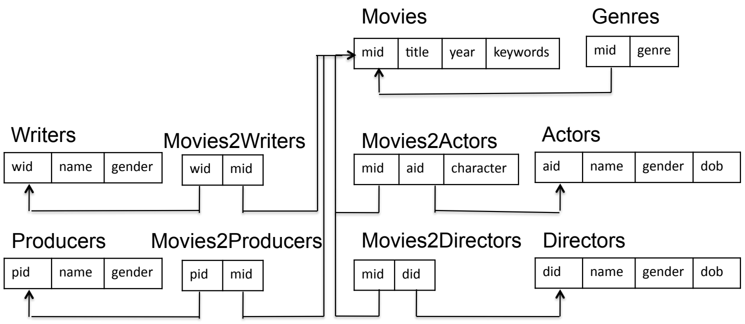


Figure 3.5: Movies database schema

time period each group covers. When a participant focuses on period he/she is provided with the summary of the group's content and the results belonging to that group.

The participants were asked to evaluate the quality of the results. For characterizing the quality, we use four measures: (i) *group coherence*, which evaluates the similarity of the results content inside a group , (ii) *baseline summary quality*, which evaluates how descriptive is the summary of the baseline method for the whole result set, (iii) *group summary quality*, which evaluates how descriptive is the summary of each group, and (iv) *usefulness evaluation*, that evaluates if the participant found the grouping method more helpful than the baseline method.

For grading the grouping, the participants were asked to evaluate for each group if the movies in the group fit well together. We used 3 values: not coherent (0), quite coherent (1), and very coherent (2). The users were also asked for each summary if it was descriptive of the result and if it was helpful for them to understand the content of the results. The summaries were also graded using three values: not descriptive (0), quite descriptive (1), and very descriptive (2). The degree of overall usefulness was graded with two values: our method is not helpful (0), and our method is helpful (1). Each query was evaluated by at least 3 participants while 95% of the queries were evaluated by at least 4 and 75% by at least 8. On average, there were 8 evaluators per query.

Group coherence. Table 3.1 shows the average values of the coherence measure for each query as they were estimated by the participants. According to the

Query	**Average group coherence**
“Daniel Craig” movies	1.50
“James Bond” movies	1.50
“James Bond” male actors	1.50
“Woody Allen” female actors	1.27
“Clint Eastwood” movies	1.50
“Peter Jackson” male actors	1.75
“Peter Jackson” movies	1.33
“Denzel Washington” Action	1.88
“Julia Roberts” Comedy	1.71
“Julia Roberts” movies	1.75
“Kevin Spacey” drama	1.27
“Jack Nicholson” female actors	1.44
“Al Pacino” movies	1.45
“Al Pacino” male actors	1.50
“Al Pacino” directors	1.50
“Stanley Kubrick” actors	1.75
“Stanley Kubrick” movies	1.60
“Lord of The Rings” Tolkien	1.30
“Robert De Niro” directors	1.60
“Francis Ford Coppola” male actors	1.75

Table 3.1: Group coherence evaluation for each query

average group coherence value, that is 1.52, the participants found the grouping of the results to be meaningful and helpful for them to understand the results.

Summary quality. Table 3.2 reports the average values of the quality measures for each query (we omit the detailed per person scores due to space limitations). As it can be seen, in 90% of the queries the quality of group summaries was better than (or equal to) the quality of the baseline summary according to the participants. This comes to complete accordance with the percent of participants (85%) who found our approach helpful. We can also draw the conclusion that while in all queries the majority of participants found the grouped summaries to be quite or very descriptive, the baseline summary was evaluated as quite or very descriptive in only 30% of the queries.

Time overhead. Finally, we study the overall impact of grouping and summarizing keyword search results in terms of time overhead for the above query examples. In particular, we measured the time needed to build the JTTs and

Query	Baseline summary quality	Group summary quality	Usefulness
“Daniel Craig” movies	0.75	1.50	0.75
“James Bond” movies	0.70	1.60	0.90
“James Bond” male actors	1.00	1.50	0.75
“Woody Allen” female actors	0.82	1.45	0.82
“Clint Eastwood” movies	1.36	1.57	0.86
“Peter Jackson” male actors	0.50	1.75	1.00
“Peter Jackson” movies	1.67	1.67	0.67
“Denzel Washington” Action	1.13	1.88	1.00
“Julia Roberts” Comedy	0.88	1.43	0.86
“Julia Roberts” movies	1.13	1.75	0.88
“Kevin Spacey” drama	0.64	1.28	0.82
“Jack Nicholson” female actors	0.22	1.56	0.89
“Al Pacino” movies	1.00	1.45	0.82
“Al Pacino” male actors	0.50	1.63	0.88
“Al Pacino” directors	0.50	1.50	1.00
“Stanley Kubrick” actors	1.00	2.00	1.00
“Stanley Kubrick” movies	1.50	1.30	0.80
“Lord of The Rings” Tolkien	1.36	1.20	0.60
“Robert De Niro” directors	0.30	1.60	0.90
“Francis Ford Coppola” male actors	1.00	2.00	1.00

Table 3.2: Summary quality evaluation for each query

the time needed for creating the groups and summaries. The additional computational cost of our approach is small in comparison with the generation of the actual keyword search results. On average, the additional time consumed for creating the summaries and groups was a magnitude smaller than the JTT building time, and in no case was the creation of groups and summaries significantly more expensive in terms of time than the building of the JTTs. For example, the time overhead for the query {Stanley Kubrick, movies} is 3.7% and for the query {Francis Ford Coppola, male actors} is 9.5%.

3.5 Conclusion

In this chapter, we considered an alternative way for presenting the results of a keyword query. We proposed a framework for organizing the results into groups that contain results with similar content that refer to similar temporal characteristics. We employed summaries of results to help users refine their searches. A summary of a result set is expressed as a set of important attribute values in the result set. Finally, we evaluated the effectiveness of our approach. Our usability results indicate that users are more satisfied when results are organized with respect to content and time than when results are simply ordered with respect to relevance.

Chapter 4

Exploratory Product Search Using Top-k Join Queries

Typically, when users are searching for a product in a database, they are presented with a ranked list of products. Frequently, a large number of the products can be combined with accessories and produce combinations that have improved properties compared to single products. In most cases however, the number of possible combinations between the products is practically infinite, and presenting to the user a large set of combinations would result in confusing her and limiting her overview of the available *main products* of interest. In this chapter, we propose a new type of query, the *Exploratory Top-k Join* query, which presents the user the available options organized according to the products of her interest.

4.1 Introduction

Nowadays, product databases contain large collections of objects, where each object is characterized by a number of different properties such as price or weight. In many cases, products can be combined with accessories and form combinations with enhanced or extended properties generating in this way numerous options for the users. For instance, a computer may be combined with additional RAM parts and SSD disks for extra memory and storage capacity. Similarly, a travel agency can offer a rented car with a small additional fee, when a user selects certain hotels that are away from the city center. The plethora of different items and combinations make it challenging for users to explore the available options and find the products that suit their needs.

Ranking queries, such as top-k [25, 34, 58] and rank-join queries [37, 57, 77] assist users to find the products that are interesting to them by selecting a small set of items or combinations that are highly ranked according to their preferences. As described in Chapter 2, in such queries products are typically modeled as multi-dimensional points, where each dimension corresponds to a specific attribute and the respective value indicates the presence or the performance of a product regarding this attribute [18, 40, 115]. User preferences are modeled as multi-dimensional vectors w, where each component (weight) of the vector denotes the importance of the respective attribute for the ranking of the objects [56, 79, 119]. The weight values could either be explicitly declared by the user through an interactive user interface [56], or be indirectly estimated [97]. Each item or combination is assigned a score, frequently using a linear scoring function [16, 79, 104, 122, 123, 129] of the form $f_w(o) = \sum_{i=1}^{d} w[i]o[i]$ that assigns scores to products, where $o[i]$ is the normalized value of the i-th attribute of a product o. However, both top-k and rank-join queries present to the users a very limited overview of the available alternatives. Top-k queries, on one hand, focus solely on the products the user is interested in, and ignore the fact that combinations can be better suited to users than single objects. Rank-join queries, on the other hand, focus on fixed-size combinations and do not consider that adding an accessory product does not necessarily result into a more preferable combination. In most cases, when main products are combined with accessory products some not-appealing attributes (e.g., price or weight) may increase. In addition, they often return similar combinations and thus they present to the user a very limited view of the available products of her interest.

To this end, we propose a new type of query, the eXploratory Top-k Join (XTJ_k) query. The XTJ_k query aims at assisting the user to explore the available options by providing her a wide range of the products she is interested in, presented in attractive combinations. In particular, an XTJ_k presents only one combination (the best) per main product focusing on the products that the user is interested in, without ignoring possible combinations that may result to a more preferable solution. Different from a rank-join query, where the user specifies the items to be combined, the combination of main and accessory products is performed automatically based on the user preferences. As a result, the user is not required to be aware of the available accessories, but she still is presented with interesting combinations. In addition to providing a wide overview of main products, an XTJ_k query provides the ability to the user to retrieve more combinations for a specific main product with minimal processing cost,

Laptops

id	CPU score	RAM (GB)	SSD (GB)	price (USD)	battery (hours)	weight (kg)	RAM type	SSD type
c_1	3346	4	0	-539	4	-2.7	1	1
c_2	3346	8	256	-1119	6.5	-2	1	2
c_3	3941	12	256	-1199	8	-2.5	0	0
c_4	3997	8	0	-1348	13	-2	2	1

Memory

mid	RAM (GB)	price (USD)	RAM type
m_1	4	-71	2
m_2	4	-45	1
m_3	8	-81	1

SSD

sid	SSD (GB)	price (USD)	SSD type
d_1	240	-143	1
d_2	512	-300	1
d_3	120	-83	2

Figure 4.1: Sample product database

Top-3 query
c_3
c_4
c_2

Rank-join query
$\{c_2, m_3, d_3\}$
$\{c_4, m_1, d_2\}$
$\{c_4, m_1, d_1\}$

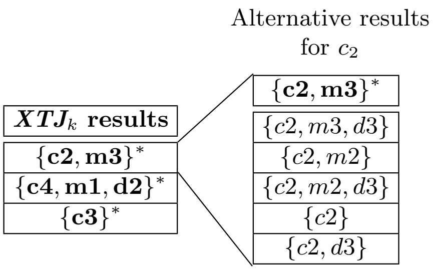
Figure 4.2: Top-k and rank-join query results

and organizes efficiently the results into groups based on the product the user is interested in.

Consider the example in Figure 4.1, which displays the database of an e-shop selling computers, and a user wishing to buy a laptop. We can assume that the user preference vector is equal to $w = (0.1, 0.2, 0.1, 0.3, 0.2, 0.1)$ and the score of each laptop is equal to $f_w = \sum_{i=1}^{d} w[i]o[i]$, where $o[i]$ is the normalized value of the i-th attribute of product o. A top-3 query and rank-join query would return the results shown in Figure 4.2. Both queries do not return the optimal results, as they do not take into account the fact that other combinations may have better scores. Table 4.1 displays the ranking of all possible combinations, and indicates that both queries do not return the best results. In addition, a rank-join query displays two very similar results, limiting the variety of the search results regarding the available products that the user is interested in (laptops).

rank	**id**
1	$\{\mathbf{c2}, \mathbf{m3}\}^*$
2	$\{c2, m3, d3\}$
3	$\{\mathbf{c4}, \mathbf{m1}, \mathbf{d2}\}^*$
4	$\{c4, m1, d1\}$
5	$\{c4, m1\}$
6	$\{\mathbf{c3}\}^*$
7	$\{c2, m2\}$
8	$\{c2, m2, d3\}$
9	$\{\mathbf{c1}, \mathbf{m3}, \mathbf{d2}\}^*$
. . .	
14	$\{c4\}$
15	$\{c2\}$
. . .	

Table 4.1: Ranking of all possible combinations



$\boldsymbol{XTJ_k}$ **results**
$\{\mathbf{c2}, \mathbf{m3}\}^*$
$\{\mathbf{c4}, \mathbf{m1}, \mathbf{d2}\}^*$
$\{\mathbf{c3}\}^*$

Alternative results for c_2

$\{\mathbf{c2}, \mathbf{m3}\}^*$
$\{c2, m3, d3\}$
$\{c2, m2\}$
$\{c2, m2, d3\}$
$\{c2\}$
$\{c2, d3\}$

Figure 4.3: Grouped results

Presenting a ranked list of all possible combinations does not help the user to acquire an insight about the available products, as the top-3 combinations (Table 4.1) involve only two laptops.

We argue that a user would be more interested in the combinations presented by an XTJ_3 query as displayed by the first table of Figure 4.3. Notice, that an XTJ_k query takes into account all possible combinations, and presents to the user only the best combination for each main product. Upon user request, more combinations for a given product can be presented to the user, providing to her a wide, yet not overwhelming view of the available options. Figure 4.3 displays the case where a user wishes to explore alternative combinations for computer c_2. The alternative combinations are organized according to the product of interest of the user, assisting her to explore the available options.

State-of-the-art techniques for processing rank-join queries [37, 57, 96] return the k highest ranked combinations according to a user-defined preference function. However, the user must be aware of the contents of the database and has to specify at query time the form of combinations. Hence, adaptations of such techniques exhibit suboptimal performance when applied to our problem. Furthermore the eXploratory Top-k Join is essentially a "star join" of the main product relation and the additional product relations, and existing techniques do not exploit the structure of this join type to achieve performance gains. We show how the structure of such queries can be exploited in order to produce efficient query processing algorithms. The proposed algorithms explore all possible joins without computing the entire set of possible combinations, and return the correct result.

To summarize, we make the following contributions:

- We propose the use of the pull-bound framework [95] for processing eXploratory Top-k Joins, and we provide a baseline algorithm that processes eXploratory Top-k Join (XTJ_k) queries, by adapting a state-of-the-art rank-join algorithm [57].
- We analyze the properties of eXploratory Top-k Joins and we propose an efficient algorithm (XRJN) that relies on an effective bounding scheme and a plain round-robin pulling strategy.
- We provide strong theoretical guarantees on the performance of our algorithm, namely we prove that XRJN is instance-optimal.
- We present a new algorithm (XRJN*), by introducing a pulling strategy that prioritizes access to relations in a deliberate manner, in order to reduce the overall processing cost.
- We propose an extension of the XTJ_k query that retrieves multiple combinations for each main product organized in groups.
- We perform an experimental evaluation that demonstrates the efficiency of our approach.

The rest of this chapter is organized as follows: Section 4.2 reviews the related work. In Section 4.3 we formally define the XTJ_k query. Thereafter, in Section 4.4, we present a baseline technique to answer an XTJ_k query. In Section 4.5, we provide a more efficient algorithm that exploits the characteristics of the combinations in order to produce faster converging upper and lower bounds. Section 4.6 describes the generation of multiple combinations per main object. The experimental evaluation is presented in Section 4.7 and we conclude in Section 4.8.

4.2 Related work

Our work is related to top-k and join queries as well as package recommendation. In the following, we present an overview of the related research work.

Top-k queries. Top-*k* queries have been well-studied in the last years to enable ranked retrieval of objects based on user preferences. Top-*k* queries where first studied by Fagin [34]. Das et al. [25] introduced an algorithm using views. Ge et al. [40] follow a similar approach using precomputed views but their goal is to improve the performance on batch top-*k* queries. Their approach shows a special interest as they avoid linear programming for calculating the upper bound. For a thorough overview of top-*k* queries we refer to [58].

Rank-join queries. Rank join queries were first studied by Natsev et al. who introduced the J* algorithm [83]. Ilyas et al. proposed the HRJN* algorithm [57] which outperforms J*. Mamoulis et al. [77] introduced the LARA-J and LARA-J* algorithms, which use lattices in order to store partial join results. The performance of LARA-J* is better than HRJN* with respect to access depth but the algorithm induces processing cost which is higher than that of HRJN*. Finger et al. [37] and Schnaitter et al. [95, 96] study the problem of finding tight bounds for terminating the rank-join algorithms. They also proposed the a-FRPA algorithm, a hybrid approach between a tight bound and HRJN*, which has improved performance over HRJN* in low data dimensionality. In higher dimensionality the performance advantage is minimal and and the performance of the algorithm is on the same levels as HRJN*. Martineghi et al. [80] study the problem of joining results produced by different sources on the Web for which the access cost varies. They assume that both sorted and random access are available and propose an algorithm for determining an efficient pulling strategy at compile time which takes into account the access cost for each source. Habich et al. [50] address the problem of increasing the overall performance of multiple top-*k* queries over joins. Their main difference is that while the previous approaches assume that the data are sorted according to the preference queries, they propose a strategy where they avoid sorting the relations for each top-*k* query by using a global sorting for merge-joins of the tuples or by using a variation of the hash-join algorithm. Agrawal et al. [1] discuss the subject of confidence aware rank-join algorithms. Xie et al. [121] study the problem of rank joins with aggregation constraints while in [75] Lu et al. introduce the top-k,m queries. Given a set of groups where each group contains a set of attributes, they study the problem of finding the best combination of attributes. They focus on ranking combinations of attributes and not combinations of objects. Zhang et al. [127] study the problem of finding the best combinations of objects on a graph. Khalefa et al. [63] optimize the performance of preference joins with the use of pruning techniques. Jin et al. study the problem of multi-relational skylines [60] and skylines over equi-joins [61], while Doulkeridis et al. [28] study the problem of rank-join queries over distributed systems.

Our main difference with the aforementioned approaches is that we do not assume that combinations should have a fixed number of elements but combinations of any size can be eligible. In addition, we examine a specific case of joins where all additional objects are combined with a main object. This case of "star"-join has specific characteristics that allow us to improve the performance of the processing of such joins. Moreover, we are considering only the best combination of each main object. This enables us to offer a wider view of the available main objects, which are the objects the user is primarily interested in.

Package recommendation. Our work is also related to package recommendation [85, 93, 120]. Xie et al. [120] study the problem of creating the best package out of a set of items given a specific budget. However, the objects are not related to each other and the problem they address is to create the most attracting package of objects for a user. Combinability of objects is not taken into account and it is assumed that all combinations are possible. Guo et al. [48] try to find packages that are not dominated by other packages. They focus on packages of the same size. Roy et al. [93] suggest a method of constructing combinations based on a central object and a set of satellite objects but they do not take into account preference vectors. Their effort focuses on creating and presenting a number of packages that maximizes the variety of the contained satellite objects and satisfies at the same time a budget constraint.

Our main differences from the aforementioned approaches are that we are focusing on maximizing the preference score of the main objects rather than creating combinations which satisfy a certain constraint. In addition, we do not assume a static combination size but we consider it to be dynamic. Finally, we do not assume that all combinations are possible but we evaluate the join conditions at the same time. None of the aforementioned approaches examine all these conditions simultaneously.

4.3 Problem definition

In this section, we formally define the XTJ_k query and all necessary structures used both for the problem definition and the description of the respective algorithms. Table 4.2 summarizes the main symbols used.

Symbol	**Explanation**
E_M	the main relation
E_i	an *additional relation* to E_M
HE	set of accessed objects of relation E
o	object of E_M
p	object of any relation E_M or E_i
c	a combination of objects
$f_w(p), f_w(c)$	score of an object p and combination c
$m(c)$	the main object of a combination
c_{mp}	the most promising combination
$C(E_M)$	all possible combinations with E_M as main relation
$B(E_M)$	all candidate combinations with E_M as main relation
$XTJ_k(w)$	top-k candidate combinations
$ALT(o)$	set of alternative combinations for a main object o

Table 4.2: Table of symbols

4.3.1 Object combinations

Let D be a database of objects and E_M be a relation in D that is connected to a set of relations $\mathcal{E} = \{E_1, \ldots, E_n\}$ of D. E_M has a set of d real valued attributes $A_{E_M} = \{a_1, \ldots, a_d\}$ and each relation E_i contains a subset $A_{E_i} \subseteq A_{E_M}$ of these attributes, i.e., it holds that $\forall E_i \in \mathcal{E}, A_{E_i} \bigcap A_{E_M} \neq \emptyset$. Each object in a relation $E \in \mathcal{E} \bigcup \{E_M\}$ is represented as a d-dimensional point $p \in \mathbb{R}^d$ where $p[i] \in \mathbb{R}$ if $a_i \in A_E$ and $p[i] = 0$ if $a_i \notin A_E$. We refer to E_M as *main relation* and to the rest of the relations as *additional relations*. The objects of the relations are called *main* and *additional objects*.

Using the main and the additional objects we can form combinations where each combination has exactly one main object and at most one object from each additional relation. We say that an object of the main entity relation $o \in E_M$ and a object $p \in E_i$ of an additional relation E_i are *combinable* if there is a join of the form $o \bowtie p \in E_M \bowtie E_i$.

Definition 4.1. ***Combination.*** *Given a main relation E_M and a set of relations $\mathcal{E}$, we define as a combination a set of objects c such that:*

- $\exists o \in c : o \in E_M$, $|E_M \bigcap c| = 1$,
- $\forall p \in c, p \neq o$ *it holds that* $\exists E_i : o \bowtie p \in E_M \bowtie E_i$ *and*

- $\forall p_i, p_j \in c, i \neq j, p_i \in E_i, p_j \in E_j$ *it holds that* $E_i \neq E_j$.

We use the notation $m(c)$ to denote the main object of a combination c, and $C(E_M)$ to denote the set of all possible combinations that can be formed using E_M as main relation. Note that a main object can participate in many combinations, but each combination has only one main object.

Using the example database in Figure 4.1, if a user wishes to buy a laptop and she is interested in CPU, RAM, SSD size and price, then the main relation of her query is *Laptops* and the additional relations are *Memory* and *SSD*. Any other attributes not specified in the query can be considered irrelevant. The set of objects $\{c_2, m_3, d_3\}$ is a valid combination, while $\{c_1, m_1\}$ is not since c_1 and m_1 are not combinable. Table 4.1 shows some of the combinations in C(Laptops), which is the set of all possible combinations with *Laptops* as the main relation.

4.3.2 Ranking combinations

We can now extend the notion of a top-k query in order to take into account not only single objects but also combinations. We therefore define the *Exploratory Top-k Join* (XTJ_k) query, which returns the top-k combinations with distinct main objects. We consider a user query to be a d-dimensional preference vector w targeted to relation E_M, and each dimension $w[i]$ of the query to represent the importance of the respective attribute to the user. Without loss of generality we assume that $w[i] \geq 0$, $\sum_{i=1}^{d} w[i] = 1$ and if a user is not interested in a specific attribute a_i of the main objects then $w[i] = 0$. Given a preference vector w targeted to the main relation E_M, the score of a combination c is defined as: $f_w(c) = \sum_{j=1}^{d} w[j] \sum_{p \in c}(p[j])$.

An XTJ_k query lists the k main objects with the best combinations and thus each main object can appear at most once in the query's result set. The following definition formally defines the XTJ_k query.

Definition 4.2. ***XTJ$_k$* query.** *Given a main relation E_M, a set of additional relations $\mathcal{E}$, a preference vector w and an integer k, the result set $XTJ_k(w)$ of an Exploratory Top-k Join query is a set of combinations such that:*

- $XTJ_k(w) \subseteq C(E_M)$ *and* $|XTJ_k(w)| = k$,
- $\forall c_1, c_2 \in XTJ_k(w)$ *it holds that* $m(c_1) \neq m(c_2)$,
- $\forall c_1 \in XTJ_k(w), c_2 \in C(E_M) - XTJ_k(w)$ *one of the following necessarily holds:*

– $f_w(c_1) \geq f_w(c_2)$ *or*
– $\exists c' \in XTJ_k(w) : m(c') = m(c_2)$ *and* $f_w(c') \geq f_w(c_2)$.

Returning to our example, Table 4.1 lists the ranked set $C(E_M)$, while the result of an XTJ_2 query are the combinations $\{c_2, m_3\}$ and $\{c_4, m_1, d_2\}$. There is a combination $\{c_2, m_3, d_3\}$ which has a better score than $\{c_4, m_1, d_2\}$, but it is omitted since it shares the same main object (c_2) with the top-1 result.

4.3.3 Theoretical properties

In the following, we present some properties of the combinations that help us to reduce the search space of the XTJ_k query.

A combination that no other tuple can be added to and improve the score of the combination is called *total combination.* A total combination does not necessarily contain objects from every additional relation. Given the fact that in the general case $p \in \mathbb{R}^d$, the score of an additional object could be negative, and the addition of such an object to a combination would make the score of the combination worse. In such cases a total combination may not contain objects from all additional relations. We should note that a main object may also have a negative score, however, a combination should always contain a main object, even if its score is negative.

As mentioned before, each object of the main relation can participate in many combinations. However, for each main object we are interested only in the combination with the best score, i.e., the *candidate combination.*

Definition 4.3. ***Candidate combination.*** *Given a main object o of the main relation E_M, the candidate combination c is a combination such that $\forall c' \neq c$: $m(c') = m(c) = o$ it holds that $f_w(c) \geq f_w(c')$.*

We denote the set of all candidate combinations as $B(E_M)$. Obviously $B(E_M) \subseteq C(E_M)$. Returning to our example, Table 4.1 lists the set $C(E_M)$, while the set of the candidate combinations $B(E_M)$ is indicated with a star (*).

Lemma 4.1. *A candidate combination is total.*

Proof. By contradiction. Assume that the candidate combination c of object o is not total. Then, there exists a combination $c' \neq c$, such that $m(c) = m(c') = o$ and it holds that $c' = c \bowtie p$, where p is an additional object, and also $f_w(c') > f_w(c)$. This contradicts with Definition 4.3. ■

The opposite does not hold. There can be many total combinations with the same main object and no other common object, but only one of them can be candidate combination as well.

Lemma 4.2. *It holds that $XTJ_k(w) \subseteq B(E_M)$.*

Lemma 4.2 is easy to be proven and it shows that it is sufficient to examine only candidate combinations during processing of a XTJ_k query.

4.4 Pull-bound framework

Based on the framework described in Algorithm 1 of Chapter 2, we present a pull-bound framework for XTJ_k queries. The pull-bound framework is based on the assumption that access to the objects of each relation is provided in descending order of score[1]. The score of an object p_i is equal to $f_w(p_i) = \sum_{j=1}^{d} w[i]p_i[j]$ and it is essentially the contribution to the total score of the combination it belongs to. In other words, for any relation, if object p_1 is accessed before p_2, this means that $f_w(p_1) \geq f_w(p_2)$.

The general structure of the family of algorithms that comply with this framework is shown in Algorithm 4. Their difference lies on (a) the bounding technique, which calculates the upper bound of the possible score of any unseen combination, and (b) the pulling technique, which determines the next relation to access.

In the following, we first introduce our pull-bound framework for processing XTJ_k queries. Then, we adapt an existing rank-join algorithm, namely *HRJN** [57], in order to be able to process exploratory top-k joins. Since the calculated bounds play an important role on the behavior of the pulling strategy, we are going to analyze the bounding technique first. We employ the modified HRJN* algorithm as a baseline to compare the performance of our algorithms.

4.4.1 XTJ_k framework

As shown in Algorithm 4, the pull-bound framework consists of a loop that is executed until k join results have been produced and no unseen tuple can produce a join result with better score. In the first step, the next relation to be accessed is selected (line 3) based on a pulling strategy. Given a pulled tuple p from that relation, we update the set of produced combinations (line 6). Finally, the lower bound is set as the score of the k-th join result (line 7), and the upper bound of any unseen join result is computed (line 8) based on the bounding scheme.

[1]Usually this is achieved by the use of multidimensional indexes or materialized views.

Algorithm 4 Pull-Bound Framework

```
Input: $E_M$: main relation
       $\mathcal{E}$: additional relations
Output: $XTJ_k(w)$
 1: $\widehat{B}(E_M) \leftarrow \emptyset$ //Set of produced candidate combinations
 2: while $|\widehat{B}(E_M)| < k$ OR $LB < UB$ do
 3:     $E \leftarrow$chooseRelation($\mathcal{E} \bigcup \{E_M\}$)
 4:     $p \leftarrow E$.pullTuple()
 5:     $HE \leftarrow HE \bigcup \{p\}$ //add p to the accessed objects
 6:     $\widehat{B}(E_M) \leftarrow$update($HE_M, HE_1 \ldots, t, \ldots, HE_n$)
 7:     $LB \leftarrow$kBest($\widehat{B}(E_M)$)
 8:     $UB \leftarrow$upperBound($\widehat{B}(E_M)$)
 9: end while
10: return topK($\widehat{B}(E_M)$)
```

Now, we focus on how the combinations are generated (line 6). Since we are interested only in candidate combinations, the method *update()* combines a newly pulled object with a combination only if that is beneficial for the combination. The generated candidate combinations are maintained as a set $\widehat{B}(E_M)$ which contains the best *seen* combination for each main object. In detail, if the accessed tuple p refers to a main object, the method *update()* finds the best additional objects of the already accessed tuples which are combinable with p, creates the combination and adds it to $\widehat{B}(E_M)$. If tuple p refers to an additional object, then we add this object to all combinations that the tuple can be added to, i.e., to all combinations that p is combinable with the main object of the combination and the addition of p results to an improved score of the combination. In that way we ensure that $\widehat{B}(E_M)$ contains only the best combination of each main object, considering of course only the accessed tuples.

As a result, the set $B(E_M)$ is computed incrementally in Algorithm 4, which means that in worst case where all objects of the relations are accessed, then $\widehat{B}(E_M)$ will be equal to $B(E_M)$. However, in practice, the algorithm will halt much earlier, thus avoiding the cost of materializing the set $B(E_M)$.

4.4.2 Modified HRJN* algorithm

Bounding scheme. Our pull-bound framework evaluates a XTJ_k query by estimating the upper bound of the score that any unseen tuple can produce and terminates when the k-th best join result found is better than the upper bound. Recall that the upper bound (UB_{HRJN*}) of the HRJN* algorithm is the maximum value produced if we combine the worst seen tuple of any relation with the best seen tuples of the remaining relations [57]. For the additional relations, we include the best tuples only if their score is positive and they can increase that way the total score of the combination. We must include however the score of the best tuple of the main relation since its presence in the results is necessary. The bounding scheme of HRJN* is formally described in Equations 4.1-4.3.

$$
\begin{aligned}
UB_{E_M} &= f_w(HE_M[last]) + \sum_{E \in \mathcal{E}} u(f_w(HE[1])) && (4.1)\\
UB_{E_i} &= f_w(HE_M[1]) + f_w(HE_i[last]) + && (4.2)\\
&+ \sum_{\substack{E \in \mathcal{E} \\ E \neq E_i}} u(f_w(HE[1])) \\
UB_{HRJN*} &= \max_{E \in \mathcal{E} \bigcup \{E_M\}} (UB_E) && (4.3)
\end{aligned}
$$

The notation HE denotes the set of accessed objects of relation E and by $HE[1], HE[last]$ we denote the first and last accessed tuples. The function $u(x)$ returns x if $x > 0$ and 0 if $x \leq 0$. The complete algorithm that calculates the bound is described in Algorithm 5.

Theorem 4.1. *(Correctness of bound) The modified version of HRJN* provides a correct solution to the Exploratory top-k join problem.*

Proof. We assume that HRJN* stops after having accessed d_0 tuples for E_M and d_i tuples for each additional relation E_i. Let $c_k = \{E_M[j_0], E_1[j_1], \ldots, E_n[j_n]\}$ be the k-best combination , i.e., $LB = f_w(c_k)$ and let the upper bound be $UB = UB(E_\zeta)$, $E_\zeta \in \{E_M\} \bigcup \mathcal{E}$. Since the algorithm has stopped then the following condition holds:

$$
\begin{aligned}
f_w(c_k) &\geq UB && (4.4)\\
\sum_{E_j[i_j] \in c_k} f_w(E_j[i_j]) &\geq f_w(E_\zeta[d_\zeta]) + \sum_{\substack{E_j \in \mathcal{E} \bigcup E_M \\ E_j \neq E_\zeta}} f_w(E_j[1]) && (4.5)
\end{aligned}
$$

Based on Equations 4.1-4.3 we conclude that Inequality 4.5 holds for any relation $E_\zeta \in \mathcal{E} \bigcup \{E_M\}$. Let us assume that there is a non-total combination

Algorithm 5 HRJN* Bound

```
Input: $E_M$: main relation
       $\mathcal{E}$: additional relations
Output: $UB_{HRJN*}$
 1: $UB$, bound $= -\infty$
 2: for all $E \in \mathcal{E} \bigcup \{E_M\}$ do
 3:   if $E = E_M$ then
 4:     bound=$f_w(HE[last]) + \sum_{E' \in \mathcal{E}} u(f_{\mathbf{w}}(HE[1]))$
 5:   else
 6:     bound=$f_w(HE[last]) + f_w(HE_M[1]) + \sum_{\substack{E' \in \mathcal{E} \\ E' \neq E}} u(f_w(HE[1]))$
 7:   end if
 8:   if bound> $UB$ then
 9:     $UB \leftarrow$ bound
10:     $E$ becomes the next relation to pull from
11:   end if
12: end for
13: return $UB$
```

c' which if combined with an unseen tuple will become better than c_k. Consequently, after joining c' with the unseen tuple it will hold that $f_w(c') > f_w(c_k)$. Since c' contains an unseen tuple, it contains a tuple $E_\tau[d_\tau + x]$ of a relation E_τ. The maximum score of c' is therefore given by either Equation 4.1 if $E_\tau = E_M$ or in the opposite case, by Equation 4.2 where we consider the last tuple to be $E_\tau[d_\tau + x]$. We have assumed that c' is better than c_k, therefore, if we substitute E_ζ with E_τ in Inequality 4.5 then we get that $f_w(E_\tau[d_\tau + x]) \geq f_w(E_\tau[d_\tau])$ which is not true because we are accessing the objects in descending order of score. This contradicts with the assumption that a non-total combination combined with an unseen object may produce a top-k result. We reach therefore the conclusion that no unseen object can produce a better combination after the stopping criterion of HRJN* is satisfied. ■

Pulling strategy. The set of bounds calculated by HRJN* in Algorithm 5 is used to decide which is the next relation to pull from. The intuition indicates that we should pull from the relation that produced the highest bound since this relation plays an important role in the upper bound of the algorithm. If we pull from the other relations, the upper bound will not be reduced significantly and the algorithm will not terminate fast. Therefore the decision on which is the next relation to pull is taken in line 10 in Algorithm 5.

4.5 Exploratory rank-join (XRJN)

In this section, we propose the eXploratory Rank-Join algorithm (*XRJN*), which also follows the pull-bound framework. We propose a tighter bounding scheme (Section 4.5.1), we prove its correctness (Section 4.5.2), and we show that the bounding scheme of *XRJN* has strong theoretical guarantees on its performance, namely that it is instance-optimal (Section 4.5.3). In addition, we present a lazy method to compute the bound more efficiently (Section 4.5.4), and we analyze the complexity of the proposed algorithm (Section 4.5.5). Finally, we present a pulling strategy that is beneficial for the proposed bounding scheme (Section 4.5.6).

4.5.1 Bounding scheme

At a random state of the algorithm, let *HE* denote the objects of a relation E that have been accessed so far, and $\widehat{B}(E_M)$ the set of all combinations that have been created so far. Recall at this point that only one combination per main object is created and that an additional object is added to a combination only if this produces a better score for the combination. There are two bounds we should consider, denoted as UB_{E_M} and UB_{comb} respectively, and our bounding scheme computes their maximum:

$$UB_{XRJN} = max(UB_{E_M}, UB_{comb}) \tag{4.6}$$

The first bound (UB_{E_M}) determines the upper bound of any unseen combination, i.e., any unseen object of the main relation E_M. In the best case, the next object to be accessed from the main relation will be combined with the best objects of the additional relations except for those that have a negative score. Obviously, the upper bound for any unseen object of the main relation is the same with the upper bound calculated by the baseline and its value is calculated by Equation 4.1.

The second bound (UB_{comb}) represents the best score of a seen main product combined with at least one unseen additional object. For any seen main object, there exists exactly one combination in $\widehat{B}(E_M)$. Any retrieved main object that cannot be combined with any of the accessed additional objects will be added to $\widehat{B}(E_M)$, as a combination with a single main object. A combination c in $\widehat{B}(E_M)$ is total based on the seen tuples if its score cannot be improved further. In other words, c is total if for all relations E_i either (a) there exists $p_i \in E_i$ such that $p_i \in c$, or (b) E_i is exhausted (all tuples have been accessed) or $HE_i[last]$ is negative. In the following, we refer to a combination c of $\widehat{B}(E_M)$ that is not total as non-total combination and we refer as *missing relation* of c all relations E_i for which (a) or (b) does not hold.

Let c be a non-total combination and E_i be a missing relation of c, then c can be combined with an unseen object of E_i and therefore the maximum contribution of E_i is equal to $f_w(HE_i[last])$. Thus, the upper bound of the score for a non-total combination c can be computed be adding for every missing relation E_i the score of the last accessed object of E_i. We call *most promising* combination, the non-total combination c_{mp} which has the highest upper bound on its score. The upper bound UB_{comb} of all seen main products is determined by the upper bound of the score of the most promising combination.

Equation 4.7 defines formally the most promising combination while Equation 4.8 calculates the respective upper bound[2].

$$c_{mp} = \underset{\substack{c \in \widehat{B}(E_M) \\ c\ not\ total}}{argmax} \left(f_w(c) + \sum_{\substack{E \in \mathcal{E} \\ E\ missing\ relation}} u(f_w(HE[last])) \right) \tag{4.7}$$

$$UB_{comb} = f_w(c_{mp}) + \sum_{\substack{E \in \mathcal{E} \\ E\ missing\ relation}} u(f_w(HE[last])) \tag{4.8}$$

Example 4.1. *Assume the objects shown in Table 4.3 where E_M is the main relation, E_1, E_2 are the additional relations and we are looking for the top-1 combination. The table shows the scores of the objects according to a given vector w. The two signs (*,+) show two possible combinations. We assume that we read the tuples in a round-robin fashion. The* id *of each tuple is the row number and the relation letter it belongs to (e.g., $E_M[1], E_1[3]$,etc.). After having read the first row, we have one combination $c_1 = \{E_M[1]\}$ and the upper bound UB is equal to $UB = f_w(E_M[1]) + f_w(E_1[1]) + f_w(E_2[1]) = 22$. The lower*

[2]We should note that this bound is the same for all additional relations, Therefore we call it UB_{comb} because it depends on the current combinations and the last objects accessed.

id	E_M	E_1	E_2
1	10^*	7^+	5
2	10	6	4
3	8	6^*	3
4	7^+	5	$3^{*,+}$
5	6	5	3
6	5	5	3
7	4	3	2
8	3	3	2
9	2	2	1
10	1	1	1

Table 4.3: Example

bound is the best score of any combination found so far which is the score of the main object $E_M[1]$, thus $LB = f_w(E_M[1]) = 10$.

When the second row is read, the combination $c_2 = \{E_M[2]\}$ is formed and the upper bound of any unseen combination involving any unseen main object is still equal to $UB_{E_M} = 22$. The most promising combination is $c_1 = \{E_M[1]\}$ and the upper bound of its score is equal to $UB_{comb} = f_w(c_1) + f_w(E_1[2]) + f_w(E_2[2]) = 20$. At this point we are not considering the tuples $E_1[1], E_2[1]$ because if any of them were combinable with $E_M[1]$ or $E_M[2]$ the combination would have been formed. Of the two bounds we pick the maximum and therefore the upper bound is equal to $UB = 22$.

Our algorithm continues by creating the combination $c_3 = \{E_M[3]\}$ and then the combination c_1 is updated and becomes equal to $c_1 = \{E_M[1], E_1[3])\}$. After the third row is read, the most promising combination is c_1 and therefore the upper bounds are formed as following: $UB_{E_M} = 20, UB_{comb} = 19$. The lower bound LB is now equal to $LB = 16$ due to the update of c_1.

After the fourth row has been read then we have that $UB_{E_M} = 19$. The combinations created are $c_1 = \{E_M[1], E_1[3], E_2[4]\}$, $c_2 = \{E_M[2]\}, c_3 = \{E_M[3]\}$ and $c_4 = \{E_M[4], E_1[1], E_2[4]\}$. The lower bound is now equal to $LB = 19$. The most promising combination is c_2 since both c_1 and c_4 are total. The upper bound for the combinations UB_{comb} is is equal to $UB_{comb} = 18$ and therefore $UB = UB_{E_M} = LB$ and so the algorithm stops.

According to the HRJN bound, the upper bound at the fourth row would be equal to $UB_{E_1} = f_w(E_M[1]) + f_w(E_1[4]) + f_w(E_2[1]) = 20$ and therefore extra*

Algorithm 6 Bounding Scheme of XRJN

```
Input: $E_M$: main relation
       $\widehat{B}$: set of all generated combinations
Output: $UB_{XRJN}$
 1: $UB_{E_M} \leftarrow f_w(HE_M[last]) + \sum_{E \in \mathcal{E}} u(f_w(HE[1]))$
 2: for all $c \in \widehat{B}(E_M)$ and $c$ not total do
 3:   $c.UB \leftarrow c.score$
 4:   for all $E \in$ missingRelations of $c$ do
 5:     $c.UB \leftarrow c.UB + u(f_w(HE[last]))$//max. possible score for each combination
 6:     if $UB_{comb} < c.UB$ then
 7:       $c_{mp} \leftarrow c$
 8:       $UB_{comb} \leftarrow c.UB$
 9:     end if
10:   end for
11: end for
12: return $max(UB_{E_M}, UB_{comb})$
```

tuples would be read until $E_1[7]$ and $E_2[7]$ were accessed where the upper bound would become equal to $UB = 19$. At this point it is clear why the new bounding technique offers a tighter estimation of the bound. First, the HRJN bound technique is overestimating the score of the unseen combinations because it uses uncombinable tuples, and second, because it does not exclude main objects of total combinations whose score cannot be improved.*

4.5.2 Algorithm and correctness

Algorithm 6 calculates the upper bound of the score for any unseen combination. In line 1, we calculate UB_{E_M} which is the upper bound of any combination involving any *unseen* objects of the main relation.

In line 2, we evaluate the already created non-total combinations which can be combined with any unseen tuples. For each non-total combination c in $\widehat{B}(E_M)$ we calculate the upper bound of its score by adding the score of the last seen tuples for all missing relations of c. The combination with the highest upper bound of its score ($c.UB$) is the most promising combination (lines 2–11). UB_{comb} is equal to the score of the most promising combination. The upper bound UB returned by the algorithm is the maximum of these two bounds (line 12).

Theorem 4.2. *(Correctness of bound) The XRJN algorithm provides a correct solution to the XTJ_k query.*

Proof. Let us assume that the algorithm has halted, after having accessed d_0 tuples for the relation E_M and d_i tuples for each relation $E_i \in \mathcal{E}$, therefore $UB \leq LB$. Let now c' be an unseen combination for which it holds that $f_w(c') > LB$.

If the unseen combination c' contains an unseen main object, then the score of the combination will be at most equal to UB_{E_M}, thus $f_w(c') \leq UB_{E_M}$. Since $UB_{E_M} \leq UB \leq LB$ it holds that $f_w(c') \leq LB$ which contradicts with the assumption $f_w(c') > LB$. The contradiction that we have reached is due to our assumption that c' contains an unseen main object and it is better than LB.

Alternatively, let us assume that unseen combination c' contains an already accessed main object, which means that there exists a non-total combination $c \in \widehat{B}(E_M)$ that will be combined with at least one unseen additional object and produce c'. In the following we assume that c has been combined with an unseen object from only one relation E_j and c' was created. Similarly, we can prove the general case of more than one relations. Let $E_j[d_j + x]$ be the first combinable object with c. Then the score of c' when joined with the unseen tuple will be equal to: $f_w(c') = f_w(c) + f_w(E_j[d_j + x]) \leq f_w(c) + f_w(E_j[d_j])$. From Equation 4.8 we conclude that $f_w(c') \leq UB_{comb} \leq UB \leq LB$. However in the beginning we assumed that $f_w(c') > LB$ and this is a contradiction. We conclude that an unseen combination cannot have a greater score than the score of the k-th combination in $\widehat{B}(E_M)$, thus our algorithm returns always the correct result set. ■

4.5.3 Instance optimality

Instance optimality is defined by Fagin et al. [34] as follows. Given a class of algorithms $\mathcal{A}$ and a set of databases $\mathcal{D}$, an algorithm $A \in \mathcal{A}$ is instance-optimal if $\forall B \in \mathcal{A}$ and $\forall D \in \mathcal{D}$ it holds that $\text{cost}(A, D) = O(cost(B, D))$. This means that there are constants c_1, c_2 such that $\text{cost}(A, D) \leq c_1 \text{cost}(B, D) + c_2$. Constant c_1 is referred to as *optimality ratio.*

Lemma 4.3. *HRJN* is not instance optimal for the XTJ_k query.*

Proof. Based on the definition of instance optimality it is sufficient to show that the cost of HRJN* is not bounded for one instance database D' compared to an algorithm A. Thus, we construct a dataset for which the cost of HRJN* is not bounded compared to XRJN. Denoting the database of Table 4.3 as D, we consider a database D'. Each relation E_M, E_1, and E_2 of D' has the same first 4 tuples as D. Thus, XRJN will return the correct answer after accessing the first 4 tuples of each relation but HRJN* does not terminate and will continue reading more tuples. Let us assume that the relations of D' contain more than 4 tuples such that $E_M[i] = E_M[4]$, $E_1[i] = E_1[4]$, $E_2[i] = E_2[4]$ for $i > 4$. HRJN* has to access at least all tuples of relation E_2 before terminating, while XRJN needs to access only the first four rows. Since relation E_2 can be arbitrarily long the cost of HRJN* cannot be bounded and therefore HRJN* is not instance optimal. ■

Theorem 4.3. *XRJN is instance optimal with optimality ratio $n + 1$ where n is the number of additional relations.*

Proof. Let A be a random deterministic algorithm solving correctly the XTJ_k query for a vector w. The algorithm halts after having accessed d_0 tuples for E_M and d_i tuples for each additional relation E_i. We define c_k to be the k best candidate combination discovered by A with score $f_w(c_k)$ and d_{max} to be equal to $d_{max} = \max\limits_{0 \leq i \leq n}(d_i)$. We will show by contradiction that XRJN will halt after accessing at most d_{max} tuples from each relation.

Let us assume that XRJN has accessed d_{max} tuples from each relation and has not halted. At this point XRJN has processed at least all combinations evaluated by A and therefore it holds that $LB_{\text{XRJN}} = f_w(c_k)$. Under the assumption that XRJN has not halted, there are two cases to be examined.

The first case is that the upper bound of XRJN at that step is defined by an unseen object of the E_M i.e., $UB_{\text{XRJN}} = UB_{E_M}$ and it holds that $LB_{\text{XRJN}} < UB_{\text{XRJN}}$ and since $LB_{\text{XRJN}} = f_w(c_k)$ it also holds that $f_w(c_k) < UB_{\text{XRJN}}$. As algorithm A is deterministic, it must halt at the same step for all instances of relations that have the same seen tuples HE_M and HE_i. We can construct a relation E_M such that $HE_M[d_0+1]$ is combinable with the first tuples of all additional relations $HE_i[1]$ and $f_w(E_M[d_0+1]) = f_w(E_M[d_0])$. For the combination c' defined by $\{HE_M[d_0+1], HE_1[1], \ldots, HE_n[1]\}$ it holds that $f_w(c') = UB_{\text{XRJN}}$, thus $f_w(c_k) < f_w(c')$. Therefore A has halted incorrectly, which leads us to a contradiction.

In the second case it holds that $UB_{\text{XRJN}} = UB_{comb}$ and $LB_{\text{XRJN}} < UB_{\text{XRJN}}$. Let c_{mp} be the most promising combination found by XRJN and $o = m(c_{mp})$ be the main object of c_{mp}. An instance of our database D can be constructed in way that $\forall E_i : \nexists p_i \in c_{mp}, p_i \in E_i$ it holds that $f_w(E_i[d_{max}]) = f_w(E_i[d_{max}+1])$ and all $E_i[d_{max}+1]$ tuples are combinable with o. In this case, when $d_{max}+1$ tuples have been read from all relations a new combination c'_{mp} is produced by updating c_{mp} and adding the newly pulled tuples. It holds that $f_w(c'_{mp}) = UB_{\text{XRJN}}$ and therefore also $f_w(c_k) < f_w(c_{mp})$. Thus, c'_{mp} belongs to the result set and A has halted incorrectly, which leads us to a contradiction.

We conclude that XRJN will halt after accessing at most d_{max} tuples from each relation. Thus, the cost of XRJN in terms of accessed tuples is at most $\text{cost}(\text{XRJN}, D) = (n+1) * d_{max}$, while the cost of algorithm A is $\text{cost}(A, D) = \sum_{0 \le i \le n} d_i$. We can derive that $\text{cost}(\text{XRJN}, D) \le (n+1) * \text{cost}(A, D)$ since $d_{max} \le \sum_{0 \le i \le n} d_i$. Hence XRJN is instance optimal with optimality ratio $n+1$. ■

4.5.4 Lazy upper bound evaluation

The processing cost of the XRJN* upper bound is determined by the size of the $\widehat{B}(E_M)$ set as it is necessary to search the entire set every time we need to find the most promising combination. However, we can reduce the overall cost of the calculation if we postpone the accurate calculation of the upper bound until it is absolutely necessary. To achieve that, we calculate the highest possible score of a combination when the combination is updated and we update the most promising combination if necessary. This approach does not take into consideration the fact that when a tuple is read, it affects the maximum possible score of possibly all non-total combinations and not only the updated ones. As a result, it is possible that the most promising combination is a combination which was not updated and therefore UB_{comb} and consequently UB_{XRJN} are underestimated. The solution to this problem is to calculate the upper bound UB_{comb} without updating the most promising combination until $UB_{XRJN} \le LB$. When the inequality holds, Algorithm 6 is executed the upper bound is accurately calculated and if it still holds that $UB_{XRJN} \le LB$, XRJN* halts.

4.5.5 Cost and complexity analysis

Both algorithms described so far follow the pull and bound paradigm presented in Algorithm 4. The I/O cost is mainly determined by the depth each relation is accessed, therefore we expect XRJN to access fewer objects than HRJN* because XRJN provides a better estimation of the upper bound than HRJN*.

The processing cost of each repetition of the pull and bound framework loop is determined by the cost of the lower and upper bound calculations and the cost of updating the candidate combinations. The cost for updating the combinations is $O(1)$ for HRJN*. The cost for updating a combination for XRJN is $O(|\mathcal{E}|)$ because the update of a combination c is followed by a calculation of the maximum possible score of c performed by the lazy evaluation. The top-k results are stored in a priority queue. The cost of updating the queue is equal to $O(logk)$ while the cost of checking the first element of the queue is equal to $O(1)$. Each time a combination is updated it is necessary to verify that the combination is not inserted twice in the queue. The check is performed in $O(1)$ time using a hash table on the elements of the queue. The cost of the upper bound calculation for HRJN* is $O(|\mathcal{E}|)$ as we have to calculate $|\mathcal{E}| + 1$ upper bounds for each tuple update. The cost for XRJN is in the best case equal to $O(|\mathcal{E}|)$ which is the cost of updating the upper bound for the most promising combination while in the worst case the cost is equal to $O(|\widehat{B}_{\text{XRJN}}(E_M)|)$ which is the cost of identifying the most promising combination when the lazy upper bound evaluation produces an upper bound smaller than the lower bound and there is at the same time a large number of non total combinations. The latter however will rarely be the case because frequent updates will create fast total combinations which can be ignored during the upper bound evaluation while in case of infrequent updates the most promising combination is unlikely to change.

The loop in the pull and bound framework is repeated at most $|\mathcal{E}{+}1||\widehat{B}(E_M)|$ times for each algorithm. Therefore, the overall cost for HRJN* is equal to $O(|\widehat{B}_{\text{HRJN*}}(E_M)|(|\mathcal{E}| + logk))$ while for XRJN it is in the best case equal to $O(\widehat{B}_{\text{XRJN}}(E_M)|(|\mathcal{E}| + logk))$. In the worst case the cost for XRJN is equal to $O(|\widehat{B}_{\text{XRJN}}(E_M)|^2)$ as the dominating cost is that of the upper bound calculation. The complexity analysis indicates that the processing cost of both algorithms is highly affected by the size of the combinations $\widehat{B}(E_M)$ created by each algorithm. XRJN is expected to generate a significantly smaller number of combinations and therefore we expect XRJN to be more efficient than HRJN* despite the fact that each processing step of XRJN has a higher processing cost than that of HRJN*.

In the following, we will introduce an improved pulling strategy which will reduce the cost of upper bound calculation and ultimately improve the performance of XRJN.

4.5.6 The XRJN* pulling strategy

The objective of the pulling technique is twofold. The first goal is the early convergence of the lower and upper bound in order to minimize the access depth (number of accessed objects) for each relation. The lower bound is increased

Algorithm 7 XRJN* Pulling strategy

```
Input: $E_M$: main relation
       $\mathcal{E}$: additional relations
Output: $E \in \mathcal{E} \bigcup E_M$: next relation to pull from
 1: if $UB_{E_M} > UB_{comb}$ or $c_{mp} = null$ then
 2:   return $E_M$ //$c_{mp} = null$: no not total combinations
 3: end if
 4: $R \leftarrow E_M$
 5: max $\leftarrow -\infty$
 6: for all missing relations $E$ of $c_{mp}$ do
 7:   $u \leftarrow$# of non-total combinations not combined with $E$
 8:   if $u >$ max then
 9:     max$\leftarrow u$
10:     $R \leftarrow E$ //relation with the highest # of uncombined tuples
11:   else if max=$u$ then
12:     if $f_w(HE[last]) > HR[last]$ then
13:       $R \leftarrow E$ //in case of tie choose the relation with the best last seen score
14:     end if
15:   end if
16: end for
```

by the formation of total combinations and is stabilized as soon as the top-k combinations have been formed. The algorithm however, will not halt as soon as the top-k combinations are discovered, but when it is certain that no better combinations can appear. This will be ensured by the decrease of the upper bound. The upper bound is affected by the formation of total combinations since the main objects that participate in total combinations can be excluded from the calculation. The upper bound is also affected by the scores of the last accessed objects. Therefore we should aim at accessing first the relations that have high score according to the user's preferences.

The second goal of the pulling technique is to reduce the processing cost of calculating the upper bound and updating the already formed combinations. As mentioned before, the processing cost of both procedures is determined by the number of non-total combinations existing in each step of the algorithm. Therefore the goal of the pulling technique should be to pull objects from the relations in such order that the number of non total combinations is minimized.

Based on the above observations we propose the pulling strategy described in Algorithm 7. We refer to this variation of XRJN as XRJN*. If $UB_{E_M} > UB_{comb}$, the algorithm reads from the main relation, while in the opposite case it reads from an additional relation. In this way, the upper and lower bound converge faster, as on each step the highest upper bound is reduced. If the algorithm chooses to read from an additional relation, it examines only the additional relations which can improve the score of the current most promising combination c_{mp}, i.e., it examines only the missing relations of c_{mp}. The algorithm selects the additional relation with the highest number of uncombined non-total combinations while ties are solved by choosing the relation with the highest last seen score $f_w(HE[last])$. By reading from a relation not combined with c_{mp}, it is ensured that the maximum score of c_{mp} and therefore UB_{comb} will be updated and therefore the upper and lower bound will converge more. At the same time, the algorithm tries to maximize the number of non-total combinations which will be updated. Updating a large number of non-total combinations leads to the formation of total combinations and the increase of the lower bound, which forces the upper and lower bounds to converge. Moreover, the fewer non-total combinations exist, the smaller the processing cost of UB_{comb} will be.

4.6 Providing more results

In this section, we generalize the proposed XTJ_k query, by retrieving k groups of combinations, where each group contains combinations of the same main object. Given a user-defined parameter m, we can extend the notion of the XTJ_k query and present to the user a group of top-m alternative combinations for each main object. Essentially, we relax the constraint of presenting only one combination per main object, and present k groups of combinations. This generalization is motivated by practical applications, where a user is interested in one or more main objects and wishes to explore more options regarding those objects.

To this end, we define as set of *alternative combinations* $ALT(o)$ the m combinations of o with the highest scores, while the score of $ALT(o)$ is defined as the score of the best combination of o. Then, the generalized $XTJ_{k,m}$ query returns the k sets $ALT(o)$ with the highest score.

Viewed from a different perspective, each main object forms a group of combinations, and an XTJ_k query identifies k such groups of combinations and presents the best combination of each group. Under this perspective, each

Algorithm 8 Update Combination

```
Input: Object $o$, Tuple $t$, $E$: relation of $t$
 1: $ALT' \leftarrow \emptyset$
 2: if $c \bigcap E = \emptyset$ and $f_w(t) > 0$ then //t is the first combinable tuple of E with o
 3:   for all $c \in ALT(o)$ do
 4:     $c' \leftarrow c \bigcup \{t\}$
 5:     $ALT' \leftarrow ALT' \bigcup \{c'\}$
 6:   end for
 7: else
 8:   $c \leftarrow$ replace$(TOP_1(ALT(o)), t, E)$ //replace the respective tuple of c with t
 9:   if $f_w(c) > f_w(TOP_m(ALT(o)))$ then //if the new combination will be in the top-m alternatives
10:     for all $(c' \in ALT(o))$ and $(c' \bigcap E = c \bigcap E)$ do //for all c' that contain the best combinable tuple of E
11:       $c' \leftarrow replace(c', t, E)$
12:       $ALT' \leftarrow ALT' \bigcup \{c'\}$
13:     end for
14:   else
15:     mark $c$ as finished for relation $E$
16:   end if
17: end if
18: $ALT(o) \leftarrow TOP_m(ALT(o) \bigcup ALT')$
```

candidate combination is the best combination of the group it belongs to, and it can be viewed as the *primary combination* in order to be distinguished from the remaining alternative combinations of the group. Each group contains a unique primary combination, and therefore, given a main object o and the respective primary combination c, we refer to each group by referring to the primary combination.

In order to be able to discover a group of m combinations for each main object of the result set, both the algorithm updating the combinations and the stopping criterion of the algorithm need to be modified. When a new main object is accessed, a new combination c is created as well as a set of alternative combinations $ALT(o)$ with c being its only element. When a new additional tuple is accessed, the sets of alternative combinations $ALT(o)$ of all main objects o that are combinable with t need to be updated.

Algorithm 8 describes the update procedure of the set of alternative combinations $ALT(o)$ of a main object o, when an additional tuple is added. We denote as $TOP_i(ALT(o))$ the i-th combination sorted by their score. Given a main object o and a new accessed tuple t of a relation E, if $TOP_1(ALT(o))$ does not contain a tuple from E, then t is combined with $TOP_1(ALT(o))$. If the result produces a combination with higher score ($f_w(t) > 0$), then all combinations in $ALT(o)$ need to be updated (line 4) since t is the first combinable tuple of E. Thus, the new tuple t is also used to generate a set of new alternative combinations $ALT(o)$ by adding t to all existing alternative combinations. The new set of combinations $ALT(o)$ consists of the already existing and the newly generated combinations and only the top-m elements are maintained.

In the opposite case, when c already contains a tuple from E, a different approach is followed. First, we create a new combination by replacing the tuple of $TOP_1(ALT(o))$ that belongs to relation E with the new tuple t. The score of this combination is an upper bound of the score of any combination that contains t. If this score is smaller than the score of the m-th already retrieved combination, then no combination that contains t can be added to $ALT(o)$. In this case, $ALT(o)$ is not modified. In addition, we do not need to access any more tuples of E, as the remaining tuples will create combinations with worse score. If a tuple $t \in E$ does not create an alternative combination which belongs to the $TOP_m(ALT(o))$ set, then no other tuple of E needs to be combined with c and therefore we consider c to be *finished* for E. Otherwise, given a new tuple $t \in E$ the algorithm creates new alternative combinations by replacing the respective tuple of E in all combinations c' of $ALT(o)$ that share the same tuple of E with $TOP_1(ALT(o))$ (for which it holds that $c \bigcap E = c' \bigcap E$, i.e., the alternative combinations which contain the same tuple of E as c does). When all updates have been made, only the top-m combinations need to be kept.

Naturally, the stopping criterion in Algorithm 4 has to be altered. In more detail, once the lower bound becomes higher than the upper bound, the algorithm stops updating the bounds and accesses the additional relations until the top-m alternative combinations for each main object have been found. The pulling strategy is modified in order to facilitate the creation of the alternative combinations after the k objects with the highest score have been found. At each pulling step, the algorithm chooses a random main object o which is not finished for at least one additional relation and pulls a tuple from a random additional relation for which o is not finished for. The main relation does not need to be accessed any longer as the k best products have already been found.

4.7 Experimental evaluation

In this section, we present the results of the experimental evaluation. All algorithms were implemented in Java and the experiments run on an AMD Opteron 4130 Processor (2.60GHz), with 32GB of RAM and 2TB of disk.

Datasets and metrics. For the dataset D, we used both synthetic and real data collections. For the synthetic data we used one uniform and one Zipfian distribution. In particular, for the uniform distribution all object values for all relations and dimensions were generated independently using a uniform distribution generator. Each additional relation has a random subset of attributes of the main relation and contains at least one positive and one negative attribute but in total it contains no more than $d-1$ attributes where d is the number of attributes of the main relation. Each additional relation has a unique combination of attributes. Each additional relation has also a joining attribute which does not participate in the ranking of each object while the main relation has $|\mathcal{E}|$ joining attributes, one for each additional relation. The values for the joining attributes are decided based on the join selectivity value σ. For two relations L, R the join selectivity is equal to $\sigma = |L \bowtie R|\,|L \times R|^{-1}$ [49]. If a joining attribute has σ^{-1} different values, then the join contains $\sigma^{-1}\sigma^2|L|\,|R| = \sigma|L|\,|R|$ tuples which gives us a join selectivity of σ.

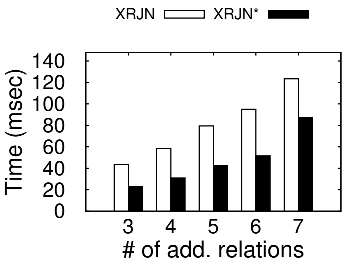
All positive attributes of the main and the additional relations were normalized in the interval $[0, 10000]$ while the negative attributes of the main relation were normalized in the interval $[-10000, 0]$. The negative attributes of each additional relation were scaled according to the number of attributes each relation had, in order to make the cost of an object proportional to the potential improvement of the main object. In particular, for an additional relation E_i the negative attributes take values in the interval $[-10000\,|A_{E_i}|\,|A_{E_M}|^{-1}, 0]$ where $|A_{E_i}|$ is the number of attributes included in the relation. Although this might seem counter-intuitive, it is quite common for the attributes of accessory objects to have a value range of positive attributes similar to the respective attribute of the main object while their cost is lower. For instance the capacity of hard disks as separate components has similar range as the capacity of disks in laptops. The price of a hard disk however, is lower than a laptop's price carrying a similar disk. Finally, the cardinality of each additional relation is equal to the cardinality of the main relation.

For the Zipfian distribution we used the generator provided by the Apache Commons project[3]. The datasets were generated by giving as parameters the maximum value of an attribute (1000) and the value of the exponent characterizing the distribution. The positive and negative attributes of all relations were generated similarly to the uniform distribution.

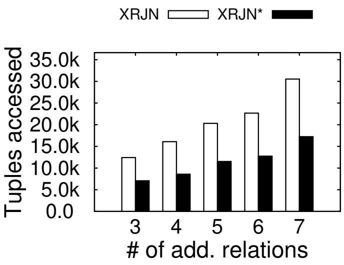
In addition, we used the real datasets HOUSE[4] (Household) and NBA[5]. HOUSE consists of 127930 6-dimensional tuples, representing the percentage of an American family's annual income spent on 6 types of expenditure: gas, electricity, water, heating, insurance, and property tax. NBA consists of 17265 5-dimensional tuples, representing a player's performance per year. The attributes are average values of: number of points scored, rebounds, assists, steals and blocks. In both relations all attributes were normalized in the interval $[0, 10000]$. In each experiment random attributes were considered as negative attributes taking values in the interval $[-10000, 0]$. Each additional relation was created by selecting a random subset of the scoring attributes of the main relation and copying the respective data. Similarly to the uniform distribution, the negative attributes were scaled according to the number of attributes of each additional relation and each additional relation has at least one positive and one negative attribute and at most $d - 1$ attributes in total, while each relation has a unique subset of attributes.

The metrics under which we evaluated the implemented algorithms were: a) execution time required by each algorithm, and b) total tuples accessed (depth). We should stress that we do not focus our performance analysis on the size of the data as the performance of the algorithms depends mainly on the number of additional relations, the join selectivity and the distribution of the values of the relations' attributes. We employed a best-case scenario regarding I/O accesses where relations are sorted for each query, stored on the disk, and accessed sequentially. This strategy minimizes the cost of the I/O accesses and allows us to study the minimum performance difference of the proposed algorithms. In practice, access to each relation will be achieved through other means such as materialized views [40] or multi-dimensional indexes [103], which will induce higher cost in terms of I/O and increase the performance gap between HRJN* and XRJN.

[3]http://commons.apache.org/proper/commons-math/
[4]http://usa.ipums.org/
[5]http://www.databasebasketball.com/



(a) Time



(b) Accessed tuples

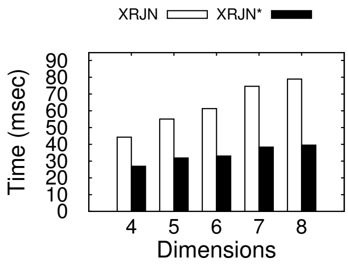
Figure 4.4: Pulling strategy evaluation: varying $|\mathcal{E}|$

Experimental procedure. We run a series of experiments varying the parameters of a) the number of additional relations ($|\mathcal{E}|$) in the interval [3-7], b) dimensionality (d) in the interval [4-8], c) number of returned results (k) in the interval [5-100], d) selectivity (σ) [0.001-0.05], and e) the number of negative attributes [1-3]. For the Zipfian distribution we varied the value of the characteristic exponent s in the interval [0.1-1.0]. Each experiment was run under 5 different dataset instances and 100 queries were used for evaluating the performance of the algorithms.

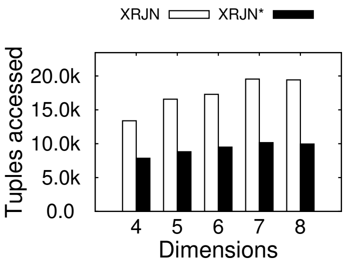
The default setup for the experiments was: $|\mathcal{E}| = 5$, $d = 6$, $|E_M| = 100K$, $k = 10$, $\sigma = 0.001$ and each relation has one negative attribute. The number of preference queries for each setting was equal to $|W| = 100$. Both the dataset and the preferences set followed the uniform distribution.

4.7.1 Pulling technique evaluation

The first series of experiments focuses on evaluating the pulling technique described in Section 4.5 and here we compare XRJN against XRJN*. XRJN uses round robin as pulling strategy, while XRJN* uses the pulling strategy described in Algorithm 7. Figures 4.4 and 4.5 indicate that XRJN* provides an advantage both in processing time and number of accessed tuples. XRJN* pulling strategy forces the upper and lower bound to converge faster, by prioritizing the update of the most-promising combination and by aiming to reduce the highest upper bound. However, the aggressive update of the most-promising combination induces more frequent invocation of the exact upper bound calculation. As a result, the processing-time gain is smaller than the access-depth gain,

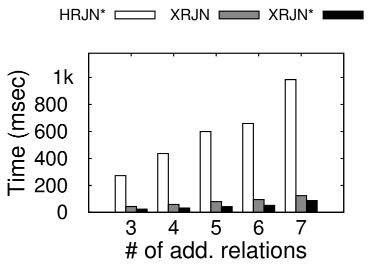


(a) Time

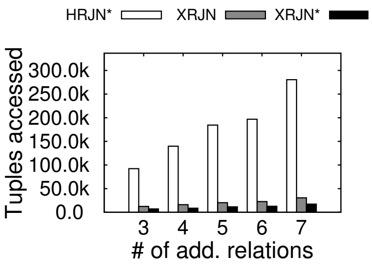


(b) Accessed tuples

Figure 4.5: Pulling strategy evaluation: varying d



(a) Time



(b) Accessed tuples

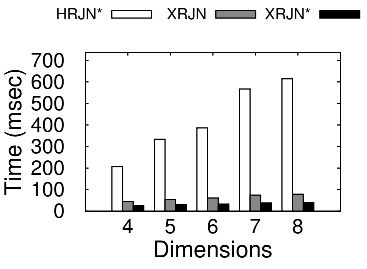
Figure 4.6: Sensitivity analysis: varying $|\mathcal{E}|$

fact that becomes more obvious as the number of additional relations increases (Figure 4.4).

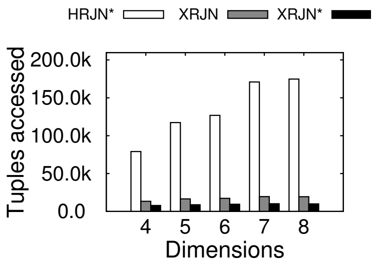
4.7.2 Sensitivity analysis

In this section, we provide a detailed sensitivity analysis by varying different parameters that influence the performance of our proposed algorithms. We start by comparing the most important parameters, namely the number of additional relations, the dimensionality of the relations, the number of returned results and the join selectivity of the relations. In addition we examine the performance of the algorithms on a Zipfian distribution.

Varying $|\mathcal{E}|$. As we increase the number of additional relations, the number of accessed tuples and the processing time of all approaches increase. Figure 4.6

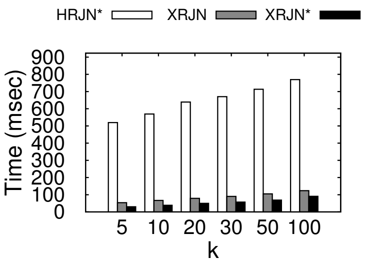


(a) Time

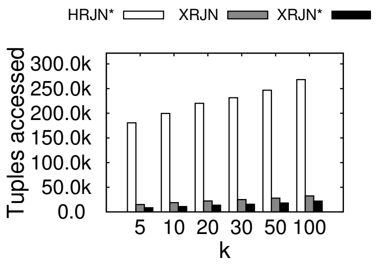


(b) Accessed tuples

Figure 4.7: Sensitivity analysis: varying d



(a) Time

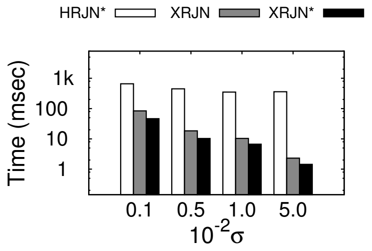


(b) Accessed tuples

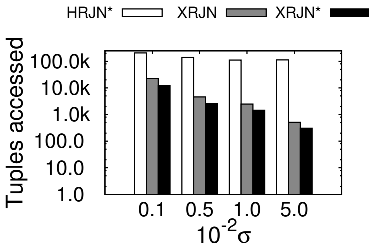
Figure 4.8: Sensitivity analysis: varying k

indicates that XRJN* access consistently less objects than HRJN* and XRJN and the performance of XRJN* is less affected by the increase of the number of additional relations. Due to the improved bounding technique XRJN* and XRJN access nearly an order of magnitude less tuples than HRJN*. The performance of HRJN* is dependent both on the convergence of the upper and lower bound and on the rate the formed combinations become total. If there are many non total combinations then for each accessed tuple there will be an increased processing cost for finding the combinations that the tuple can be added to. On the other hand, XRJN* aims at creating complete combinations fact that helps the lower bound to increase fast and also reduces the updating cost.

Varying dimensionality. Similar conclusions can be drawn when we vary the dimensionality. Figure 4.7 indicates that XRJN and XRJN* are nearly an



(a) Time



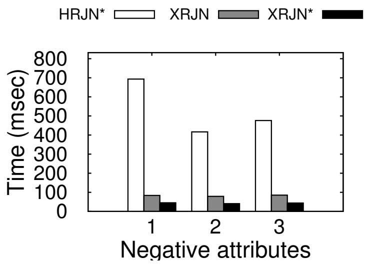
(b) Accessed tuples

Figure 4.9: Sensitivity analysis: varying σ (log scale)

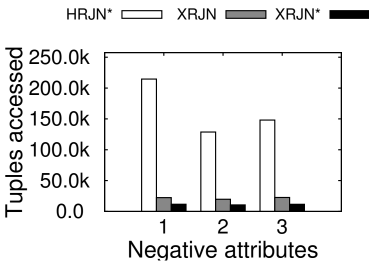
order of magnitude more efficient than HRJN* with respect to time and depth access.

Varying k. The same conclusions hold when we change the number of returned results. Figure 4.8 shows the performance of all algorithms. It is noteworthy that the processing cost for HRJN* is increasing significantly as k is increased. The processing cost for XRJN* also increases but with a slower rate and in all cases it remains nearly an order of magnitude less than HRJN*. The same applies for the access depth, where HRJN* accesses more tuples as k increases. Except for the increased number of accessed tuples, the increase of k causes more combinations to be created and evaluated. HRJN* and XRJN create an arbitrary number of non-total combinations which have to be maintained and updated when a new tuple is accessed. XRJN* on the other hand minimizes the effect of the increased number of non-total combinations through its pulling strategy and therefore is less affected by the increase of the result set size.

Varying selectivity σ. As expected, join selectivity plays an important role in the performance of both algorithms. As Figure 4.9 indicates, when the value of join selectivity increases the performance gap between the HRJN* and XRJN* increases as well. The reason lies in the fact that XRJN exploits the fact that total combinations are formed faster as selectivity increases by considering only non total combinations in the upper bound calculation. In addition, frequent combination updates allow XRJN* to reduce the cost of upper bound calculation as the size of $\widehat{B}_{\text{XRJN}^*}(E_M)$ is small for high selectivity values. In total, XRJN* benefits from higher selectivity values in two ways; the upper bound of XRJN*

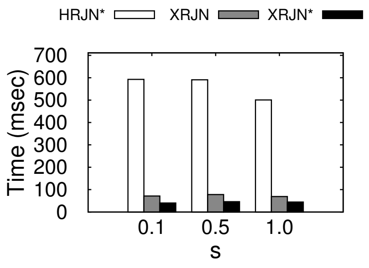


(a) Time

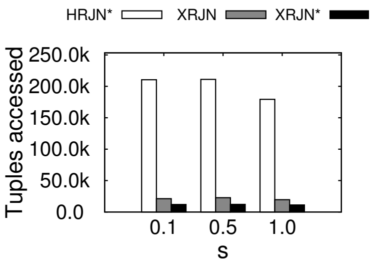


(b) Accessed tuples

Figure 4.10: Sensitivity analysis: varying # of negative attributes



(a) Time

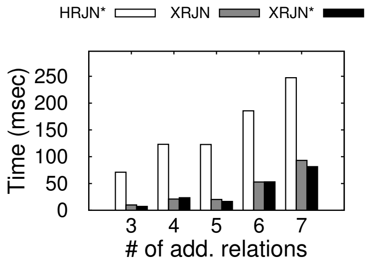


(b) Accessed tuples

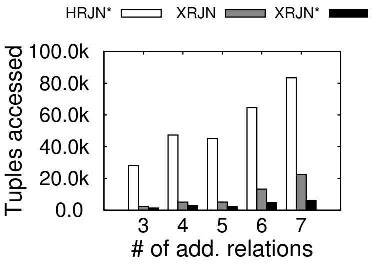
Figure 4.11: Sensitivity analysis: Zipfian distribution

converges to the lower bound much faster than the upper bound of HRJN* and the cost of the upper bound calculation drops as selectivity rises.

Varying the number of negative attributes. Figure 4.10 illustrates the performance of all algorithms as we vary the number of negative attributes. When the number of negative attributes increase, the number of tuples which cannot improve a combination increase as well. Naturally, all algorithms benefit from that fact. HRJN* is affected more than XRJN and XRJN* because when an additional tuple with negative score is read, the non accessed part of the relation can be safely discarded. The upper bound of XRJN allows both XRJN and XRJN* to terminate before the tuples with negative score are accessed and therefore their performance is not affected significantly by the number of negative attributes. They remain however in all cases significantly more efficient than HRJN*.

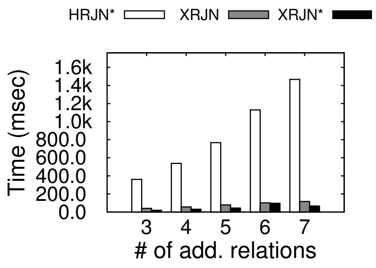


(a) Time

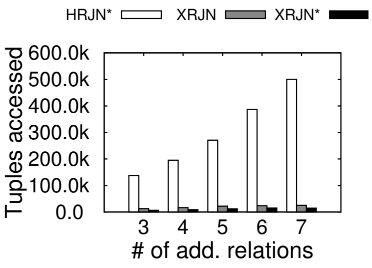


(b) Accessed tuples

Figure 4.12: NBA: varying $|\mathcal{E}|$



(a) Time

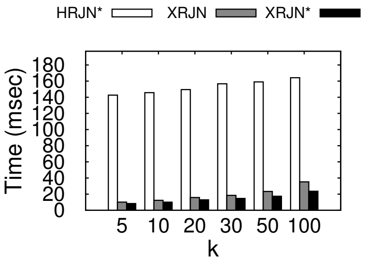


(b) Accessed tuples

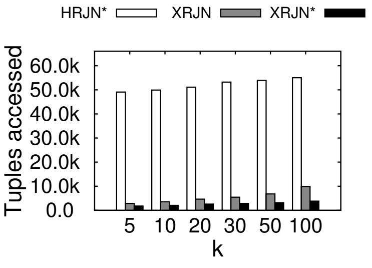
Figure 4.13: HOUSE: varying $|\mathcal{E}|$

Zipfian distribution. We evaluated our algorithms against a Zipfian distribution as well. Figure 4.11 illustrates the performance of the algorithm against different values of the the exponent characterizing the distribution. As the value of the exponent increases, the performance of the all algorithms remains relatively unaffected. In all cases the XRJN and XRJN* remain almost an order of magnitude more efficient than HRJN* both in respect of processing time and accessed tuples.

Real datasets. The results using the real datasets are in accordance with the results of the synthetic ones. As the number of additional relations increases, the performance gain in terms of accessed tuples increases as well. Figures 4.12(b) and 4.13(b) show that processing time and the number of tuples accessed by HRJN* increases much faster than in the case of XRJN*. XRJN* is up to 10 times more efficient for both HOUSE and NBA datasets. We should note at

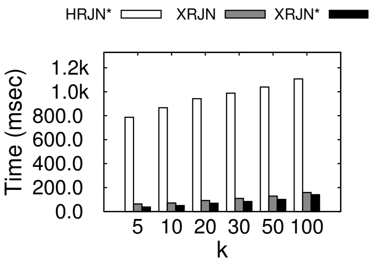


(a) Time

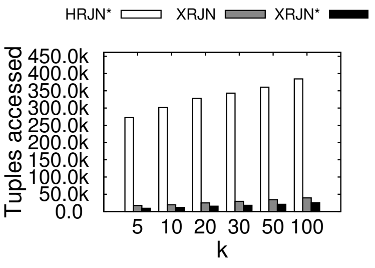


(b) Accessed tuples

Figure 4.14: NBA: varying k



(a) Time



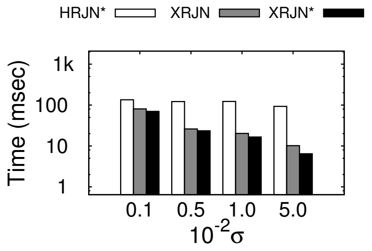
(b) Accessed tuples

Figure 4.15: HOUSE: varying k

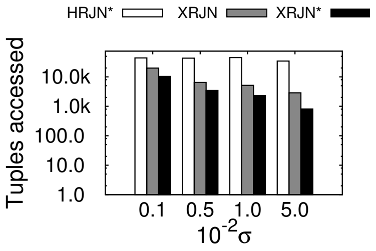
this point that the default value of the join selectivity parameter for the NBA dataset was set to $\sigma = 0.01$ due to the small size of the dataset.

Figures 4.14 and 4.15 depict the behavior of the algorithms when varying parameter k. Both XRJN and XRJN* are more efficient than HRJN* regarding the access depth. It is noteworthy that XRJN* is not only more efficient than HRJN* but also the performance of XRJN* is minimally affected by the increase of parameter k. On the contrary Figures 4.14(b) and 4.15(b) indicate that the cost of HRJN* increases linearly with respect to k.

Figures 4.16 and 4.17 illustrate the performance of the algorithms when varying the join selectivity σ. All algorithms behave as expected based on the evaluation of the uniform distribution. It is worth noting that the performance of XRJN* improves much faster than HRJN* both in terms of processing time and in terms of access depth. Similarly to the case of the uniform distribution sets the XRJN* forces the upper and lower bound to converge faster than HRJN*

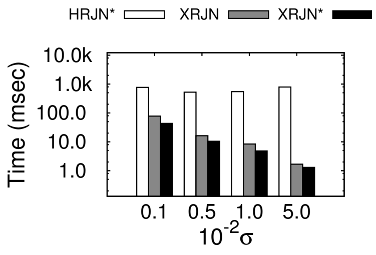


(a) Time

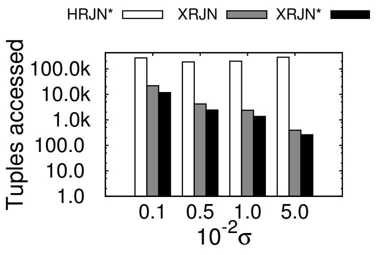


(b) Accessed tuples

Figure 4.16: NBA: varying σ (log scale)



(a) Time



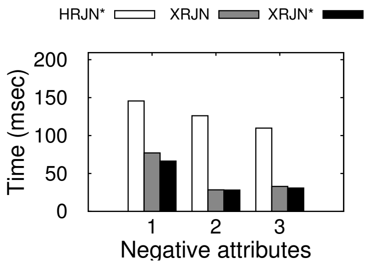
(b) Accessed tuples

Figure 4.17: HOUSE: varying σ (log scale)

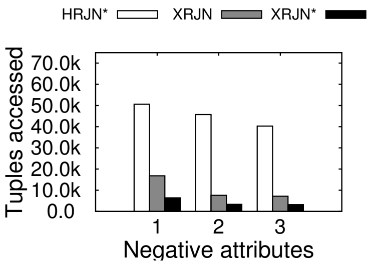
as it aims towards creating total combinations fast, fact that helps the lower and upper to converge.

Figures 4.18 and 4.19 illustrate the performance of the algorithms when varying the number of negative attributes. The performance of the algorithms varies due to the random selection of the negative attributes in each run of the experiments. Nevertheless, XRJN and XRJN* perform in all cases significantly better than HRJN*. Especially in the case of the HOUSE dataset both XRJN and XRJN* are nearly an order of magnitude more efficient than HRJN*.

Alternative combinations generation. We tested both algorithms using the default experimental setup. Figure 4.20 illustrates the performance of the two algorithms when a set of m combinations is presented for each main product. As expected the processing cost rises as the number of combinations per main object increases. Interestingly, while the access depth increases for XRJN*, it

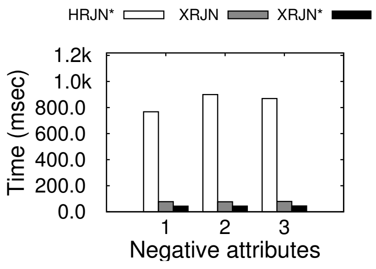


(a) Time

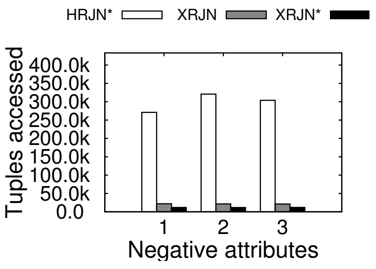


(b) Accessed tuples

Figure 4.18: NBA: varying # of negative attributes



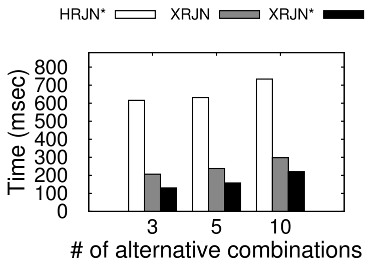
(a) Time



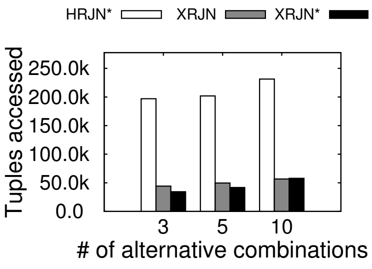
(b) Accessed tuples

Figure 4.19: HOUSE: varying # of negative attributes

remains stable for HRJN*. As HRJN* accesses 4 times more tuples than XRJN* to discover the best combination for each main object, it has already accessed enough tuples to form the alternative combinations before the lower bound exceeds the upper bound. Therefore, the only extra cost for HRJN* is that of calculating the top-m alternative combinations. In contrast, XRJN* utilizes a more efficient bounding scheme which allows the algorithm to identify the best combination for each main object after accessing a much smaller number of tuples than HRJN*. Consequently, XRJN* needs to continue reading from the relations after the lower bound has exceeded the upper bound in order to form the top-m combinations. As a result, the access-depth for XRJN* rises as the number of alternative combinations increases. In all cases however, it remains up to 4 times more efficient in terms of access-depth and up to 3 times more efficient in terms of processing time than HRJN*.



(a) Time



(b) Accessed tuples

Figure 4.20: Varying the number of alternative combinations

4.8 Conclusion

In this chapter, we address the problem of discovering the top-k combinations between a single main product relation and several additional relations that can be joined with the former one. Our approach tries to balance between finding the best combinations and giving the user the ability to explore the products of the database and the possible combinations between them, without the need to specify which relations will be joined to the main relation. To this end, we define the Exploratory Top-k Join query and we present a pull-bound framework for query processing. We propose a bounding scheme that exploits the properties of the formed combinations in order to efficiently calculate the result. The resulting algorithm has strong theoretical guarantees, namely it is instance-optimal. We also propose a more effective pulling strategy than plain round-robin, which further boosts the performance of query processing. In our experimental evaluation, we show that our algorithms perform consistently better than an adaptation of a state-of-the-art rank-join algorithm.

Part III

Exploratory Analysis of User Queries

In this part, we present exploratory analysis algorithms that aim to identify objects that are interesting and potentially attractive to a wide range of users. In Chapter 5, we present the notion of the *most continuous influential object* and we describe algorithms for processing the respective query, while in Chapter 6, we present algorithms for identifying objects that are attractive to diverse groups of users. In Chapter 7, we propose algorithms for improvement of the influence score of spatio-textual objects, through an enhancement of their textual description.

Chapter 5

Discovering Influential Data Objects over Time

Users typically access online databases through top-k queries, which return to them a ranked list of the k best products according to their preferences. Naturally, it is important for product manufacturers to identify which of their products are consistently ranked highly in user queries and therefore are visible to a large number of users over long periods of time. In this chapter, we study the problem of identifying database objects that have constantly a high influence score for long time periods. To take into account the temporal dimension, we define the *continuous influential query*, which retrieves the product that remains influential for the longest temporal range within a time horizon. We present algorithms for efficient processing and retrieval of continuous influential data objects that support incremental retrieval of the next continuous influential data object. The performance of the presented algorithms is evaluated through a detailed experimental study.

5.1 Introduction

As already described, top-k queries help users navigate and explore online databases by presenting them a small set of products which best fit their preferences. From the perspective of the product manufacturers, top-k queries are of great interest as well, since the visibility of a product clearly depends on the number of different top-k queries for which it belongs to the respective top-k

result sets. The reason for this is twofold: 1) users usually consider only a few of the presented products and ignore the remaining ones, and 2) products that appear in the top-k result sets are far more likely to be chosen by a potential customer, because those products are better suited to the customers' preferences. Intuitively, a product that appears in as many as possible top-k query result sets, has a higher visibility and therefore also a higher impact on the market. This relationship between products and queries is captured by reverse top-k queries which return the set of user preferences for which a given product is in the result set of the respective top-k queries. Naturally, reverse top-k queries lead to defining the *most influential products* based on the cardinality of their reverse top-k result sets [112]. Identifying the most influential products from a given set of products is important for market analysis, since the product manufacturers can estimate the impact of their products in the market.

An important aspect that affects the impact of a product that has not been taken into account yet, is the fact that user preferences change over time. The customers' criteria can differ significantly over time for various reasons. For example, in online marketplaces, new customers pose queries and new preferences are collected. In addition, users who have already posed queries will disconnect after some time. As user preferences change over time, a product that appears consistently in the top-k results of as many customers as possible, thus satisfying many customers' criteria at any time, has a higher impact on the market than a product that is absent from those results.

In this chapter, we study the problem of finding the product that belongs consistently to the most influential products over time, the *continuous influential product*. This is an important problem for many real-life applications. For example, products that are constantly interesting to a large number of customers have a potentially large impact on the market, and advertising them in the first page of an online marketplace can attract more customers and motivate them to explore the product database and potentially increasing this way the visibility of the products in the database. Continuous influential objects can be exploited also in promoting alternative products or services as well. For instance, routes or destinations in a city which are continuously preferred by drivers and therefore congested could be identified. A service offering navigation through GPS could identify parts which are constantly congested and take this information into consideration when proposing a route to a driver.

In the following, we first define formally the problem of continuous influential objects and provide a baseline algorithm that sequentially scans all time intervals in order to retrieve the most continuous influential object. Then, we provide a bounding scheme in order to facilitate early termination of our algorithms and avoid processing time intervals that do not alter the result set. Summarizing, the main contributions are:

- We study, for the first time, the problem of identifying the data object that has the highest impact over time.
- An appropriate score of influence (called *continuity score*) based on the reverse top-k query is defined to capture the product impact over a period of time.
- We derive upper and lower bounds for the continuity score of a given object that lead to efficient algorithms for retrieving the most continuous influential object. Two different algorithms are presented that provide early termination based on the bounds, but follow different strategies in order to terminate as soon as possible.
- We conduct a detailed experimental study for various setups and demonstrate the efficiency of our algorithms.

The rest of this chapter is organized as follows: Section 5.2 provides an overview of related work. In Section 5.3, we provide the necessary preliminaries, while in Section 5.4, we formulate the problem statement. Section 5.5 presents a baseline algorithm for finding the data object that belongs consistently to the most influential products. Section 5.6 provides the foundation for our bounding scheme and describe the two threshold-based algorithms. Our experimental results are presented in Section 5.7. Finally, in Section 5.8, we present the conclusions of our study.

5.2 Related work

The problem of discovering influential objects over time is closely related to top-k, reverse top-k, and influential top-m queries.

Top-k queries. Recent approaches to efficient processing of top-k queries [16, 25, 40, 54] use materialized views of already processed queries in order to efficiently compute new queries by reducing the amount of data points examined to produce the result. Methods using precomputed results perform better in cases of static data. Efficient maintenance of materialized queries is discussed in [123, 124]. Other approaches use threshold-based algorithms [18, 34, 47, 79] and they exploit the use of multiple-sources. For a thorough overview we refer to the excellent survey by Ilyas et al. [58].

Reverse top-k queries. Vlachou et al. [110] were the first to address the problems of monochromatic and bichromatic reverse top-k queries. The monochromatic reverse top-k query of an object q returns a locus in the space of the weighting vectors, since only the data objects are given. Any weighting vector inside that locus defines a top-k query and the object q belongs in the result set of this query. In the bichromatic, both the objects and the weighting vectors are given, and the result is the set of weights for which the query point is among the top-k highest ranked objects. Bernecker et al. [10] study reverse queries through a unified approach. The authors examine the Inverse ϵ-Range, Inverse $k-$NN and Inverse Dynamic Skyline queries using a three-filter approach. The first two filters use only the query points whose number is usually small and the third query accessed the dataset points in ascending order of maximum distance from the query points.

Influential objects were introduced by Vlachou et al. [112]. As influential are considered the objects that appear in the top-k sets of many weight vectors. Arvanitis et al. [7] try to discover attractive products to users using the principle of skyline sets [13]. Several approaches [81, 118, 119] try to find the area where a product should be more visible.

Temporal queries. Jestes et al. [59] study the problem of performing top-k queries on a time window. They assume that the values of the objects change over time and instead of performing instant top-k queries, they retrieve the top-k objects by ranking them after aggregating their scores in a query interval. Lee *et al.* [68] discuss the idea of objects that appear continuously in top-k queries over data streams. They focus on discovering objects that appear continuously on a moving window of time. In [106] the authors study techniques for *durable top-k search* in document archives, where the aim is to identify documents that are consistently in the top-k results of a given query. Kontaki et al. [66] study the problem of discovering the objects that remain the most dominant over a data stream. Our main difference towards these approaches is that they consider the ranking functions to be static while the values of the objects are changing while we consider the exact opposite.

Other work related to top-k and time includes processing of top-k queries on temporal data where the aim is finding the top-k objects at a particular time [70], monitoring top-k queries over sliding windows [82], and efficient processing of a large number of continuous top-k queries by exploiting clusteredness in user preferences [125]. Moreover, in [111] the authors define the distance-based re-

Symbol	Description
$\mathcal{T}$	Time domain of V intervals $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$
$\mathcal{D}$	d-dimensional dataspace
S	Set of data objects
$t_s(w), t_e(w)$	Start and end of validity of w
f_w	Preference function associated with w
k	Value of top-k
m	Number of most influential objects retrieved
$cis(o)$	Continuity score of object o
$L(o), U(o)$	Lower and upper bound: $L(o) \leq cis(o) \leq U(o)$

Table 5.1: Overview of symbols

verse top-k query and study how to monitor efficiently the distance-based reverse top-k result set over a set of mobile devices.

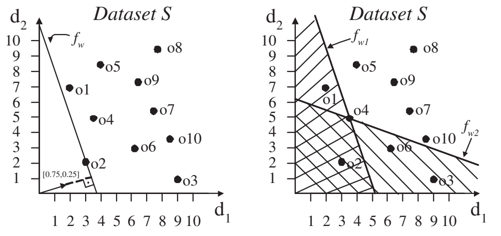
5.3 Preliminaries

Let $\mathcal{D}$ be a dataspace with d dimensions and S be a set of data objects on D. A data object is represented as a point $o = \{o_1, \ldots, o_d\}$ where o_i is the value of the i^{th} attribute. We refer to Table 5.1 for an overview of the main symbols.

5.3.1 Time-invariant case

We remind at this point that given a monotonic scoring function $f : S \rightarrow \mathbb{R}$, a top-$k$ query returns the k best objects $o \in S$ ranked according to their scores. We denote the result set of top-k query defined by a weighting vector w as $TOP_k(w)$. One of the most commonly used scoring functions is the linear function, where for a given data object o and a weighting vector w, its score $f_w(o)$ is equal to the weighted sum of the individual values of o: $f_w(o) = \sum_{i=1}^{d} w[i]o[i]$, where $w[i] \geq 0$ $(1 \leq i \leq d)$.

Geometrically, in the Euclidean space an object $o \in S$ can be represented as a point and a linear top-k query can be represented by a vector w. Consider the dataset S depicted in Figure 5.1(a), and the query $w = (0.75, 0.25)$. In the d-dimensional space, the hyperplane (line in 2d) which is perpendicular to vector w and contains a point o defines the score of point o and all points lying on the



(a) Top-k query (b) Reverse top-k query

Figure 5.1: Top-k and reverse top-k query examples

same perpendicular hyperplane have the same score based on w. The rank of a point o based on a weighting vector w is equal to the number of the points enclosed in the half-space defined by the perpendicular hyperplane that contains the origin of the data space. In Figure 5.1(a), o_2 is the top-1 object for this query, since no other data object is enclosed in the corresponding half-space.

Given a dataset S of objects, a set W of weighting vectors, an object q and an integer k, a reverse top-k query returns all weighting vectors $w \in W$ for which $q \in TOP_k(w)$. We denote the result set of weighting vectors as $RTOP_k(q)$. As mentioned in Chapter 2 the cardinality of the $RTOP_k(q)$ set of an object q is the *influence score* of q and we denote it as $f_k^I(q)$. A top-m influential query returns the object with the highest influence score, i.e., the m objects that appear in the top-k results of most user queries.

In Figure 5.1(b), a dataset S is depicted together with two different weighting vectors w_1 and w_2. Assume that a reverse top-3 query is posed, while the query object is o_4. Then, w_1 belongs to the reverse top-3 query result set, since only 2 objects are contained in the underlined half-space by w_1. However, w_2 does not belong to the reverse top-3 query result set, since there exist 3 objects in the underlined half-space by w_2.

5.3.2 Temporal model

We model the time domain $\mathcal{T}$ as an ordered set of V disjoint time intervals that cover the complete domain, i.e., $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_V\}$ and $\mathcal{T}_i \bigcap \mathcal{T}_j = \emptyset$ for $i \neq j$. We denote the start and end of time interval $\mathcal{T}_i$ with $t_s(\mathcal{T}_i)$ and $t_e(\mathcal{T}_i)$ respectively. Then, it also holds that $t_e(\mathcal{T}_i) = t_s(\mathcal{T}_{i+1})$, and that $t_s(\mathcal{T}_1)$ and $t_e(\mathcal{T}_V)$ denote the start and end of $\mathcal{T}$ respectively. Obviously, the number of time intervals V is user-specified and application-dependent, and its exact value depends on the desired level of detail for monitoring temporal changes.

In order to model the interval that a user is online, we associate the weighting vector representing the user preferences with a time interval. Thus, given a weighting vector w, we denote the start of this interval as $t_s(w)$ and its end as $t_e(w)$. We are now ready to define the validity of a weighting vector with respect to a time domain $\mathcal{T}$ that consists of time intervals.

Definition 5.1. ***Validity of weighting vector.*** *Given a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$ and a weighting vector w, the validity of w with respect to $\mathcal{T}$ is the interval $[t_s(\mathcal{T}_i), t_e(\mathcal{T}_j))$, where $t_s(w) \in \mathcal{T}_i$ and $t_e(w) \in \mathcal{T}_j$.*

Based on Definition 5.1, we consider as the validity period of a weighting vector w the interval defined by the start and end of the time intervals ($\mathcal{T}_i$ and $\mathcal{T}_j$) that enclose $t_s(w)$ and $t_e(w)$ respectively. Henceforth, we will use $t_s(w)$ to refer to $t_s(\mathcal{T}_i)$ and $t_e(w)$ to refer to $t_e(\mathcal{T}_j)$.

5.4 Problem definition

Given a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, we define a total order $\prec$ such that $\mathcal{T}_i \prec \mathcal{T}_j$ if $t_e(\mathcal{T}_i) \leq t_s(\mathcal{T}_j)$ for any $\mathcal{T}_i, \mathcal{T}_j \in \mathcal{T}$. Furthermore, we use $ITOP_k^m(\mathcal{T}_i)$ to refer to the result set of the top-m most influential objects by taking into account only the weighting vectors that are valid in the interval $\mathcal{T}_i$.

In order to identify products that are consistently highly ranked for multiple users as time passes, we define the *continuity score* of an object $o \in S$.

Definition 5.2. ***Continuity score.*** *Given a dataset S, a set of weighting vectors W, and a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, the* continuity score $cis(o)$ *of an object $o \in S$ is the maximum number of consequent intervals $\mathcal{T}_i$ for which o belongs to the top-m most influential data objects, i.e., $o \in ITOP_k^m(\mathcal{T}_i)$.*

The continuity score of an object is practically a measure of the object's aggregated influence over time. As we are interested in discovering the object with highest continuity score, we derive the definition of the most continuous influential data object in a straightforward way.

Definition 5.3. ***Most continuous influential data object.*** *Given a dataset S, a set of weighting vectors W, and a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, the* most continuous influential data object $o \in S$ *is the object for which it holds that $\nexists o' \in S$ such that $cis(o') > cis(o)$.*

We are now ready to define formally the problem of discovering the most influential object over time. Another closely related problem is the one of discovering a ranked set of the most influential objects over time.

Definition 5.4. ***Most continuous influential object.*** *Given a dataset S, a set of weighting vectors W, and a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, find the most continuous influential object $o \in S$.*

Definition 5.5. ***Top-N continuous influential objects.*** *Given a dataset S, a set of weighting vectors W, a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, and an integer N, find the ranked set of the N most continuous influential object $\{o1, o2, \ldots, oN\} \in S$.*

In this chapter, we focus our attention to Problem 5.4 and present our algorithms for solving this problem. However, our algorithms can be extended in a straightforward way to solve also Problem 5.5.

5.5 Sequential interval scan

A baseline algorithm for solving Problem 5.4 is to compute the $ITOP_k^m(\mathcal{T}_i)$ sets for all time intervals $\mathcal{T}_i$ of $\mathcal{T}$ and simply follow a counting approach of the appearance of any data object o in consequent intervals. Then, the most continuous influential object is the one that appears in the $ITOP_k^m(\mathcal{T}_i)$ sets for the maximum number of consequent intervals. In the following, we refer to this algorithm as *Sequential Interval Scan* (*SIS*).

Intuitively, in each iteration, *SIS* examines the next consequent interval $\mathcal{T}_i \in \mathcal{T}$ and computes the set of most influential objects $ITOP_k^m(\mathcal{T}_i)$ within $\mathcal{T}_i$. For each retrieved object $o \in ITOP_k^m(\mathcal{T}_i)$, we maintain its current continuity score, which is derived based on the processed intervals so far. We use the concept of an *alive object* to refer to any object retrieved in a previous interval $\mathcal{T}_j (j \leq i)$ that is influential in all intervals between $\mathcal{T}_j$ and $\mathcal{T}_i$ and also belongs to the most recently processed $ITOP_k^m(\mathcal{T}_i)$ set; we also refer to objects that stopped being influential at some intermediate interval between $\mathcal{T}_1$ and $\mathcal{T}_i$ as *dead objects.* To ensure correctness, *SIS* needs to maintain the alive objects and only a single dead object, which is the one with the highest continuity score among all other dead objects. After having examined all intervals, the most continuous influential object is either the alive object with the highest score among alive objects or the dead object.

Algorithm 9 presents the pseudocode of *SIS.* In more detail, in each iteration (lines 2–15) the $ITOP_k^m(\mathcal{T}_i)$ set for the next time interval $\mathcal{T}_i$ is computed. Dead

Algorithm 9 *Sequential Interval Scan (SIS)*

```
Input: S: set of data objects,
       $\mathcal{T} = \{\mathcal{T}_1, \dots \mathcal{T}_V\}$
       $k, m$: the parameters of the $ITOP_k^m$ queries
Output: $o$: the most continuous influential object
 1: A← $\emptyset$; d←null; //A: alive objects, d: dead object
 2: for $i = 1 \dots V$ do
 3:    $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_i)$;
 4:    for all $o \in$ A and $o \notin \mathcal{I}$ do
 5:       A← A − {o} //remove dead objects
 6:       d← objMaxScore({d} $\bigcup$ {$o$}) //select the object with the highest score
 7:    end for
 8:    for all $o \in \mathcal{I}$ do
 9:       if $o \in$ A then
10:          $o$.incScore() //increase score
11:       else
12:          A← A $\bigcup$ {o} //add new objects
13:       end if
14:    end for
15: end for
16: $o \leftarrow$ objMaxScore(A$\bigcup${d})
17: return $o$
```

objects are identified and removed from the list A of alive objects, and also the dead object d with the highest score is found using function *objMaxScore()* (lines 4–7). Then, the retrieved influential objects in $\mathcal{T}_i$ are examined, and if an object belongs to A (i.e., was and remains alive) then its score is increased by 1(function *incScore()*, line 10). Also, for any retrieved object that was not alive in the previous iteration we add it to the list of alive objects A (line 12). When all time intervals of $\mathcal{T}$ have been examined, the algorithm terminates and reports as most continuous influential object the object with maximum score among the alive objects and the dead object (line 16).

The main shortcoming of *SIS* is that it needs to evaluate the $ITOP_k^m$ query for all $|V|$ time intervals. In the following, we study how to derive appropriate score bounds, in order to find the most continuous influential object without processing all $ITOP_k^m$ queries.

5.6 Algorithms with early termination

SIS relies on processing multiple consequent intervals of $\mathcal{T}$ to produce the most continuous influential object. In fact, all our algorithms rely on the evaluation of multiple $ITOP_k^m$ queries in different intervals $\mathcal{T}_i$, in order to find the most continuous influential object, however these intervals are not necessarily consequent. In this sense, our algorithms treat the $ITOP_k^m$ computation as black-box, hence any existing techniques that solve efficiently the problem of identifying influential objects can be directly exploited by our algorithms.

Let us assume that at some point during query processing, a subset of (not necessarily consequent) intervals of $\mathcal{T}$ has been processed. We define the following sets for any retrieved data object o.

Definition 5.6. *Given a data object o, a set of processed intervals $\{\mathcal{T}_i\}$ and a set of corresponding result sets $\{ITOP_k^m(\mathcal{T}_i)\}$, we define:*

- *$\mathcal{T}^+(o)$ is the set of intervals $\{\mathcal{T}_i\}$, such that $\mathcal{T}_i \in \mathcal{T}^+(o)$ if $o \in ITOP_k^m(\mathcal{T}_i)$*
- *$\mathcal{T}^-(o)$ is the set of intervals $\{\mathcal{T}_i\}$, such that $\mathcal{T}_i \in \mathcal{T}^-(o)$ if $o \notin ITOP_k^m(\mathcal{T}_i)$*
- *$\mathcal{LB}(o)$ is a maximal sequence of intervals $\{\mathcal{T}_i, \mathcal{T}_{i+1}, ..., \mathcal{T}_j\}$, such that $\forall \mathcal{T}_z \in \mathcal{LB}(o) : \mathcal{T}_z \in \mathcal{T}^+(o)$*
- *$\mathcal{UB}(o)$ is a maximal sequence of intervals $\{\mathcal{T}_i, \mathcal{T}_{i+1}, ..., \mathcal{T}_j\}$, such that $\forall \mathcal{T}_z \in \mathcal{UB}(o) : \mathcal{T}_z \in \mathcal{T} - \mathcal{T}^-(o)$*

We emphasize that according to Definition 5.6, $\mathcal{T}^+(o)$ and $\mathcal{T}^-(o)$ are sets of intervals, i.e., they may contain non-consequent intervals. Instead, the sequences $\mathcal{LB}(o)$ and $\mathcal{UB}(o)$ contain consequent intervals, and moreover they are of maximal size, i.e., and there exists no other longer sequence of intervals with the same properties respectively.

By exploiting the above sets and sequences, we derive an upper and a lower bound on the score of any candidate most continuous influential object.

Lemma 5.1. (Score bounds): *The continuity score of object o is bounded by the lower bound $L(o)$ and the upper bound $U(o)$, i.e., $L(o) \leq cis(o) \leq U(o)$, where $L(o) = |\mathcal{LB}(o)|$ and $U(o) = |\mathcal{UB}(o)|$ are the lengths of the sequences $\mathcal{LB}(o)$ and $\mathcal{UB}(o)$ respectively.*

Proof. By contradiction. Let us assume that $cis(o) < L(o)$. Then it holds that there exists a sequence of processed intervals of length $|\mathcal{LB}(o)|$ such that for each time interval $\mathcal{T}_i$ of $\mathcal{LB}(o)$ it holds that $\mathcal{T}_i \in \mathcal{T}$ and $o \in ITOP_k^m(\mathcal{T}_i)$, which leads to a contradiction since $cis(o)$ is defined by the sequence of maximum length (according to Definition 5.2). Similarly, the assumption $cis(o) > U(o)$ leads to a contradiction, because for each time interval $\mathcal{T}_i$ of the sequence that defines $cis(o)$, it holds that $\mathcal{T}_i \notin \mathcal{T}^-(o)$ for any set of processed intervals $\{\mathcal{T}_i\}$. In other words, the sequence of intervals whose length defines $cis(o)$ is always smaller or equal to the sequence $\mathcal{UB}(o)$ whose length defines $U(o)$, hence $cis(o) \leq U(o)$ which is a contradiction. ■

The lower bound $L(o)$ of o is equal to the continuity score of the object o, if we take into account only the time intervals that have been processed so far. The upper bound $U(o)$ of o is the continuity score of the object o, if we assume that for any time interval $\mathcal{T}_i$ that does not belong to $\mathcal{T}^-$ the object o belongs to $ITOP_k^m(\mathcal{T}_i)$ (because optimistically for all unprocessed time intervals, o may belong to the most influential objects).

Theorem 5.1. (Early termination condition): *The data object o is the most continuous influential object, if for any other data object o' it holds that $L(o) \geq U(o')$.*

Proof. By contradiction. Let us assume that o is not the most continuous influential object, even though it holds that $L(o) \geq U(o')$. Thus, there must exist another object o' which is the most continuous influential object (i.e., $cis(o) < cis(o')$). Then, it holds that $L(o) \leq cis(o) \leq U(o)$ and $L(o') \leq cis(o') \leq U(o')$. From these inequalities, we derive that $L(o) \leq cis(o) < cis(o') \leq U(o')$ and finally that $L(o) < U(o')$, which is a contradiction. ■

The intuition of the above condition for early termination is that if an object has a continuity score based on some processed time intervals that is definitely higher than the score of any other object, then it can be safely reported as the most continuous influential object, because the score of any other object cannot increase sufficiently in the remaining time intervals.

Algorithm *SIS* is oblivious of the derived bounds and examines all time intervals following a brute-force approach. Hence, we propose two algorithms, termed *Early Termination Interval Scan* (*TIS*) and *Early Termination Best-First Interval* (*TBI*) that exploit the bounds to provide early termination. However, despite using the same concept of bounding, *TIS* and *TBI* follow different strategies in order to terminate as soon as possible. *TIS* aims to maximize as quickly as possible the lower bound of the current most continuous influential object o and therefore examines time intervals sequentially. Instead, *TBI* aims to reduce the upper bound of any object o by breaking the longest unprocessed sequence of time intervals.

Algorithm 10 *Early Termination Interval Scan (TIS)*

```
Input: $S$: set of data objects
       $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_V\}$
       $k, m$: the parameters of the $ITOP_k^m$ queries
Output: $o$: the most continuous influential object
1: $i \leftarrow 1$, upperBound $\leftarrow 0$, lowerBound $\leftarrow -1$, $A \leftarrow \emptyset$
2: while lowerBound<upperBound do
3:   $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_i)$
4:   $i = i + 1$
5:   $o \leftarrow$objMaxScore(A$\bigcup$\{d\})//select the object with the highest score
6:   lowerBound $\leftarrow o$.score()
7:   for all $o \in$ A and $o \notin \mathcal{I}$ do
8:     A$\leftarrow$ A $-$ \{o\} //remove dead objects
9:     d$\leftarrow$ objMaxScore(\{d\} $\bigcup$\{$o$\})
10:  end for
11:  for all $o \in \mathcal{I}$ do
12:    if $o \in$ A then
13:      $o$.incScore() //increase score
14:    else
15:      A$\leftarrow$ A $\bigcup$\{o\} //add new objects
16:    end if
17:  end for
18:  $o' \leftarrow$objMaxScore(A$-$\{$o$\})
19:  upperBound= $max(o'$.score()+$(V - i)$, $d$.score())
20: end while
21: return $o$
```

5.6.1 Early termination interval scan

In this section, we describe the Early Termination Interval Scan (*TIS*) algorithm. Similar to the *SIS* algorithm, *TIS* processes sequentially the time intervals of time domain $\mathcal{T}$. However, the significant advantage of *TIS* lies in the fact that it can terminate early and report the most continuous influential object o without processing the $ITOP_k^m$ query for all V time intervals $\mathcal{T}_i$.

Intuitively, the main objective of *TIS* is to increase the lower bound of any retrieved object, by scanning the time intervals sequentially. Notice that only consequent time intervals may lead to a higher lower bound. *TIS* takes advantage of the fact that it processes the time intervals sequentially and computes the $L(o)$ and $U(o)$ without maintaining the sets $\mathcal{T}^-(o)$ and $\mathcal{T}^+(o)$. The lower bound is defined as the continuity score of the current most continuous influential object, which can be computed by maintaining only the alive and dead objects similar to *SIS*. For *TIS*, the upper bound is defined as the maximum value of the score of the dead object, or the second highest score of the alive objects plus the number of remaining time intervals.

Although these bound definitions of *TIS* are simpler than the ones of lower and upper bound in Lemma 5.1, it can be easily shown that they are equivalent. The reason for their simplicity is that *TIS* examines intervals sequentially, which is a special case of interval selection and the computation of the bounds can be simplified. Instead, the bound definitions of Lemma 5.1 and Definition 5.6 apply in the general case of selecting any interval for processing next (not necessarily in a sequential manner).

Algorithm 10 contains the pseudocode of *TIS*. In each iteration, the next interval of the time domain $\mathcal{T}$ is examined, and the result set $ITOP_k^m(\mathcal{T}_i)$ is computed. For each retrieved object, a score is maintained, which is the maximum number of consequent intervals for which this object belongs to the respective $ITOP_k^m$ sets. The retrieved data objects that belong to the most recent $ITOP_k^m$ set are considered to be alive, while we also keep track of the dead object with the highest score.

In more detail, as long as the termination condition does not hold (lines 2-20), the $ITOP_k^m$ set for the next time interval is computed and the alive and dead objects are updated (lines 7-10, 13, 15), similarly to the case of the *SIS* algorithm. Furthermore, in each iteration, the current most continuous influential object o is found (line 5). The current score of o defines the lower bound (line 6), as any other point must have a higher score to become the most continuous influential. Also, the alive object o' with the second highest score is found (line 18)[1]. The maximum possible score of any object (regardless of whether it has been retrieved or not) is equal to maximum value between the score of the dead object and the score of o' plus the number of remaining unprocessed intervals. This is because any object that is still alive may be (in the best case scenario) in the $ITOP_k^m$ set for all remaining time intervals. Also, the score of the dead object cannot be increased further. Notice that if the same object appears in the $ITOP_k^m$ set, it is considered to be a new alive object. Any new alive object can appear only in the $V - i$ remaining time intervals. Thus, if the termination condition holds, no object can exceed the score of the currently

[1]In the extreme case where $\mathrm{A}-\{o\} = \emptyset$ we assume that $o'.score = 0$.

most continuous influential object and the algorithm safely reports this object as the result.

It should be noted that *TIS* reports the most continuous influential object over a time domain, however it does not report its score accurately. One can draw parallels with Fagin's NRA algorithm [34], which produces the top-k objects from ranked lists but without guaranteeing accuracy of scores. In order to calculate the exact continuity score of the most continuous influential object, we need to proceed until we find an interval where the object does not belong to the $ITOP_k^m$ set.

5.6.2 Early termination best-first interval

In the following, we describe the Early Termination Best-first Interval (*TBI*) algorithm. The most important difference to *TIS* is that *TBI* follows a different strategy with respect to interval selection, namely *TBI* does not process intervals sequentially.

For each retrieved object o, *TBI* maintains the two sets $\mathcal{T}^+(o)$ and $\mathcal{T}^-(o)$ that correspond to the processed time intervals for which o belongs to or not to the most influential data objects respectively. This information is sufficient to derive the lower bound $L(o)$ and upper bound $U(o)$ of o. The algorithm first computes the influential objects $ITOP_k^m(\mathcal{T}_1)$ and $ITOP_k^m(\mathcal{T}_V)$. The following example demonstrates the information maintained by *TBI* at this point.

Example 5.1. *Let us assume that* $V = 6$, $m = 2$, *and that* $ITOP_k^m(\mathcal{T}_1) = \{o_1, o_2\}$ *and* $ITOP_k^m(\mathcal{T}_6) = \{o_2, o_3\}$. *Then,* TBI *maintains the following sets:* $\mathcal{T}^+(o_1) = \{\mathcal{T}_1\}$, $\mathcal{T}^-(o_1) = \{\mathcal{T}_6\}$, $\mathcal{T}^+(o_2) = \{\mathcal{T}_1, \mathcal{T}_6\}$, $\mathcal{T}^-(o_2) = \emptyset$, $\mathcal{T}^+(o_3) = \{\mathcal{T}_6\}$, $\mathcal{T}^-(o_3) = \{\mathcal{T}_1\}$. *In addition, the derived bounds are:* $L(o_1) = 1$, $U(o_1) = 5$, $L(o_2) = 1$, $U(o_2) = 6$, $L(o_3) = 1$, $U(o_3) = 5$.

TBI iteratively selects a time interval that has not yet been processed and computes the influential objects in the selected time interval. Then, the bounds of retrieved objects can be updated as indicated in the following.

Example 5.2. *Continuing the previous example, assume that the next interval that is processed is* $\mathcal{T}_3$ *and* $ITOP_k^m(\mathcal{T}_3) = \{o_2, o_4\}$. *Then, the following sets are maintained:* $\mathcal{T}^+(o_1) = \{\mathcal{T}_1\}$, $\mathcal{T}^-(o_1) = \{\mathcal{T}_3, \mathcal{T}_6\}$, $\mathcal{T}^+(o_2) = \{\mathcal{T}_1, \mathcal{T}_3, \mathcal{T}_6\}$, $\mathcal{T}^-(o_2) = \emptyset$, $\mathcal{T}^+(o_3) = \{\mathcal{T}_6\}$, $\mathcal{T}^-(o_3) = \{\mathcal{T}_1, \mathcal{T}_3\}$, $\mathcal{T}^+(o_4) = \{\mathcal{T}_3\}$, $\mathcal{T}^-(o_4) = \{\mathcal{T}_1, \mathcal{T}_6\}$. *In addition, the bounds are updated as follows:* $L(o_1) = 1$, $U(o_1) = 2$, $L(o_2) = 1$, $U(o_2) = 6$, $L(o_3) = 1$, $U(o_3) = 3$, $L(o_4) = 1$, $U(o_4) = 4$.

The remaining challenge is how to select the most beneficial time interval for the next influential query to be processed, i.e., the time interval that will lead the algorithm to terminate as quickly as possible. *TBI* follows a best-first approach by selecting the time interval that will split the longest $\mathcal{UB}(o)$ sequence for any o in the queue. Intuitively, this "breaks" long sequences of unknown time intervals, in an attempt to reduce the upper bound of any data object.

In more detail, the next interval to be processed is selected in the following way. Given a candidate data object o and the corresponding $\mathcal{UB}(o) = \{\mathcal{T}_i, ..., \mathcal{T}_j\}$, the middle time interval $\mathcal{T}_z$ is computed such that $z = i + \lceil \frac{j-i}{2} \rceil$. If $\mathcal{T}_z \notin \mathcal{T}^+(o)$ then $\mathcal{T}_z$ is the next interval. Otherwise, it means that $\mathcal{T}_z$ has been already processed and in this case the sequence $\{\mathcal{T}_i, ..., \mathcal{T}_z\}$ is tried to be split by finding the middle interval $\mathcal{T}_{z'}$ of it. If also $\mathcal{T}_{z'} \in \mathcal{T}^+(o)$, then the middle interval of $\{\mathcal{T}_z, ..., \mathcal{T}_j\}$ is examined if it qualifies for being the next interval. This is done recursively by examining always the longest sequence until an interval is found that does not belong to $\mathcal{T}^+(o)$. Note that it is guaranteed that such an interval exists, because otherwise $L(o) = U(o)$ and the algorithm terminates. Intuitively, computing $ITOP_k^m(\mathcal{T}_z)$ may break the longest sequence $\mathcal{UB}(o)$ in two smaller sequences if $o \notin ITOP_k^m(\mathcal{T}_z)$, thus reducing the upper bound, which will allow the algorithm to terminate faster.

During query processing, *TBI* keeps the retrieved data objects in a priority queue. The queue is sorted in descending order based on the upper bound $U(o)$ of each object o, so that immediate access to the object with the highest upper bound is provided. Algorithm 11 presents the pseudocode of *TBI.* First, the intervals $\mathcal{T}_1$ and $\mathcal{T}_V$ are processed and the retrieved objects are inserted in the queue (lines 1–4). The lower and upper bounds are initiated based on the object located at the head of the queue (lines 5, 6). In each iteration, we remove from the queue the object o (candidate object) with maximum upper bound $U(o)$ (line 8). Note that the candidate object is not necessarily the object with the highest continuity score based on the processed partitions (which is the lower bound), and there may exist another object o' that has a higher score (lower bound) currently. But it is guaranteed that the algorithm cannot terminate at this iteration even if o' was processed next, because it holds that $L(o') \leq U(o')$ and $U(o') \leq U(o)$ so that the termination condition cannot hold. Thus, *TBI* does not process unnecessary time intervals.

After selecting the candidate o with the highest upper bound, *TBI* recursively selects the middle interval to be processed (line 9) and processes the query (line 10). Afterwards, the queue is updated (line 11), which means that every object in $ITOP_k^m(\mathcal{T}_i)$ is either added to the queue (if it is the first time that it was retrieved) or the existing object is updated by changing the correspond-

Algorithm 11 *Early Termination Best-first Interval* (*TBI*)

```
Input: $S$: set of data objects
       $\mathcal{T} = \{\mathcal{T}_1, \dots, \mathcal{T}_V\}$
       $k, m$: the parameters of the $ITOP_k^m$ queries
Output: $o$: the most continuous influential object
 1: $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_1)$
 2: PQ.update($\mathcal{I}$) //Priority Queue based on upperBound
 3: $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_V)$
 4: PQ.update($\mathcal{I}$)
 5: upperBound = $U(queue.peek())$
 6: lowerBound = $L(queue.peek())$
 7: while lowerBound<upperBound do
 8:     $o \leftarrow queue.dequeue()$
 9:     $i$ = nextInterval($\mathcal{UB}(o)$) //find next interval
10:     $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_i)$
11:     PQ.update($\mathcal{I}$)
12:     upperBound = $U(queue.peek())$
13:     lowerBound = $L(o)$
14:     PQ.enqueue(o) //add o back to queue
15: end while
```

ing $\mathcal{T}^+$ set. Moreover, for every object in the queue that does not belong in $ITOP_k^m(\mathcal{T}_i)$, the set $\mathcal{T}^-$ is updated.

The algorithm terminates when it holds that the candidate object o has $L(o) \geq U(o'), \forall o' \in$ queue. This is the *termination condition* (line 7), which means that o has a higher lower bound than the upper bound of the current head object o' in the queue.

In principle, we can also free part of the memory during the processing of the algorithm, by evicting candidate points that will never become the most continuous influential object. The condition for eviction is if a candidate object o has $U(o) \leq L(o')$, where o' is another candidate object.

5.7 Experimental evaluation

In this section, we present the results of the experimental evaluation of all proposed algorithms. All algorithms were implemented in Java and the experiments

Parameter	**Values**
Dimensionality d	2, **3**, 4, 5
Cardinality of S	10K, **50K**, 100K
Cardinality of W	100K, **300K**, 500K
Data distribution of S	RL, **UN**, CO, AC
Data distribution of W	**UN**, CL
m	5, **10**, 15
k	5, **10**, 15
V	50, **100**, 150
σ_W	0.01, 0.5, **0.1**

Table 5.2: Experimental parameters and values

run on 2x Intel Xeon X5650 Processors (2.66GHz), 128GB. The index structure used was an R-tree with a buffer size of 100 blocks and the block size is 4KB.

Datasets. For the dataset S, we employ both real and synthetic data collections, namely uniform (UN), correlated (CO) and anticorrelated (AC) are used. For the uniform dataset, the data object values for all d dimensions are generated independently using a uniform distribution. The correlated and anticorrelated datasets are generated as described in [13]. To simulate the real-life scenario where no data object has optimal (i.e., minimum) values in all dimensions, we use the concept of a *cut value* C that prevents data from being generated in the area $[0, C]^d$. In addition, we use two real datasets, NBA and HOUSE, which were described in Section 4.7.

For the dataset W of the weighting vectors, two different data distributions are examined, namely uniform (UN) and clustered (CL). The clustered dataset W is generated as described in [112] and models the case where many users share similar preferences. In more detail, first C_W cluster centroids that belong to the (d-1)-dimensional hyperplane defined by $\sum w[i] = 1$ are selected randomly. Then, each coordinate is generated on the (d-1)-dimensional hyperplane by following a normal distribution on each axis with variance σ_W^2, and a mean equal to the corresponding coordinate of the centroid. To perform our experiments, we consider a set of $V = 100$ time intervals and assign a weighting vector $\mathbf{w}$ to a time interval $\mathcal{T}_i$ $(1 \leq i \leq 100)$ uniformly at random.

We conduct a thorough sensitivity analysis varying the dimensionality (2-5d), the cardinality (10K-100K) of the dataset S, the cardinality (100K-500K) of the dataset W the value of k (5-15), the value of m (5-15), and the number of intervals V (50-150). Unless explicitly mentioned, we use the default setup of: $|S| = 50K$, $|W| = 300K$, d=3, k=10, m=10, V=100, and uniform distribution for S and W. For the clustered dataset W we use $C_W = 5$ and $\sigma_W = 0.1$, and try different values of σ_W. The experimental parameters and the tested values are also shown in Table 5.2.

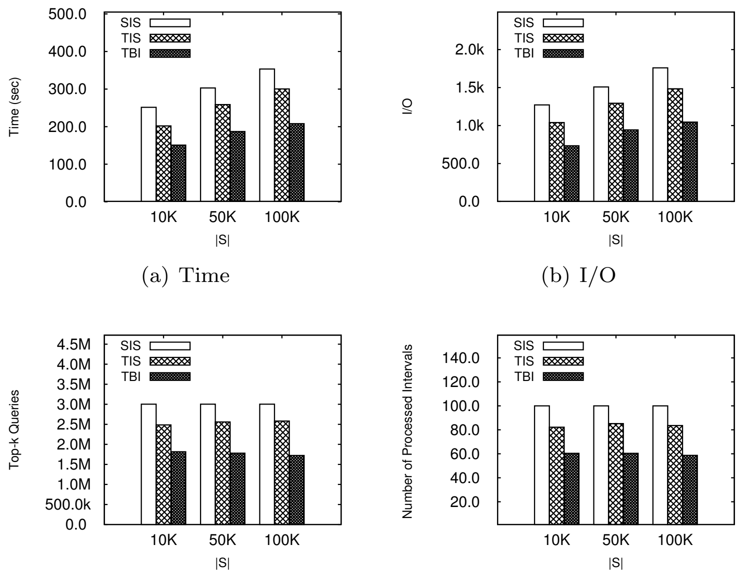
Algorithms. We evaluate the proposed algorithms, namely: a) sequential interval scan (*SIS*) algorithm, b) early termination interval scan (*TIS*) algorithm, and c) early termination best-first interval (*TBI*) algorithm. The algorithm employed for the underlying computation of most influential objects is the branch-and-bound algorithm proposed in [112].

Metrics. Our metrics include: a) the total execution time, b) the number of I/Os, c) the number of top-k evaluations, and d) the number of processed time intervals by each algorithm. Notice that we do not measure the I/Os that occur by reading W, since this is the same for every algorithm and does not affect their relative performance. For our experiments on synthetic data, we report the average of each metric over 10 different instances of the dataset. We generate the different instances by keeping the parameters fixed and changing the seeds of the random number generator. We adopt this approach in order to factor out the effects of randomization.

5.7.1 Performance of query processing

In the following we analyze the performance of the described algorithms with regard to the aforementioned parameters.

Effect of dataset size |S|. Figure 5.2 illustrates the performance of all algorithms when we vary the dataset cardinality. For all metrics, *TBI* outperforms both *TIS* and *SIS*. In terms of time (Figure 5.2(a)), *TBI* is significantly faster than the other algorithms, and more importantly, its gain increases as the dataset size increases. This is strong evidence that *TBI* scales gracefully with $|S|$. Similar observations can be made for the I/O metric depicted in Figure 5.2(b). In Figure 5.2(c), we depict the number of top-k queries that were processed by the $ITOP_k^m$ evaluations. Clearly, this metric is not significantly affected by increasing $|S|$, thus all algorithms present relatively stable values. Figure 5.2(d) depicts the number of processed intervals by each algorithm, which is a factor that affects all other metrics. *SIS* always processes the complete set



(a) Time

(b) I/O

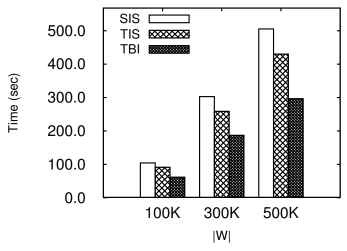
(c) Top-k queries

(d) Processed intervals

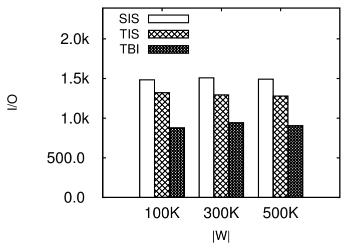
Figure 5.2: Effect of varying data cardinality $|S|$

of V intervals. *TIS* improves the performance of *SIS*, by exploiting the bounds and allowing for early termination. It should be clarified that *TIS* cannot process fewer than $V/2$ intervals to produce the correct result. Thus, in this setup ($V = 100$), *TIS* would in best case process 50 intervals. Still, *TBI* outperforms all other algorithms, which indicates that its best-first strategy for selecting the next interval performs more efficiently.

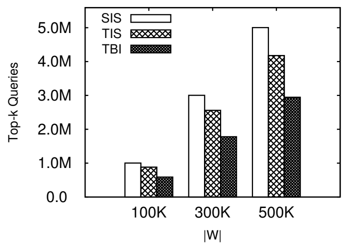
The advantage in the performance of *TIS* against *SIS* lies on the fact that *TIS* terminates when it is certain that the object with the highest continuity score cannot be surpassed. The advantage of *TBI* over *TIS* lies on the way the two algorithms calculate the lower and upper bounds of the continuity score of the objects. As mentioned before *TIS* and *TBI* calculate the lower and upper bounds of the score of each object. However, *TIS* tries to increase the lower bound until it is equal to the upper bound, while *TBI* follows the opposite strategy. In *TIS*, the lower bound of each alive object is increased each time by one interval. The upper bound of all dead objects decreases each time by one.



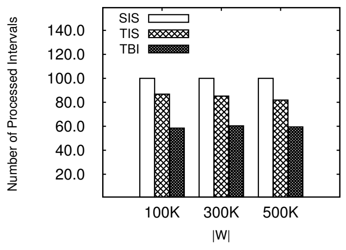
(a) Time



(b) I/O



(c) Top-k queries

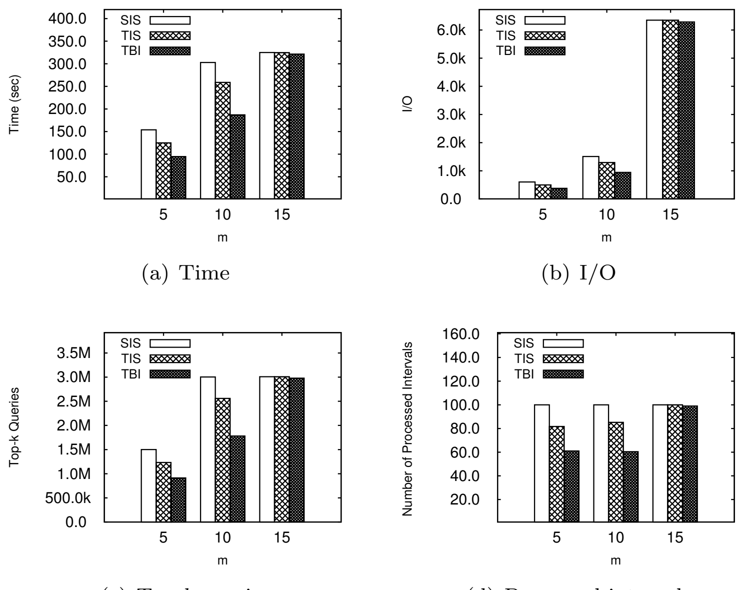


(d) Processed intervals

Figure 5.3: Effect of varying cardinality of weighting vectors $|W|$

However, since *TBI* chooses to process and split the largest unseen interval, it manages to reduce the upper bound of the objects significantly in the first steps of the algorithm's execution. In the best case, every $2^{\lambda+1} - 1$ steps the upper bound will have been reduced to $|V|/(2^{\lambda}+1)$, while for *TIS* the upper bound in the best case will have been reduced to $|V| - (2^{\lambda+1} - 1)$. Obviously, in the early steps of *TBI* the upper bound diverges the lower bound faster than in *TIS*.

Effect of varying cardinality of weighting vectors $|\mathbf{W}|$. In Figure 5.3, we study the effect of increasing the size of $|W|$. First, with respect to time (depicted in Figure 5.3(a)), we observe that time increases linearly with $|W|$ for all algorithms. This is expected, since the size of W determines the number of user preferences, which is the number of potential top-k queries that may be evaluated. In Figure 5.3(a), a similar trend is observed for the number of processed top-k queries, which also increase linearly with $|W|$. When the induced I/Os are considered, we see in Figure 5.3(b) that all algorithms show



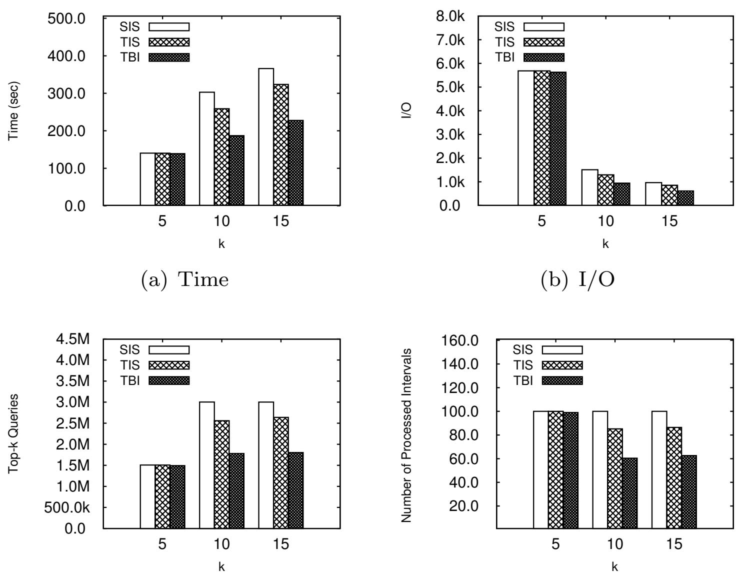
(a) Time (b) I/O

(c) Top-k queries (d) Processed intervals

Figure 5.4: Effect of varying m

a stable performance irrespective of $|W|$. Recall that we only measure the I/O induced on dataset S, and this metric does not depend on W. Hence, this explains the stability of the measured I/O values. Figure 5.3(d) shows the processed intervals by each algorithm. Also in this setup, *TBI* performs better than its competitors. It can be also observed that the size of W does not affect the number of processed intervals. The observations made for varying the data cardinality hold also here. The increased computation cost in respect of time is due to the fact that the complexity of the $ITOP_k^m$ queries increases when the weight cardinality rises.

Effect of varying m. Figure 5.4 shows the effect of increasing m, which is the number of retrieved influential objects, on the three algorithms. *TBI* has a significant performance advantage over *SIS* and *TIS* when the value of m is relatively small. When m increases, we observe that all algorithms demonstrate similar performance. The reason for this behavior is that for larger values



(a) Time (b) I/O

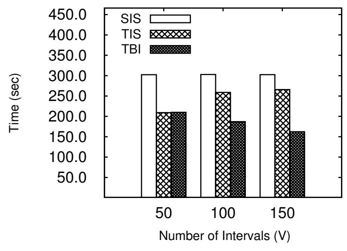
(c) Top-k queries (d) Processed intervals

Figure 5.5: Effect of varying k

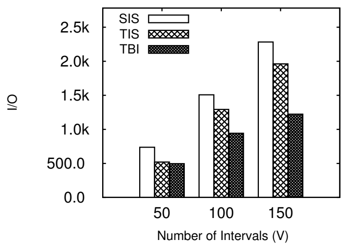
of m we observe that there exist data objects that have maximum continuity score equal to V. In other words, some data objects are influential in all V intervals. In this degenerate case, no algorithm can perform better than *SIS*, since all intervals must be processed in order to safely report the most continuous influential object.

Effect of varying k. Figure 5.5 depicts the obtained results for varying the value k of top-k queries. As k increases, all algorithms need more time to produce the result set as depicted in Figure 5.5(a).

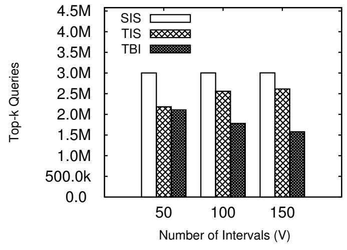
For smaller values of k, objects are not likely to appear in the results of a large number of queries, fact that reduces the influence score of all objects and naturally the influence score of the most influential objects. As a consequence, a large number of objects achieve the maximum influence score and as a result all algorithms perform similarly because again there exist objects with maxi-



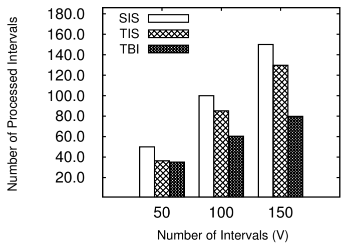
(a) Time



(b) I/O



(c) Top-k queries

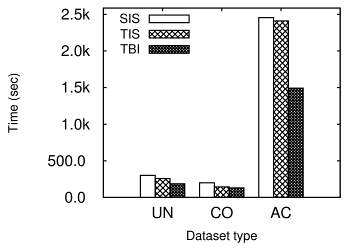


(d) Processed intervals

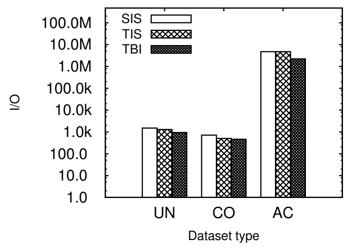
Figure 5.6: Effect of varying V

mum continuity score, which can only be reported when all intervals have been processed. For higher values of k *TBI* performs better than all other algorithms.

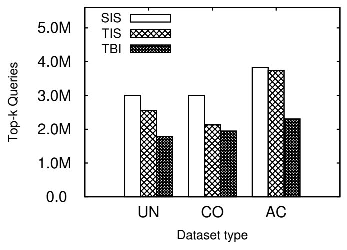
Effect of varying V. In Figure 5.6, the results of an experiment for increased number of intervals V are depicted. Based on Figure 5.6(a), we observe that *TIS* has a bigger advantage over *SIS* for small number of intervals, while *TBI* benefits more from large number of intervals. The reason is that the more the time intervals the smaller the possibility for an object to be influential in all of them. This fact is exploited by *TBI* which manages to reduce the upper bound fast in the first loops of its execution, and thus the lower bound and the upper bound converge fast and allow *TBI* to finish earlier that *SIS* and *TIS*. Contrary to the upper bound, the lower bound is expected to increase slowly when the time domain is partitioned with high granularity since many objects (including the one with the highest continuity score) are likely to disappear and re-appear



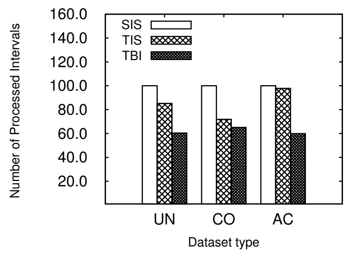
(a) Time



(b) I/O



(c) Top-k queries

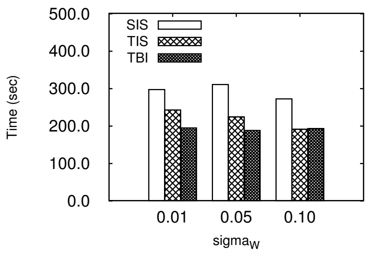


(d) Processed intervals

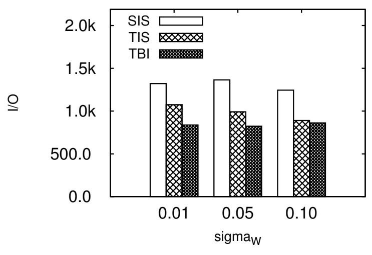
Figure 5.7: Effect of varying the data distribution of S

from the $ITOP_k^m$ influential sets, and consequently the convergence between the lower and upper bounds is delayed.

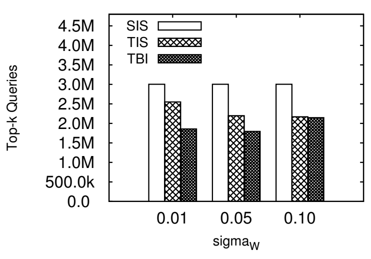
Effect of different data distributions of S. Figure 5.7 compares the performance of the three algorithms when the set of data objects S follow different distribution, namely uniform (UN), correlated (CO) and anti-correlated (AC). Notice that we use log-scale in Figure 5.7(b). Clearly, the cost of all algorithms (in terms of time and I/O) increases for AC. This is due to the more expensive processing of the underlying computation for influential data objects in the case of AC. However, as depicted in Figure 5.7(d), the difference between the algorithms is significant in terms of processed intervals. Also, notice that *TBI* is not significantly affected by the challenging AC data distribution and processes comparable number of intervals, irrespective of the data distribution of S.



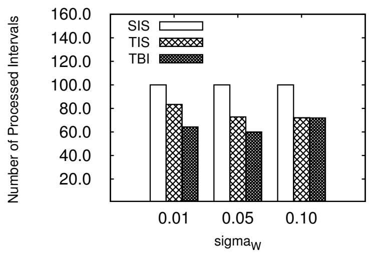
(a) Time



(b) I/O



(c) Top-k queries

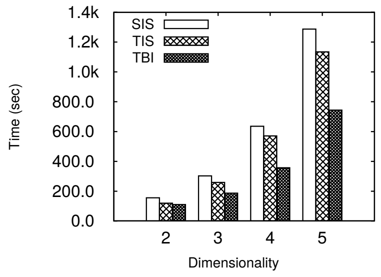


(d) Processed intervals

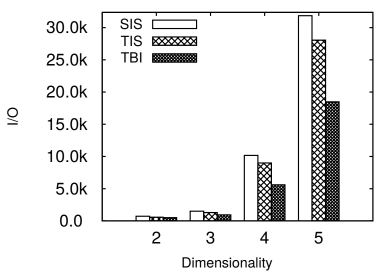
Figure 5.8: Effect of varying the standard deviation for clustered dataset *W*. Performance of the three algorithms for different values of standard deviation

Effect of clustered dataset W. Figure 5.8 shows the results of using a clustered dataset *W* for different values of σ_W. Smaller values of σ_W correspond to more clustered datasets, or in other words the weighting vectors are more compact with respect to the cluster centroids. For smaller values of σ_W, *TBI* performs better than the other algorithms. However, an interesting observation is that when σ_W increases, the performance of *TIS* tends to be similar to *TBI*.

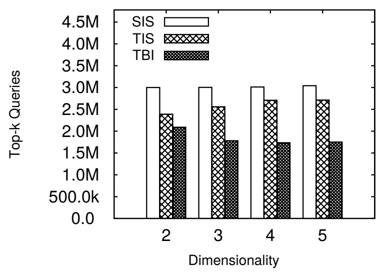
Effect of increasing dimensionality. Figure 5.9 illustrates the results for varying the number of dimensions. With respect to time (Figure 5.9(a)) and I/O (Figure 5.9(b)), the performance of all algorithms degrades with increased dimensionality. However, notice that *TBI* is less affected by the increased dimensionality, compared to the other algorithms. With respect to the number of



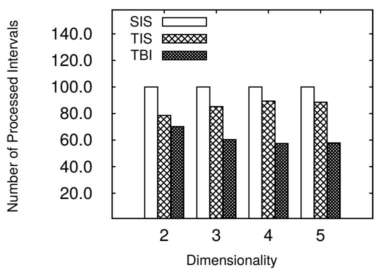
(a) Time



(b) I/O



(c) Top-k queries



(d) Processed intervals

Figure 5.9: Effect of varying dimensionality d

Algorithm	**Time(sec)**	**I/O**	**Top-k queries**	**Proc. Intervals**
SIS	710.23	7720	3011907	100.0
TIS	626.61	6801	2650585	86.9
TBI	417.40	4556	1779292	59.4

Table 5.3: Experimental results of real dataset NBA

processed top-k queries (Figure 5.9(c)) and number of processed intervals (Figure 5.9(d)), we observe that these metrics increase with dimensionality in the case of *TIS*. When *TBI* is considered, these metrics drop for increased values of n. This means that *TBI* manages to process fewer top-k queries and fewer intervals as n grows, however each top-k processing costs on average more for increased d, which explains why both time and I/O increase for *TBI* too.

Algorithm	**Time(sec)**	**I/O**	**Top-k queries**	**Proc. Intervals**
SIS	522.86	8056	3001495	100.0
TIS	500.53	7575	2902970	93.8
TBI	496.10	7555	2820868	91.8

Table 5.4: Experimental results of real dataset HOUSE

Experiments with real datasets. Tables 5.3 and 5.4 show the results obtained for the two real datasets employed in our study, namely NBA and HOUSE respectively. In both cases, the observed values closely follow the results and conclusions drawn from synthetic data. *TBI* outperforms the other two algorithms for both NBA and HOUSE. This gain is more clear in the case of NBA, where *TBI* needs almost half the time of *SIS* to identify the most continuous influential object.

5.8 Conclusion

In order to make users consider and buy products, visibility of products in online marketplaces is important. In this chapter, we have studied the problem of finding the products that belong consistently to the most influential products over time. The influence score is based on reverse top-k queries, and the aim has been to find those products that are among the most influential products over a longer time period, i.e., continuous influential products. In order to be able to determine efficiently those products, we have studied the properties of the proposed score and derived appropriate bounds that lead to efficient algorithms for solving the challenge. We have conducted a thorough experimental evaluation that demonstrated the efficiency of our algorithms.

Chapter 6

Finding the Most Diverse Products using Preference Queries

In the previous chapter, we discussed how user queries can be analyzed with respect to time in order to identify objects that are continuously appealing to the users. Although such objects are likely to attract a large number of users, companies such as product manufacturers typically address customers with a wide range of preferences. Products that are consistently popular, do not necessarily address users with diverse preferences. In this chapter, we study the problem of identifying sets of objects which are attractive to users with different preferences. We model this problem as a diversity problem, where each object is represented by its reverse top-k result set, and seek r objects that maximize their diversity value. Since the problem is NP-hard, we employ a greedy algorithm that takes as input the reverse top-k result sets of all candidate objects. To further improve performance, we also design a more efficient approximate algorithm that does not require the computation of all reverse top-k sets. Our experimental evaluation demonstrates the performance of the proposed algorithms and quality of the selected diverse objects.

User Preferences:

User	$w[1]$	$w[2]$	$w[3]$	**Top-k**
Bob	0.1	0.2	0.7	p_1
Tom	0.1	0.3	0.6	p_1
Jack	0.3	0.1	0.6	p_2
Max	0.8	0.1	0.1	p_3

Products:

Product	$p[1]$	$p[2]$	$p[3]$	**Reverse top-k**
p_1	1	2	6	Bob,Tom
p_2	2	1	6	Jack
p_3	6	5	2	Max

Table 6.1: Example of product database and user preferences

6.1 Introduction

Product manufacturers are interested in advertising to users a small set of items that will motivate them to browse the available products. Analyzing therefore the top-k queries posed by the users is necessary, in order to select and promote products that are appealing to a wide range of users. For instance, consider an electronic marketplace that wishes to advertise r products on its front page aiming to attract as many new customers as possible. Advertising diverse products that are attractive to different existing customers increases the probability that a new customer finds one of those products attractive. The strategy of advertising the r most influential products [112], i.e., the r products that attract the highest total number of customers, does not necessarily lead to a set of diverse products, and it may fail to attract many new customers, since such products may be attractive to customers with similar preferences.

Consider for example the set of user preferences and products depicted in Table 6.1, where maximum values in product attributes are preferable. Assume that the goal is to advertise two products for attracting new customers. Our proposed method selects the $r = 2$ most diverse products based on user preferences, which in our example is the set $\{p_1, p_3\}$. This set is more probable to attract more new customers because p_1 and p_3 satisfy more diverse preferences. For example, a customer with similar preferences to Jack is highly probable to be attracted also to p_1, even though it is not the best option for her on the market. This is because both p_1 and p_2 satisfy users that have high preference for the third dimension (expressed with a high weight $w[3]$). On the other hand, p_3 satisfies users that have totally diverse preferences compared to p_1 and p_2, namely, users such as Max that prefer the first dimension.

In this chapter, we introduce the problem of finding the r *most diverse* products based on user preferences. The user preferences are captured by the reverse top-k set of each product. We model this problem as a *dispersion* problem [89] using as distance function the dissimilarity of the reverse top-k sets. In this sense, the set of r objects with the maximum diversity is returned to the user. Consequently, the selected objects are appealing to many different customers with dissimilar user preferences. Existing solutions for identifying diverse objects rely solely on product attributes and largely overlook the user preferences [108]. On the other hand, approaches that identify r objects with high total number of customers [71, 112], often fail to discover truly diverse products that can be appealing to new customers with different preferences than those of the existing ones.

To summarize the main contributions are:

- We study the novel problem of finding the r most diverse products based on user preferences. We model this problem as a *dispersion* problem and define an appropriate distance function that captures the dissimilarity of products based on their reverse top-k sets.
- As dispersion problems are known to be NP-hard [33], we use a greedy algorithm that retrieves r diverse products, after computing the reverse top-k sets of the products efficiently.
- To improve the performance of our algorithm, we propose an alternative algorithm that progressively computes an approximation of the reverse top-k sets of a limited set of candidate products and retrieves a set of r products of high diversity.
- We present maintenance techniques for updating the r most diverse products in the case of dynamic data in a cost-efficient way. In addition, we generalize our approach to support any set-based similarity function.
- We demonstrate the efficiency and achieved diversity of our algorithms using both synthetic and real-life datasets.

The rest of this chapter is organized as follows: Section 6.2 reviews the related work. In Section 6.3, we formally define the r-Diversity problem. Thereafter, in Section 6.4, we present a greedy algorithm applied on the reverse top-k

sets. In Section 6.5, we provide a more efficient algorithm that iteratively computes an approximation of the reverse top-k sets and refines the set of most diverse products. Section 6.6 addresses the case of dynamic data, while Section 6.7 generalizes our approach for set-based similarity functions. The experimental evaluation is presented in Section 6.8, and we conclude in Section 6.9.

6.2 Related work

In this section, we provide an overview of the related research literature.

Reverse top-k queries. Vlachou *et al.* first introduced the reverse top-k query in [110]. Two versions of the reverse top-k query were presented, namely monochromatic and bichromatic. Based on the geometrical properties of the monochromatic reverse top-k query, an algorithm for the two dimensional case was proposed. For computing bichromatic reverse top-k queries, an algorithm (called *RTA*) was proposed that exploits the fact that similar queries share common results, in order to avoid evaluating the top-k queries for all user preferences. Thereafter, several papers have studied the problem of efficient reverse top-k computation. An efficient algorithm for the two-dimensional monochromatic reverse top-k that relies on a novel index was proposed in [20]. In [40], efficient evaluation of multiple top-k queries is studied, which in turn enables the computation of the reverse top-k set of a query point. The proposed method avoids evaluating the top-k queries one-by-one by grouping similar queries and evaluates them in a batch. This approach is suitable for processing many reverse top-k queries at once. An approach for processing a large number of continuous top-k queries has appeared in [125]. The proposed framework can be employed to process reverse top-k queries efficiently, however, it requires to build an index over the k-th ranked objects of each query that results in high pre-processing cost. Vlachou et al. [113] proposed a novel branch-and-bound algorithm for reverse top-k queries, where both the object datasets and the preferences set are indexed using an R-tree.

Product impact and visibility. Several papers have proposed methods that aim to quantify the impact of products in the market. DADA [69] aims to help manufactures position their products in the market, based on three types of dominance relationship analysis queries. Creating competitive products has been studied in [115]. Customer identification and product positioning has been recently studied in [7], where the attractiveness of a product is defined based on the concept of reverse skyline query. Nevertheless, in these approaches user preferences are expressed as data points that represent preferable products, whereas reverse top-k queries examine user preferences in terms of weighting vectors. Miah *et al.* [81] study a different problem, namely how to select the subset of attributes that increases the visibility of a new product. Product promotion is studied in [118, 119], where the aim is to find the most interesting regions for promotion of a product. Only a few papers have proposed methods for retrieving interesting products by using the reverse top-k queries. In [112], the influence of a product is defined as the size of its reverse top-k set. Then, an algorithm was presented to efficiently retrieve the m most influential products. Discovering k products with maximum number of customers has been studied in [71], where the number of customers is estimated as the size of the reverse top-k set. The problems studied in [71, 112] differ from the diversity problem described in this chapter. Both approaches focus on maximizing the number of existing customers and ignore the similarity of the retrieved reverse top-k sets. These approaches fail to take into account the fact that attracting new customers requires promoting products that are attractive to customers with diverse preferences. Koh *et al.* [65] consider as products packages consisting of multiple components. They study the problem of creating and selecting packages from an existing pool of components such that the number of potential customers is maximized. Similarly to the aforementioned approaches, the number of potential customers is estimated using reverse top-k sets, yet they do not study the diversity of the result set.

Diversity in databases. Many approaches have been proposed for retrieving a set of diverse objects. Angel *et al.* [6] study the problem of retrieving k documents relevant to a query q, but are also diverse with each other. The diversity is computed based on document similarity metrics. Drosou *et al.* [32] study the problem of finding the k most diverse objects in a continuous data stream. DivDB, a system that provides query result diversification, was presented in [109]. Result diversification based on dissimilarity is studied also in [31]. Estimating the diversity of a set of points that fulfill a special property has been studied mainly for selecting representative skyline points. For instance the diversity of two skyline points can be defined as the distance between them [102] or by using their sets of dominated points [72, 108]. More specifically, in [102] the authors define the set of representative skyline to be a set of k objects that maximize the minimum Euclidean distance between any two of the k points. In [72], the representative skyline points are defined based on the distinct number of domi-

Symbol	**Description**
S	Set of data objects
D	Subset of S ($D \subseteq S$)
p,q	Data objects/products ($p, q \in S$)
W	Set of weighting vectors
w	A weighting vector ($w \in W$)
f_w	Preference function associated with w
k	Value of top-k
$TOP_k(w)$	Top-k data objects based on w
$RTOP_k(p)$	Reverse top-k result set for object p
$\mathbf{c}_p$	Centroid of vectors in set $RTOP_k(p)$
$\delta(p, q)$	Cosine distance between centroids $\mathbf{c}_p, \mathbf{c}_q$
$\delta(u, v)$	Cosine distance between vectors u, v
$div(D)$	Diversity value of a set of objects D
D^*	Optimal solution of the r-Diversity problem
$D_r(S)$	Approximate solution of the r-Diversity problem

Table 6.2: Overview of symbols

nated points. Valkanas *et al.* [108] estimate the diversity of two skyline points by calculating the Jaccard distance of their respective sets of dominated points. The main difference to our work is that the definitions of diversity in the above approaches rely on the attribute values only and cannot exploit the existing user preferences.

6.3 Problem definition

Given a space $\mathbb{R}^d$, we assume that we have a set of data objects S where each object $o \in S$ can be represented as an d-dimensional point $o = (o[1], \dots, o[d])$ where $o[i] \in \mathbb{R}$. Each point o can be regarded as an object of a database and each dimension of the point as a specific numerical attribute. Without loss of generality, we assume that larger values are preferable.

We model a user query as a top-k query where the result set of each user query is a ranked set of k objects which have the highest score according to a scoring function $f : S \to \mathbb{R}^+$. A function commonly used is the linear function of the form $f(o) = \sum_{i=1}^{d} w[i]o[i]$ where $w[i] \geq 0$. Such functions can be represented by an d-dimensional *weighting vector* $w = (w[1], \ldots, w[d])$. In such cases we denote the function that results from w as f_w.

When w represents the preferences of a user over the objects in S we call this vector *preference vector* or simply *preference*. We remind at this point that if we have a set of preferences $W \subseteq \mathbb{R}^d$ over a set of products $S \subseteq \mathbb{R}^m$ then for a given product q, we say that the result set of a *reverse top-k* query is a set $RTOP_k(q)$ that consists of all the preference vectors w for which it holds that $q \in TOP_k(w)$.

Let p and q denote two products (data objects) from a product database S. Also, given a set W of customer preferences (weighting vectors) and an integer k, let $RTOP_k(p) \subseteq W$ and $RTOP_k(q) \subseteq W$ denote the reverse top-k sets of p and q respectively. We also define a distance function $\delta : S \times S \rightarrow \mathbb{R}^+$ as:

$$\delta(p, q) = f_\delta(RTOP_k(p), RTOP_k(q))$$

that determines the dissimilarity of any two objects p and q based on their corresponding reverse top-k sets. Notice that this is a radically different approach from existing initiatives that define the distance of two objects based on the objects' attributes only.

The problem of selecting the r most diverse products from a given set S can be viewed as a *dispersion* problem [32, 33, 89, 108], where the aim is to find r objects such that an objective function of their distance δ is optimized. The *dispersion sum problem* maximizes the sum of pairwise distances between the r selected products and it has been proved that it is NP-hard by reduction from the clique problem [33].

Definition 6.1. ***r-Diversity Problem.*** *Given a set of data objects S and a distance function δ measuring the dissimilarity between two objects, the r-Diversity problem is to identify a subset $D^* \subseteq S$ such that:*

$$D^* = \underset{\substack{D \subseteq S \\ |D| = r}}{\arg\max} \sum_{\substack{p,q \in D \\ p \neq q}} \delta(p, q)$$

The remaining challenge is to define an appropriate function f_δ that captures the dissimilarity of the reverse top-k result sets. Hence, the function f_δ takes as input two sets of weighting vectors and computes their dissimilarity. We employ a function that relies on the concept of a *centroid* of a set of vectors.

Definition 6.2. ***Centroid of RTOPk.*** *Given a set of data objects S, a set of weighting vectors W, and an object $p \in S$ such that $RTOP_k(p) \neq \emptyset$, we define as the* centroid *of p the vector:*

$$\mathbf{c}_p = \frac{1}{|RTOP_k(p)|} \sum_{w \in RTOP_k(p)} w$$

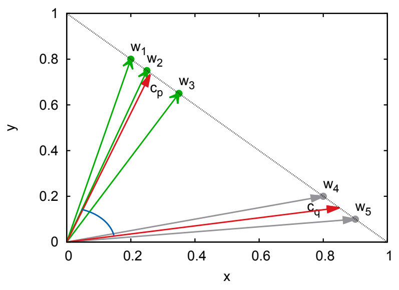


Figure 6.1: Example of dissimilarity function

Since each $RTOP_k$ set corresponds to exactly one data point, the respective centroid corresponds to exactly one data point as well. Therefore, each data point can be mapped to exactly one centroid vector and vice-versa.

Definition 6.3. ***Dissimilarity function*** $\mathbf{f}_\delta$. *Given a set of data objects S, a set of weighting vectors W, two objects $p, q \in S$, and their respective centroids $\mathbf{c}_p$ and $\mathbf{c}_q$, the distance of p and q is defined based on the cosine similarity of the centroids:*

$$f_\delta(RTOP_k(p), RTOP_k(q)) = 1 - cos(\mathbf{c}_p, \mathbf{c}_q)$$

The advantage of using the centroid $\mathbf{c}_p$ instead of the actual set of vectors $RTOP_k(p)$ is that the centroid is a compact and accurate representation of the set, which in turn allows efficient processing of the dissimilarity function, compared to other dissimilarity metrics that operate on sets of arbitrary size. As a distance function, we use the function $\delta(p, q) = 1 - cos(\mathbf{c}_p, \mathbf{c}_q)$. In a slight abuse of notation, we also use $\delta(u, v) = 1 - cos(u, v)$ to denote the cosine distance between any two vectors u and v.

Example 6.1. *Figure 6.1 shows an example of the reverse top-k sets $RTOP_k(p) = \{w_1, w_2, w_3\}$ and $RTOP_k(q) = \{w_4, w_5\}$, which belong to products p and q respectively. In the Euclidean space, a linear top-k query can be represented by a vector w [110]. The magnitude of the query vector does not influence the query result, as long as the direction remains the same, therefore without loss of generality we assume that $\sum_{i=1}^{d} w[i] = 1$. In the 2-dimensional space, all weighting vectors belong to the line as depicted in Figure 6.1. Moreover, top-k queries defined by similar weighting vectors w are expected to produce similar result sets [110]. Thus, the weighting vectors of the reverse top-k set of p are expected to lie nearby on the line. Furthermore, for a hypothetical weighting* *vector which lies on the line between w_1 and w_3, it is expected that p is highly ranked, and therefore it is highly probable that this vector would belong to the reverse top-k set of p.*

The centroid of the weighting vectors captures the above intuitions, and the angle between two centroids represents the dissimilarity of the weighting vectors. Obviously, different functions for set dissimilarity (hence also for measuring distance) are supported by our approach, including (for instance) the Jaccard similarity of the reverse top-k sets. Nevertheless, the Jaccard similarity fails to capture the locality of the weighting vectors.

Furthermore, we define the *diversity* $div(D)$ of a set of objects $D \subseteq S$. Notice that the set D^* with the highest diversity value $div(D^*)$ among all r-sets of points in S, is the optimal solution for Problem 6.1. The diversity value $div(D)$ is normalized in [0,1].

Definition 6.4. ***Diversity value*** *Given a set of points S, a subset $D \subseteq S$ of size r, and set of vectors W, we define as* diversity *of D:*

$$div(D) = \frac{2}{r(r-1)} \sum_{\substack{p,q \in D \\ p \neq q}} (1 - cos(\mathbf{c}_p, \mathbf{c}_q))$$

6.4 Algorithms with centroid computation

The process of discovering r diverse products $D_r(S)$ from a set of products S consists of two main steps: (1) identifying a set C of *candidate centroids* that correspond to candidate products for inclusion in the most diverse products (Section 6.4.1), and (2) selecting r of these candidates as the most diverse products (Section 6.4.2).

Each candidate centroid in $\mathbf{c}_p \in C$ corresponds to exactly one product $p \in S$, and it is the centroid vector of the $RTOP_k(p)$ set of p. More formally $C = \{\mathbf{c}_p | p \in S, RTOP_k(p) \neq \emptyset, \mathbf{c}_p \text{ is centroid of } RTOP_k(p)\}$. Obviously, products that are not preferable for any customer are ignored.

Algorithm 12 describes the afore-described method and returns a set of r diverse products. In line 1, the candidate centroids C are computed using any of the methods that will be described in Section 6.4.1. As the set of centroids C may be large depending on the data distribution, a sample R of fixed size s is created by picking centroids uniformly at random (line 2). Finally, in line 3, the second step entails solving the r-Diversity problem by applying a greedy algorithm, called Diverse Product Selection Algorithm (*DPSA*), on the sampled set of centroids R, as will be described in Section 6.4.2.

Algorithm 12 r-Diverse Products

```
Input: $S$: set of products
       $W$: set of weighting vectors
       $k$: value of top-$k$ and reverse top-$k$
       $s$ : size of initial sample
       $r$ : required number of diverse products
Output: $D_r(S)$: the set of $r$ most diverse products of $S$
1: $C \leftarrow$ CandidateCentroids($S$, $W$, $k$)
2: $R \leftarrow$random subset of $C$ with $|R| = s$
3: $D_r(S) \leftarrow DPSA(C, R, r)$
4: return $D_r(S)$
```

6.4.1 Retrieving the candidate centroids

Different alternatives exist in order to compute the set C of candidate centroids. In the following, we present three alternative methods for determining the set C. Notice that all methods produce an identical set C of centroids.

The most straightforward method is to perform a reverse top-k query for each product p in S and compute the centroid vector of each set $RTOP_k(p)$ using Definition 6.2. We denote this approach *Rtopk*. Its processing cost is basically determined by the computation of $|S|$ reverse top-k queries. Since any existing algorithm for reverse top-k processing can be employed for the underlying reverse top-k computation, this method is quite generic.

An improvement of the first method is derived based on the observation that some products have empty reverse top-k sets (i.e., they do not belong to the top-k result of any weighting vector). Hence, it is possible to avoid processing some reverse top-k sets. To achieve this, we exploit the progressive result generation of the algorithm in [112], which is able to retrieve objects in decreasing order of the sizes of their reverse top-k sets. We denote this method as *Itopk* based on the fact that the algorithm [112] has been proposed for retrieval of influential objects. As a result, we avoid processing a reverse top-k query for objects with empty reverse top-k sets, thus improving the performance of *Rtopk*.

The third method exploits the observation that it may be more efficient to process all top-k queries, instead of processing multiple reverse top-k queries. Thus, we perform a top-k query for each preference vector $w \in W$, which makes straightforward the computation of the reverse top-k sets of any data object, and hence also their respective centroids. In fact, the top-k sets do not need to be maintained until all top-k queries have been processed, but instead the centroids can be calculated progressively. For each retrieved object in the top-k result set, the centroid is updated by adding the new vector $\mathbf{w}$ to its previous centroid, while also the number of vectors per object is maintained. After finishing all top-k queries, for each centroid the coordinates are divided by the cardinality of the reverse top-k set. Since top-k queries for all vectors in W are processed, we call this method all top-k, i.e., *Atopk*. An advantage of *Atopk* is that the processing cost in terms of top-k evaluations is fixed, namely $|W|$ top-k queries, in contrast to *Rtopk* and *Itopk* where in the worst case the top-k evaluations can be up to $|W| \cdot |S|$. Thus, the efficiency of *Atopk* is influenced slightly by the cardinality of S, in contrast to *Rtopk* which computes the reverse top-k set even for products with empty reverse top-k sets.

6.4.2 Diverse product selection algorithm

After having computed the centroid vectors of all non-empty reverse top-k sets, the next step is to find the r most diverse centroids and the products that they represent. As already mentioned, the r-Diversity problem is defined as a dispersion problem that is known to be NP-hard [33]. Thus, computing the optimal solution for the r-Diversity problem is not feasible even for relatively small datasets. Hence, we employ an algorithm that efficiently computes an approximate solution of high quality [29]. More specifically, we use a greedy algorithm, called Diverse Product Selection Algorithm (*DPSA*), that iteratively selects the next centroid that maximizes the value of the objective function. Its pseudocode is depicted in Algorithm 13.

Description. The algorithm takes as input the set of candidate centroids C, a random sample set R of the candidate centroids that is going to be used, and an integer r which is the desired number of most diverse products. It returns an approximate set $D_r(S)$ of the r most diverse products and their centroids. The sample R is typically much smaller in size than C, in order to reduce the cost of the first part of the algorithm, which is to find the two most distant vectors in R (line 2) and add them to the result set $D_r(S)$ (line 3). Then, the algorithm iteratively selects the next centroid $\mathbf{c}_q$ until r centroids have been retrieved (loop in line 4). Each time, the selected centroid is the one that maximizes the sum of distances from the already selected most diverse vectors $D_r(S)$. Notice that R is used only for the initialization of $D_r(S)$ (line 3), while the remaining centroids are selected from C.

Algorithm 13 Diverse Product Selection Algorithm *DPSA*()

```
Input: $C$ : set of candidate centroids
       $R$ : sample of $C$
       $r$ : required number of diverse products
Output: $D_r(S)$: the set of $r$ most diverse products of $S$
1: $\mathbf{c}_{p1}, \mathbf{c}_{p2} \leftarrow \mathbf{c}_{p1}, \mathbf{c}_{p2} : \forall \mathbf{c}_{pi}, \mathbf{c}_{pj} \in R : \delta(\mathbf{c}_{p1}, \mathbf{c}_{p2}) \geq \delta(\mathbf{c}_{pi}, \mathbf{c}_{pj})$
2: $C \leftarrow C - \{p_1, p_2\}$
3: $D_r(S) \leftarrow \{p_1, p_2\}$
4: while $|D_r(S)| < r$ do
5:     $\mathbf{c}_q = \underset{\mathbf{c}'_q \in C}{\arg\max} \left( \sum_{p \in D_r(S)} \delta(\mathbf{c}'_q, \mathbf{c}_p) \right)$
6:     $D_r(S) \leftarrow D_r(S) \bigcup \{q\}$
7:     $C \leftarrow C - \{q\}$
8: end while
9: return $D_r(S)$
```

Complexity. The selection of the two most diverse products (seeds) has a cost $O(|R|^2) = O(s^2)$. The remaining part of the algorithm has a cost of $O(r^2|C|)$ and therefore the total cost is equal to $O(s^2 + r^2|C|)$. If no sample is used $(s = |C|)$ in the seed selection then the algorithm will have a cost of $O(|C|^2)$.

Implementation details. In each loop iteration of the *DPSA* algorithm (lines 4-8), the algorithm calculates the sum of distances between a centroid vector $\mathbf{c_q} \in C - D_r(S)$ and the centroid vectors in $D_r(S)$. As described above this procedure has a cost of $O(r^2|C|)$. In the case of the cosine distance, we can reduce this cost to $O(r|C|)$ by exploiting the properties of the cosine function. As shown in Equation 6.1 the sum of distances of $\mathbf{c}_q$ to all centroids in $D_r(S)$ is equal to $|D_{r(S)}| - \frac{\mathbf{c}_q}{|\mathbf{c}_q|} \cdot \mathbf{c}_{D_r(S)}$. In that way, it is only necessary to calculate the centroid of $D_r(S)$ before each loop iteration.

$$\begin{aligned}
\sum_{p \in D_r(S)} \delta(\mathbf{c}_q, \mathbf{c}_p) &= \sum_{p \in D_r(S)} 1 - cos(\mathbf{c}_q, \mathbf{c}_p) \\
&= |D_r(S)| - \frac{\mathbf{c}_q}{|\mathbf{c}_q|} \cdot \sum_{p \in D_r(S)} \frac{\mathbf{c}_p}{|\mathbf{c}_p|} \\
&= |D_r(S)| - \frac{\mathbf{c}_q}{|\mathbf{c}_q|} \cdot \mathbf{c}_{D_r(S)}
\end{aligned} \tag{6.1}$$

6.5 Selective top-k algorithm

The main drawback of the previous algorithm is that it requires the computation of all centroids, which has a significant processing cost regardless of the employed method for candidate centroid computation. In particular, depending on the cardinality of W and S, the computation of the centroids may be time-consuming. In order to alleviate this shortcoming, in this section, we propose a method that fuses the centroid computation with the selection of diverse objects. Our goal is to efficiently compute an approximation of the centroids (by evaluating only a handful of carefully selected top-k queries), which is sufficient to produce a set of r products with high diversity.

6.5.1 Centroid approximation

Conceptually, the proposed algorithm uses a series of iterations, where each iteration consists of three parts: (1) select a weighting vector w_i in order to process the top-k query it defines, (2) compute an approximation $\widehat{\mathcal{C}}$ of the centroid-set $\mathcal{C}$, based on the results of all already processed top-k queries, and (3) select diverse products by invoking the *DPSA* algorithm (Section 6.4.2) with input the approximate centroid-set. In each iteration, a top-k query based on w_i is executed. Some objects $p \in TOP_k(w_i)$ may not have been retrieved before and those are added to the centroid-set $\widehat{\mathcal{C}}$. For the remaining objects $p \in TOP_k(w_i)$ the approximate centroid is updated, since w_i is added to their reverse top-k sets. In fact, the reverse top-k sets are not maintained, but the centroid of an object is computed progressively as in the case of *Atopk*. Thus, in each iteration the centroid-set is only an approximation of the candidate centroids C computed by Algorithm 12 because (a) C may contain more centroids as some objects may not have been retrieved yet and (b) the centroids of an object p are estimated based on a limited set of top-k queries only. However, in each iteration, the candidate-set is enriched with the results of the next top-k query. Additionally, a set of r diverse products $D_r(S)$ is computed based on the current set of centroids. Finally, the selection of the next weighting vector to be processed is based on maximizing the sum of distances to the set of centroids defined by $D_r(S)$.

The main idea of our algorithm is that the maximum cosine distance (i.e., maximum diversity) of two objects is bounded by the maximum cosine distance of any two weighting vectors. Let us assume that w_1 and w_2 are the two weighting vectors with the maximum cosine distance (the most diverse). Let us further assume that there exist two products p_1 and p_2 for which holds:

Algorithm 14 Selective Top-k Algorithm

```
Input: $S$: set of products
       $W$: set of weighting vectors
       $k$: value of top-$k$ and reverse top-$k$
       $s$ : size of initial sample
       $r$ : required number of diverse products
       $steps$ : number of iterations ($steps \geq r$)
Output: $D_r(S)$: the set of $r$ most diverse products of $S$
 1: $W' \leftarrow$random subset of $W$ with $|W'| = s$
 2: $w_1, w_2 \leftarrow w_1, w_2 : \forall w_i, w_j \in W' : \delta(w_1, w_2) \geq \delta(w_i, w_j)$
 3: $\widehat{\mathcal{C}} \leftarrow$ ComputeCentroids($\bigcup_{x=1,2} TOP_k(w_x)$)
 4: $D_r(S) \leftarrow DPSA(\widehat{\mathcal{C}}, \widehat{\mathcal{C}}, 2)$
 5: $i = 2$
 6: while $i < steps$ do
 7:     $i++$
 8:     $w_i = \underset{w \in W}{\arg\max} \left(\sum_{p \in D_r(S)} \delta(\mathbf{c}_p, w)\right)$
 9:     $\widehat{\mathcal{C}} \leftarrow$ ComputeCentroids($\bigcup_{x=1 \dots i} TOP_k(w_x)$)
10:     $D_r(S) \leftarrow DPSA(\widehat{\mathcal{C}}, \widehat{\mathcal{C}}, min(i+1, r))$
11: end while
12: return $D_r(S)$
```

$RTOP_k(p_1) = \{w_1\}$ and $RTOP_k(p_2) = \{w_2\}$. Then, it holds that for 2-Diversity problem the optimal solution is $\{p_1, p_2\}$ and their diversity equals to $1 - cos(w_1, w_2)$, since $\mathbf{c}_{p_i} = w_i$. If more weighting vectors belong to $RTOP_k(p_1)$ then the diversity between $\{p_1, p_2\}$ decreases. Therefore, our algorithm starts by evaluating the top-k queries for the most diverse weighting vectors. In each step, the most diverse weighting vector to the current most diverse centroids is selected, as each centroid may summarize several weighting vectors.

6.5.2 Algorithmic description

Algorithm 14 shows the pseudocode of the proposed algorithm that uses a limited set of top-k queries only. We call this algorithm *Selective Top-k Algorithm* and denote it with *Stopk*.

Description. The first major difference to Algorithm 12 is that the initial centroid computation is avoided. First, the algorithm computes a random sample W' (of size s) of W (line 1) and the two most dissimilar weighting vectors w_1 and w_2 of W' are selected (line 2). Notice that the sample W' is produced uniformly at random, thus, it follows the distribution of W and is used only for the initialization of $\widehat{\mathcal{C}}$. Applying the initialization step on W would result in a cost of $O(|W|^2)$, while with the sample it is reduced to $O(|W'|^2)$. Next, the top-k queries for w_1 and w_2 are processed and a set $\widehat{\mathcal{C}}$ of centroids is computed from the resulting merged set of products (line 3). Notice that $\widehat{\mathcal{C}}$ is computed based solely on the products retrieved thus far by the two top-k queries. These centroids form the candidate set for finding the most diverse products. In the following step, Algorithm 13 is invoked with input the candidate set, and the two most diverse products are retrieved and placed in $D_r(S)$ (line 4). Note that $\widehat{\mathcal{C}}$ is much smaller than C, thus *DPSA* algorithm is applied on $\widehat{\mathcal{C}}$ and no random sample is required.

In each iteration, the most dissimilar weighting vector w_i to the centroid vectors $\mathbf{c}_p$ ($p \in D_r(S)$) is selected (line 8). For this w_i, the respective top-k query is executed and the candidate list $\widehat{\mathcal{C}}$ is updated as before (line 9). Then, the *DPSA* algorithm is invoked again to produce a new set of diverse products (line 10). The same procedure is repeated for at least r steps. Notice that when the iteration counter i is smaller than r, the algorithm produces i diverse products, and only when i becomes greater than r does the algorithm return r diverse products.

In order to improve further the approximation of the centroids more iterations can be applied. The number of iterations (*steps*) is a system parameter that captures an interesting trade-off between the diversity of the result set and the processing time. Small values of *steps* increase the efficiency of the algorithm by reducing its processing time. In the experimental evaluation, we demonstrate that a small number of iterations is sufficient to produce results with diversity comparable to that of Algorithm 12, with significantly lower processing cost. Notice that in the extreme case that the number of iterations of *Stopk* is set equal to the cardinality of W and also no sampling is used ($s = |W|$), the set of diverse products and number of required top-k queries will be the identical with *Atopk*. Nevertheless, in this case, *Stopk* will have the computational overhead of applying multiple times the *DPSA* algorithm and finding the diverse weighting vectors.

Example 6.2. *Table 6.3 shows an example of the execution of Stopk for* $r = 2$. *We assume that in the initialization step vectors* w_1 *and* w_2 *are selected. Fur-*

Initialization step:
$TOP_k(w_1) = p_1, p_2, p_3$, $TOP_k(w_2) = p_2, p_4, p_5$ $\mathbf{c}_{p_1} = w_1$, $\mathbf{c}_{p_2} = \frac{1}{2}(w_1 + w_2)$, $\mathbf{c}_{p_3} = w_1$, $\mathbf{c}_{p_4} = w_2$, $\mathbf{c}_{p_5} = w_2$
First step:
$TOP_k(w_3) = p_3, p_4, p_5$ $\mathbf{c}_{p_1} = w_1$, $\mathbf{c}_{p_2} = \frac{1}{2}(w_1 + w_2)$, $\mathbf{c}_{p_3} = \frac{1}{2}(w_1 + w_3)$ $\mathbf{c}_{p_4} = \frac{1}{2}(w_2 + w_3)$, $\mathbf{c}_{p_5} = \frac{1}{2}(w_2 + w_3)$
Second step:
$TOP_k(w_4) = p_1, p_2, p_6$ $\mathbf{c}_{p_1} = \frac{1}{2}(w_1 + w_4)$, $\mathbf{c}_{p_2} = \frac{1}{3}(w_1 + w_2 + w_4)$, $\mathbf{c}_{p_3} = \frac{1}{2}(w_1 + w_3)$, $\mathbf{c}_{p_4} = \frac{1}{2}(w_2 + w_3)$, $\mathbf{c}_{p_5} = \frac{1}{2}(w_2 + w_3)$, $\mathbf{c}_{p_6} = w_4$

Table 6.3: Example of *Stopk*

thermore (assuming $k = 3$), the top-3 results for the selected vectors are depicted. After the initialization step, the sets of approximate centroids $\widehat{\mathcal{C}}$ contains 5 centroids (namely $\mathbf{c}_{p_1}$, ..., $\mathbf{c}_{p_5}$), which correspond to the data points that have been retrieved by at least one top-k query. Algorithm 13 is applied on $\widehat{\mathcal{C}}$ and we assume that $\mathbf{c}_{p_1}$ and $\mathbf{c}_{p_4}$ are the two most diverse vectors. In the first iteration of Stopk, the most diverse (according to $\mathbf{c}_{p_1}$ and $\mathbf{c}_{p_4}$) weighting vector of W is selected. In this step, w_3 is selected and the approximate centroids $\widehat{\mathcal{C}}$ are updated based on $TOP_k(w_3)$ as depicted in Table 6.3. Again, Algorithm 13 is applied on $\widehat{\mathcal{C}}$ and $\mathbf{c}_{p_1}$ and $\mathbf{c}_{p_4}$ are identified as the two most diverse vectors. Stopk continues with a second iteration by evaluating w_4. In this step, $\mathbf{c}_{p_6}$ is added to $\widehat{\mathcal{C}}$ as it belongs to $TOP_k(w_4)$. Again, Algorithm 13 will be invoked and the most dissimilar weighting vector of W will be selected. The same procedure continues until steps iterations have been executed.

Complexity. To perform a cost analysis of the algorithm, we need to identify its basic cost factors. These factors include the initial computation of the two most dissimilar vectors ($O(s^2)$), the processing cost of *steps* top-k queries, the cost of determining the next most dissimilar weighting vector ($O(steps \cdot r \cdot |W|)$) (line 8), and the cost induced by invoking the *DPSA* algorithm *steps* times. The cost of processing *steps* top-k queries will always be much cheaper than

Algorithm 12, which needs to process W top-k queries (in the case of *Atopk*) to perform the centroid computation. It should also be noted that the calls to the *DPSA* algorithm are much cheaper, because it operates on $\widehat{C}$ which is much smaller than C. Overall, the cost of the algorithm is $s^2 + steps \cdot r \cdot |W|$, since these are the dominant cost factors, and the complexity is linear with respect to W ($O(steps \cdot r \cdot |W|)$), when *steps* is small (r is typically small anyway).

6.6 Maintenance

In this section, we present techniques for maintaining the diverse set of products in the case of dynamic data. In fact, the methodology of *Stopk* (Algorithm 14) can be applied to maintain the r-diverse products. We consider two cases: (1) new products are inserted in the product database, and (2) new preferences (in the form of weighting vectors) are added in the customer preference database. Both cases actually occur in online shops, when new products appear in the market and new customer preferences are extracted from social sites.

In order to support product insertions efficiently, we exploit the top-k queries that where computed during the computation of $\widehat{C}$. Let W^* be the set of weighting vectors for which the top-k queries have been computed. We maintain for each weighting vector $w \in W^*$ the score of the k-th product. When a new product p_{new} is inserted in the database, we check for each query $w \in W^*$ if p_{new} has a better score than the k-th product. If this does not occur for any $w \in W^*$, we can safely ignore p_{new}, as it does not affect the determination of the diverse products. On the other hand, if p_{new} becomes top-k for some weighting vector w, we compute the new centroids only for the affected products (i.e., p_{new} and the products p_i that used to be at the k-th rank, but were evicted by p_{new}) and update the set $\widehat{C}$. We then apply *DPSA* algorithm on $\widehat{C}$ and produce a new set of diverse products. Note that in the first case, we can ensure that the result set is the same as in the case where *Stopk* would be executed on the updated dataset, but this does not hold for the second case. The similarity of the centroids before and after the update can be used in order to decide when the *Stopk* algorithm should be invoked to have a result set of higher quality.

In order to be able to handle new preferences effectively, during the computation of the diverse product we maintain the minimum (*min*) of all maximal sums of distances between a centroid and any selected weighting vector (i.e., the expression in line 8 of Algorithm 14). In the case of a new weighting vector w_{new}, we follow the same principle as Algorithm 14 to decide whether the respective top-k query should be evaluated. If $\sum_{p \in D_r(S)} d(\mathbf{c}_p, w_{new})$ is larger than *min*, then the top-k query for w_{new} is computed, the set of centroids $\widehat{C}$ updated and *DPSA* algorithm is executed on $\widehat{C}$ to produce a new set of diverse products. Intuitively, when vector w_{new} induces small changes to the set of centroids, we do not need to recompute the r-diverse products as w_{new} would not have been selected by *Stopk* in any case. Again, the retrieved diverse products are not the same as if *Stopk* would be executed on the updated weighting vector set, therefore a threshold on the similarity of the centroids before and after the update may trigger executing *Stopk* to get a set of higher quality.

6.7 Supporting other set-based similarity functions

In general, our approach is applicable also for other functions that compute the similarity between sets of vectors. In such a case, our algorithms would not calculate centroids (which is simply a representation of a set of weighting vectors), but would instead directly operate on the reverse top-k sets of products. Following this line of thought, $\widehat{C}$ would represent a set of approximate reverse top-k sets (instead of a set of centroids) and the computation of the most diverse sets becomes independent of the similarity or distance function.

In more technical terms, Algorithm 12 would not calculate centroids but would only maintain the reverse top-k sets, and Algorithm 14 would not compute centroids incrementally but would simply update the approximate reverse top-k sets of products based on the executed top-k queries. Then, Algorithm 13 can be directly applied to the reverse top-k sets.

For instance, one popular similarity function is the Jaccard similarity, which is defined as the size of the intersection divided by the size of the union of two sets. Our approach readily supports the Jaccard similarity on reverse top-k sets, as outlined above. Notice that the advantage of the cosine similarity compared to Jaccard for the problem of finding diverse products is that it returns more fine-grained similarity values. For example, in the case of disjoint sets, the Jaccard similarity value equals to zero, and does not distinguish between the sets. Instead, the cosine similarity of the centroid vectors allows us to distinguish between them by returning a non-zero value. Moreover in the case of a set A that is a subset of another set B ($A \subseteq B$), the Jaccard similarity is equal to $\frac{|B|-|A|}{|B|}$ which can get arbitrarily close to the maximum value. In such cases, using the Jaccard similarity would not be helpful, as it would lead to selecting A product that is covered by another one. The centroid vectors reduce this

Parameter	**Values**	
Datasets	UN, CO, AC, CL HOUSE	
Data cardinality	1K, **5K**, 10K	(Diversity quality)
	5K, **10K**, 30K	(Scalability Analysis)
	100K, 200K, 500K	(Sensitivity Analysis)
	17265, 127930	(Real datasets)
Weight cardinality	1K, **5K**, 10K	(Diversity quality)
	5K, **10K**, 30K	(Scalability analysis)
	100K, 200K, 500K	(Sensitivity analysis)
	100K, 200K, 500K	(Real datasets)
# results(r)	3, 4, **5**	(Diversity quality)
	10	(Scalability analysis)
	5, **10**, 30	(Sensitivity analysis)
	5, **10**, 30	(Real datasets)
# top-k results(k)	**10**, 20	(Diversity quality)
	5, **10**, 30, 50	(Scalability analysis)
	5, 10, **30**, 50	(Sensitivity analysis)
	5, 10, **30**, 50	(Real datasets)
# dimensions(d)	3	(Diversity quality)
	3	(Scalability analysis)
	3, 4, 5, 6	(Sensitivity Analysis)

Table 6.4: Parameter values with default values presented in bold

problem (they do not eliminate it) by choosing sets that are selected by distant user preferences.

6.8 Experimental evaluation

In this section, we present the results of the experimental evaluation. All algorithms were implemented in Java and the experiments run on 2x Intel Xeon X5650 Processors (2.66GHz). The algorithms are disk-based and the index structure used was an R-tree with a buffer size of 100 blocks, where the block size is 4KB. The main parameters and values used through the experiments are presented in Table 6.4.

Datasets. For the dataset S, we use both synthetic and real data. We examine four different synthetic data distributions, namely uniform (UN), correlated (CO), anticorrelated (AC) and clustered (CL). For the uniform dataset, the data object values for all d dimensions are generated independently using a uniform distribution. The correlated and anticorrelated datasets are generated as described in [13]. The clustered dataset was created as follows: first 5 cluster centroids were selected randomly. Then, each coordinate is generated on the d-dimensional space by following a normal distribution on each axis with variance $\sigma_S^2 = 0.345$, and a mean equal to the corresponding coordinate of the centroid. In addition, we use the dataset HOUSE (Household), which is described in Section 4.7. For the dataset W of the weighting vectors, we used a uniform (UN) distribution.

Algorithms. We implemented the three algorithms for centroid computation (*Rtopk*, *Itopk*, and *Atopk*) coupled with the *DPSA* algorithm as described in Section 6.4, and the selective top-k algorithm (*Stopk*) described in Section 6.5. We also implemented an exact algorithm (denoted *opt*) that finds the optimal solution, but obviously cannot scale well. For reverse top-k processing, *Rtopk* uses the state-of-the-art BBR* algorithm [113], while *Itopk* uses the BB algorithm [112]. In all algorithms, the dataset is indexed by an R-tree and for the top-k query processing we employ a state-of-the-art branch-and-bound algorithm [102].

Metrics. The metrics under which we evaluated the implemented algorithms were: (a) execution time required by each algorithm, (b) I/O accesses, (c) achieved diversity values. We also measured the number of processed top-k queries, but in the interest of space we do not report them since they follow exactly the I/O metric. We measure only the I/O induced on the dataset S, since the I/O on W are caused by sequential access and accessing data sequentially is much faster than the random accesses of S.

We conduct an experimental study varying the parameters of dimensionality (3-6), cardinality (1K-500K) of S, cardinality (1K-500K) of W, value of k (5-50), value of r (3-30), sample size $|W'|$ ($0.001|W|$-$0.1|W|$), and number of *steps* (100-1000). Each experiment was repeated 10 times over different instances of the datasets with the same parameters and different seed to the random generator, in order to factor out the effect of randomization. Average values are reported in all cases.

Evaluation methodology. Our evaluation was divided in three parts. In the first part (6.8.1), we compare the algorithms *Atopk* and *Stopk* against the exact algorithm (*opt*) in order to evaluate the quality of approximation of diversity. In the second part (6.8.2), we evaluate the performance of *Stopk* against the algorithms that rely on centroid computation (*Rtopk*, *Itopk*, and *Atopk*). In the last part (6.8.3 and 6.8.4), we perform a thorough sensitivity analysis of

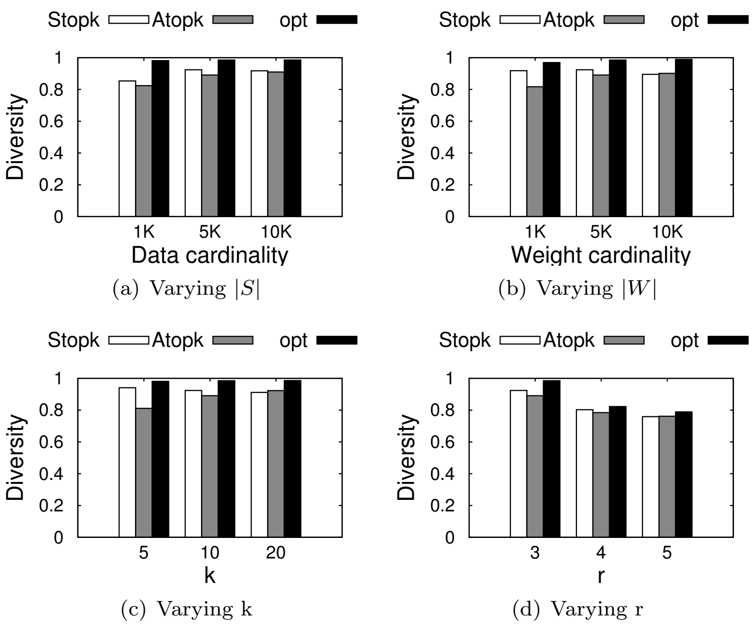
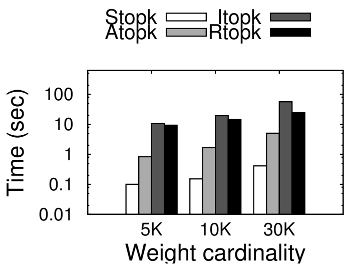


Figure 6.2: Comparison to optimal diversity value

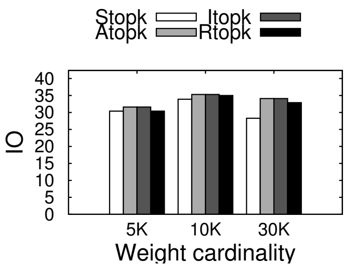
Atopk (which proved to perform best among the three algorithms with centroid computation) against *Stopk*. We should stress here that the diversity of the result set of *Stopk* is calculated using the whole set of preferences W, and not only the vectors used for the identification of the most diverse products.

6.8.1 Quality of diversity

The purpose of this series of experiments is to study the loss of diversity compared to optimal solution when using our algorithms *Atopk* and *Stopk*. Thus, we compare to the optimal diversity produced by an exact algorithm (*opt*), which examines all possible $\binom{|S|}{r}$ combinations of products exhaustively to find the optimal solution. Recall that *Rtopk* and *Itopk* produce exactly the same result set as *Atopk* and therefore have also the same diversity. The default setup for this series of experiments was: d=3, $|S| = 5K$, $|W| = 5K$, $k = 10$, $r = 5$, $s = 0.1 \cdot |W|$, *steps* = 100, S:UN, and W:UN.



(a) Time



(b) I/O

Figure 6.3: Performance of all algorithms when varying $|W|$

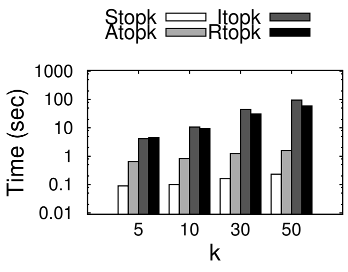
Figure 6.2 depicts the diversity values for varying different parameters, namely $|S|$, $|W|$, r and k. The diversity achieved by the greedy algorithm (*Atopk*) is in most cases quite close to optimal, while *Stopk* results in similar diversity values. As we can observe, our approximate algorithms perform very well in comparison with the exact algorithm. In the worst case, when the size of the objects dataset is only $|S|$=1000 objects(Figure 6.2(a)), the maximum difference in diversity is 19%. As the datasets grow in size, the diversity of the result set of the approximate algorithms approaches the optimal diversity. It is noteworthy that as the number of returned objects (r) increases, the diversity value drops. This behavior is expected, as the more points we select the smaller their average distance will become.

We omit the figures comparing the performance of our algorithms to *opt*, since, as expected, our algorithms outperformed the exact approach by 1-4 orders of magnitude in terms of execution time.

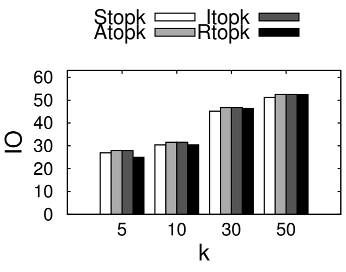
6.8.2 Scalability analysis

In this series of experiments we compare the performance of the algorithms with centroid computation (described in Section 6.4) in terms of execution time and I/O. We also include the *Stopk* algorithm in the charts for completeness. The default setup for this series of experiments is d=3, $|S| = 10K$, $|W| = 10K$, $k = 10$, $r = 10$, S:UN, W:UN, $steps = 100$, $s = 0.01 \cdot |W|$.

Varying |W|. As Figure 6.3 illustrates, *Atopk* and *Stopk* outperform by orders of magnitude the *Rtopk* and *Itopk* algorithms in terms of execution time. This



(a) Time

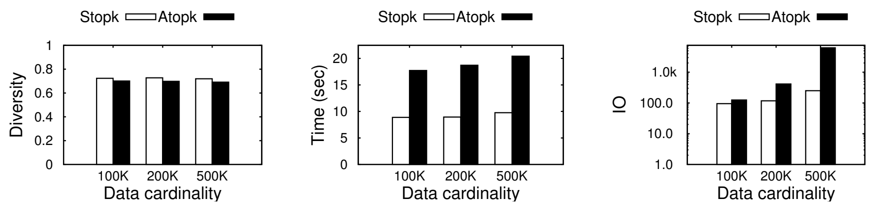


(b) I/O

Figure 6.4: Performance of all algorithms when varying k

difference is not reflected in the measured I/O, because of the use of the buffer of the R-tree. When the number of issued top-k queries is considered, both *Rtopk* and *Itopk* process at least one order of magnitude more top-k queries than *Atopk* and *Stopk*. This processing cost is responsible for their slow runtime. We note that even though both *Rtopk* and *Itopk* are more efficient than *Atopk* when a single reverse top-k query or a small number of influential points is needed, they are less efficient when they are run repeatedly multiple times. In this case, *Atopk* has a benefit and performs better. In particular, *Rtopk* has no memory of the completed executions for different queries, and therefore it computes repeatedly the top-k results of many preference vectors. On the other hand, *Itopk* performs fewer reverse top-k queries than *Rtopk*, but shares the same shortcoming for those reverse top-k queries that it processes. Therefore, it faces the same problem as the *Rtopk*, however in not such great extent. Similar conclusions are drawn when varying the data cardinality $|S|$. The performance of the algorithms is much less affected by the increase of data cardinality, as during the execution of the top-k queries very few data objects are accessed.

Varying k. Figure 6.4 illustrates the effect of varying parameter k. *Atopk* consistently outperforms *Rtopk* and *Itopk* in terms of time, while *Stopk* improves further the performance in terms of both time and I/O. *Atopk*, *Rtopk* and *Itopk* have the same performance in terms of I/O due to the R-tree buffer employed during query processing. Furthermore, for all algorithms, both time and I/O increase for increasing values of k.



(a) Diversity (b) Time (c) I/O

Figure 6.5: Varying $|S|$

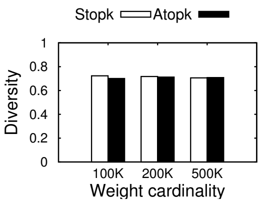
Since *Atopk* consistently outperforms the two other algorithms (*Rtopk* and *Itopk*) that rely on centroid computation, we use only *Atopk* in the remaining experiments as representative of this family of algorithms.

6.8.3 Sensitivity analysis

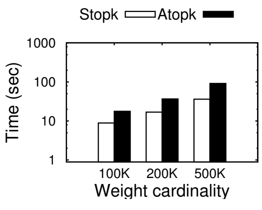
In this section, we provide a detailed sensitivity analysis by varying different parameters that influence the performance of our proposed algorithms. The default setup for this series of experiments is $d = 3$, $|S| = 100K$, $|W| = 100K$, $k = 30$, $r = 10$, S:UN, W:UN, $steps = 500$, $s = 0.01 \cdot |W|$.

Varying |S|. Figure 6.5 illustrates the performance of *Atopk* and *Stopk* for increasing cardinality of the dataset S. We observe that *Stopk* retrieves a set of r objects with similar diversity compared to the set retrieved with *Atopk*. *Stopk* performs constantly better than *Atopk* with respect to time and I/O. In particular, the processing time of *Stopk* is minimally affected by the size of S while the processing time of *Atopk* increases linearly. Regarding I/O, there is a slight deterioration in the performance of *Stopk* due to the fact that the increased size of the dataset causes more misses when reading from the buffer of the R-tree. The same holds for *Atopk*, but in this case the phenomenon is enhanced by the large number of top-k queries performed.

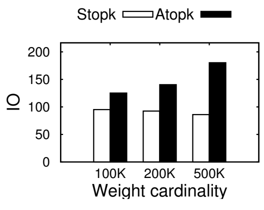
Varying |W|. Figure 6.6 studies the behavior of *Atopk* and *Stopk* with respect to the cardinality of the dataset W. We observe that *Stopk* achieves similar values of diversity compared to *Atopk* while it constantly performs better with respect to both time and I/O. Figure 6.6(b) indicates that affects the processing



(a) Diversity

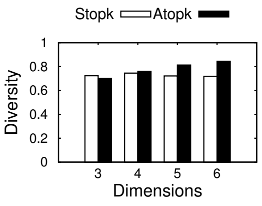


(b) Time

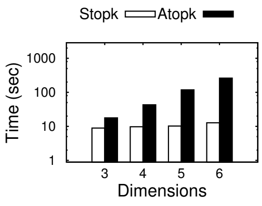


(c) I/O

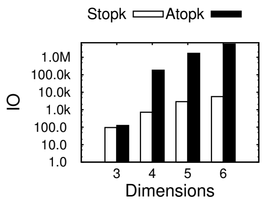
Figure 6.6: Varying $|W|$



(a) Diversity



(b) Time

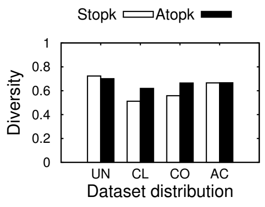


(c) I/O

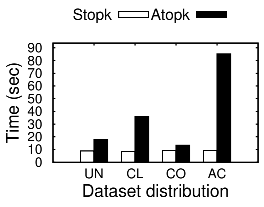
Figure 6.7: Varying d

time of both *Stopk* and *Atopk*. *Atopk* needs to perform more top-k queries while *Stopk* is affected as it needs to search longer for diverse preferences in W. Regarding I/O, *Stopk* is minimally affected as the number of top-k queries executed are not affected by the size of $|W|$. On the contrary the I/O accesses of *Atopk* increase linearly with respect to $|W|$ as more top-k queries need to be executed.

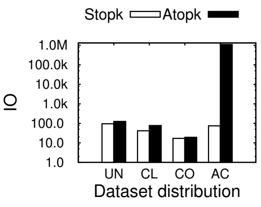
Varying d. For increased dimensionality, as depicted in Figure 6.7(a), the diversity value of *Stopk* compared to *Atopk* is influenced more than for the other parameters. Recall that the diversity between the products was calculated for *Stopk* using all vectors in W and not only the ones used for the identification of the products. However, the processing time of *Stopk* remains almost unaffected while it increases significantly for *Atopk*. Regarding I/O, the performance of



(a) Diversity

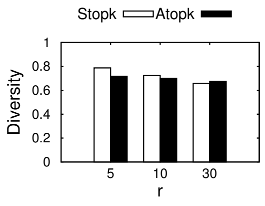


(b) Time

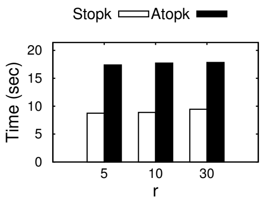


(c) I/O

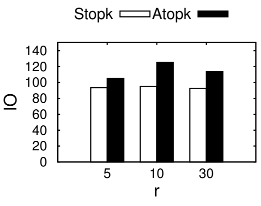
Figure 6.8: Varying data distribution



(a) Diversity



(b) Time

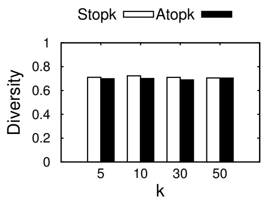


(c) I/O

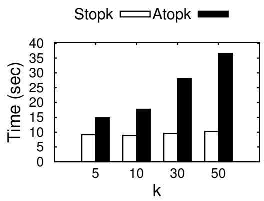
Figure 6.9: Varying r

Stopk deteriorates slower than in the case of *Atopk*. Note that both Figures 6.7(b) and 6.7(c) are in log scale in the y-axis.

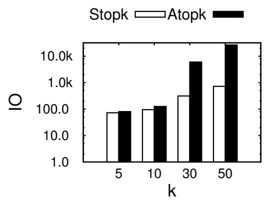
Varying dataset distribution. The efficiency of the proposed algorithms was evaluated against 4 different distributions (UN,CL,CO,AC). In all cases, *Stopk* achieved diversity values similar to the ones achieved by *Atopk*. The case of anticorrelated data is of particular interest, as *Stopk* performs an order of magnitude better with respect to time and several orders of magnitude with respect to I/O. We should note that in most cases the diversity achieved by *Stopk* is almost equal to the one achieved by *Atopk* and in some cases it is even slightly higher. This happens because *Stopk* locates the most diverse preferences and based on them it identifies the most diverse products. *Atopk* on the other hand, bases the search for most diverse objects on the centroids of the $RTOP_k$ sets of the products.



(a) Diversity

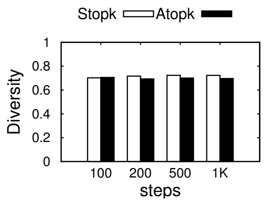


(b) Time

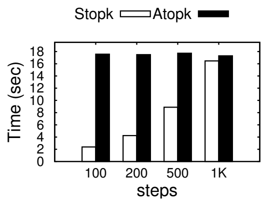


(c) I/O

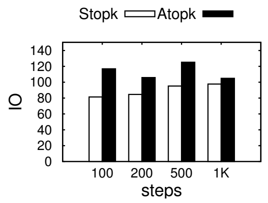
Figure 6.10: Varying k



(a) Diversity



(b) Time

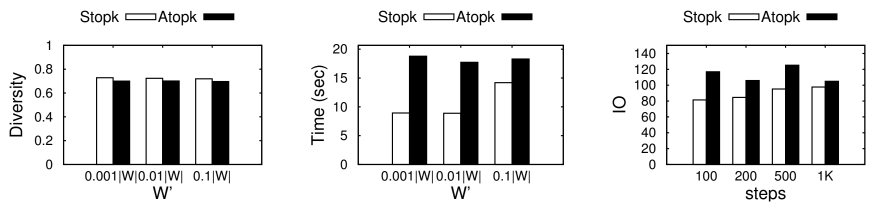


(c) I/O

Figure 6.11: Varying *steps*

Varying r. Figure 6.9 examines the effect of varying the number r of retrieved products on our algorithms. First, we observe in Figure 6.9(a) a decreasing tendency of the diversity value as r increases for both algorithms, which is expected as also the diversity value of the optimal solution will decrease as the most diverse products are selected first. As far as the performance is considered, in Figure 6.9(b), the time of *Atopk* is not influenced by the increase of r, because *Atopk* computes all top-k queries independently of the size of the result set r and the computational cost of *DPSA* algorithm is not significant compared to the cost of the top-k queries. *Stopk* is also not significantly affected, since the values of r are relatively small, and the algorithm is executed *steps* times in any case. Still, *Stopk* remains always much faster than *Atopk*.

Varying k. In Figure 6.10, we gradually increase the parameter k of the reverse top-k queries from 5 to 50. In Figure 6.10(a), we notice that the diversity value is stable as k increases, which seems counter-intuitive at first. By increasing k, the size of the reverse top-k set increases for some objects and more



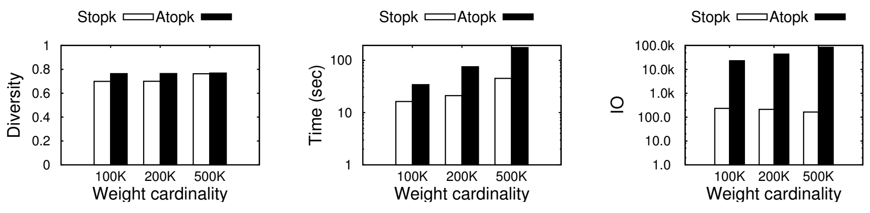
(a) Diversity (b) Time (c) I/O

Figure 6.12: Varying W'

objects have a non-empty reverse top-k set. However, this does not influence the diversity value significantly, as the most diverse centroids may not change. Figure 6.10(b) depicts the time obtained for different values of k. Although we witness a small deterioration in the performance of both algorithms, *Stopk* consistently outperforms *Atopk*. Processing top-k queries is more time-consuming for higher values of k and the *DPSA* algorithm gets slower with increasing k because the number of candidates for finding the diverse objects increases. We should add however, that the effect of parameter k has much smaller impact on the performance of *Stopk* because *Stopk* performs a small number of top-k queries.

Varying steps. The *steps* parameter is an essential parameter for the *Stopk* algorithm as it balances the efficiency of the algorithm and the diversity the algorithm achieves. Recall that *Stopk* performs only *steps* top-k queries, which is only a small fraction of the $|W|$ top-k queries that *Atopk* performs. On the other hand, *Stopk* executes also *steps* times *DPSA* algorithm on a small set of approximate centroids, which is not necessary for *Atopk*. In Figure 6.11, we observe that the diversity achieved using very few vectors is quite close to the diversity achieved by the *Atopk* algorithm. As we increase the *steps* parameter, the achieved diversity increases marginally. However, the execution time increases proportionally with the increase of the *steps* parameter. This experiment verifies that a small value of *steps* is sufficient to produce results of high diversity in a very efficient way.

Varying sample size $|\mathbf{W'}|$. The size of sample of preferences from which we select the two initial centroids plays an important role in the performance of the *Stopk* algorithm. The complexity of the selection process is $O(|W'|^2)$ and



(a) Diversity (b) Time (c) I/O

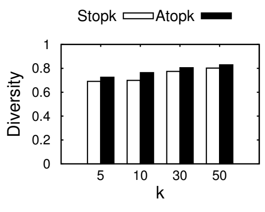
Figure 6.13: HOUSE dataset: varying $|W|$

therefore a large sample can have significant impact on the performance of the algorithm. However, as shown in Figure 6.12, the increased cost in performance is not accompanied by an increased gain in diversity. The reason behind this fact is that once the sample is large enough to offer a good representation of the whole set of preferences, further enlargement will not help significantly in finding better initial centroids.

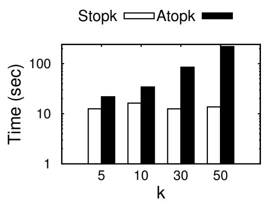
6.8.4 Results on real data

We have also performed an evaluation of our algorithm using a real dataset. The conclusions drawn are overall in accordance with the conclusions made by the evaluation with synthetic data, thus verifying our findings. The default setup for this series of experiments is $|S| = 127930$, $|W| = 100K$, $k = 30$, $r = 10$, W:UN, $steps = 500$, $s = 0.01 \cdot |W|$. The size of the dataset used and the high complexity of the exact algorithm (opt) did not allow the exact algorithm to terminate and therefore we did not include its performance results in this series of experiments.

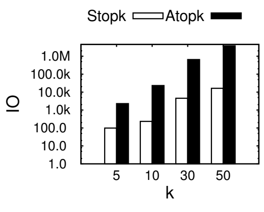
Analysis for varying |W|, k, and r. Figures 6.13-6.15 depict performance of the two algorithms. For all values of the varying parameters *Stopk* achieves diversity values close to the ones of *Atopk*. For both algorithms, we notice a drop in the diversity values when r is increases which is expected as analyzed in 6.8.1. With respect to processing time, it is evident that both parameters $|W|$ and k play a significant role in the performance of *Atopk*. This does not come as a surprise as the processing cost of *Atopk* is dominated by the processing cost of the top-k queries needed for the computation of the centroids of the $RTOP_k$ sets for each product. On the contrary, *Stopk* is much less affected by those



(a) Diversity

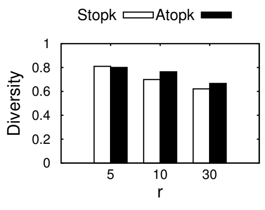


(b) Time

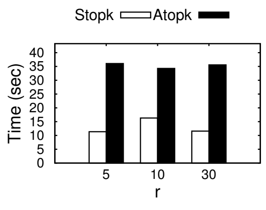


(c) I/O

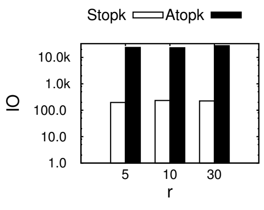
Figure 6.14: HOUSE dataset: varying k



(a) Diversity



(b) Time



(c) I/O

Figure 6.15: HOUSE dataset: varying r

parameters as it performs a limited number of top-k queries. The increase of parameter r has little effect in both algorithms. The performance difference with respect to I/O is in all cases larger than two orders of magnitude. Only exception is for $k < 10$ where *Stopk* is one order of magnitude more efficient.

6.9 Conclusion

In this chapter, we address the problem of selecting the r most diverse products based on customers' preferences. The reverse top-k set of each product is represented by its centroid and the distance between centroids is then expressed using cosine distance. In order to find products that are attractive to customers with dissimilar preferences, we define the r-Diversity problem as a *dispersion* problem applied on the products' reverse top-k sets. As dispersion problems are known to be NP-hard, we propose two approximate algorithms that solve the problem. The first algorithm computes the reverse top-k sets and then applies a greedy algorithm that retrieves a set of products of high diversity. The second applies the greedy algorithm on an approximation of the reverse top-k sets by evaluating only some carefully selected top-k queries. In our experimental evaluation, we study the performance of the proposed algorithms and the diversity of the retrieved products in various experimental setups. In particular, we demonstrate that our algorithms both achieve diversity values close to optimal and are very efficient in practice.

Chapter 7

Maximizing Influence of Spatio-Textual Objects

In the previous chapters, we studied the problem of analyzing user preference queries in order to identify objects that can attract the interest of users and motivate them to explore the database content of a service provider or a product manufacturer. Nowadays however, there is an increasing number of platforms that offer the users the ability to explore the products or services of a large number of providers by merging in unified catalogs the offered products or services. In such cases, it not uncommon for user preferences to include a spatial location associated with the desired product or service. For example, given a database of hotels annotated with features (in the form of keywords), tourists can pose *spatio-textual* queries, which combine spatial distance and textual relevance and retrieve a set of hotels ranked according to their distance from a user specified location and textual similarity to the query keywords. In this context, a challenging problem is to select a bounded set of at most b keywords to describe the facilities of a spatial object, in order to make the object appear in the top-k results of as many users as possible. In this chapter, we study this problem, called *Bests-terms* and we show that it is NP-hard. Hence, we present a greedy approach and a graph based algorithm for keyword selection. By means of a thorough experimental evaluation using real data, we compare the two algorithms and we demonstrate the efficiency of the graph based approach.

7.1 Introduction

Spatio-textual search has attracted increased attention recently, due to the numerous applications that provide value-added services to the users by combining spatial location with textual relevance. Given a database of geographical points of interest that are annotated with textual information (also called *spatio-textual objects*), the objective of a spatio-textual query is to retrieve a ranked set of top-k spatio-textual objects that are close to the query point and have high textual similarity to the query keywords. As a notable example, consider hotels that are annotated with their facilities (e.g., in the form of keywords) and tourists that search for hotels close to some location of interest and a set of query keywords indicating desired facilities (for example "pool" or "Wi-Fi").

An interesting problem encountered in real-life applications that rely on spatio-textual retrieval is how to improve the ranking of a spatio-textual object for as many users as possible. For instance, for a newly established hotel at some location, the question is how to enrich its textual annotation in order to maximize its rank for many different users. To address this challenging problem, we capitalize on reverse top-k queries [110], which retrieve the set of users that have a given object in their top-k results. We model the problem as a maximization problem of the cardinality of the reverse top-k result set, and we explore the different combinations of keywords that will increase the query object's rank for many users, when added to its textual annotation. We call this problem *Best terms* problem, we show that it is NP-hard, and we present a greedy solution that serves as baseline. Then, we propose a novel algorithm that boosts the performance of query processing, by deliberately selecting keywords that increase the score of the query object for many users simultaneously. Finally, we present the results of our experimental evaluation that verifies the performance gains of our algorithm.

In summary, our main contributions are outlined below:

- We formulate the novel problem, called *Best terms*, of increasing the rank of a spatio-textual object for many different users, by enriching its textual description.
- We show that the *Best terms* problem is NP-hard and we provide a baseline solution.
- We propose an efficient query processing algorithm that significantly outperforms the baseline consistently.

- We provide an experimental evaluation that demonstrates the merits of our approach.

The rest of this chapter is structured as follows: Section 7.2 provides an overview of the related work. Section 7.3 presents the necessary background and preliminary concepts. Then, in Section 7.4, we formally describe the problem statement. Section 7.5 presents the baseline algorithm, while Section 7.6 describes our efficient query processing algorithm. Section 7.7 shows the experimental evaluation, and Section 7.8 presents the conclusions.

7.2 Related work

In this section, we provide an overview of the related research literature.

Keyword recommendation. Zhang et al. [128] present a method for recommending keywords for advertisements in keyword search results using Wikipedia. They focus mostly in cases where the advertisement (target) consists of short-text web pages that contain inadequate textual content to describe the advertised entity. Based on the fact that a large number of entities are described in Wikipedia, they use Wikipedia articles relevant to the advertised entity in order to recommend keywords to connect to the target. Fuxman et al. [38] follow a different approach. They suggest keyword queries to advertisers using logs that store the queries posed by the users and the URLs of the result set that were selected by the users. Some of the URLs are also connected to a set of concepts. The target of the authors is to connect the set of concepts to the queries using the Markov Random Field model and suggest the most relevant queries for each concept to the advertisers. Ravi et al. [90] propose variety of methods for automatic generation of bid phrases. Among others, they introduce the usage of a translation model that extends a predefined mapping between bidding phrased and target web pages. Papadimitriou et al. [84] study the problem of mapping an advertisement in a set of URLs based on keyword queries. In particular, they assume that each advertisement is mapped to a set of keyword queries and their aim is to map each advertisement in a set of URLs that will be representative of the results produced by the attached keyword queries. Choi et al. [21] create a representative summary of the advertisement based on the context of the advertised material. Their method is making use of co-occurrence and semantic vectors in order to enrich the ad context and create a representative set of terms. Cholette et al. [22] study the problem of finding optimal bids in search based algorithms. Agrawal et al. [2] introduce an approach for recommending bid phrases from a given ad landing page by classifying a set of labels generated by click logs. Their classifier has logarithmic complexity and can efficiently make predictions on large sets of labels.

The aim of the aforementioned approaches is to identify potentially relevant queries to the advertised products and form bid phrases based on the identified queries. Our approach is inherently different, because the above techniques try to predict relevant queries and do not consider the relevance of the advertised product in relation to similar products. In addition, they do not consider top-k search criteria as the appearance of a product in a search result is decided mainly on the bidding strategy. On the contrary, our aim is to enhance the description of a spatio-textual object and to increase the number of queries for which the target product appears in the top-k list of the search results. In this effort, we take into consideration not only the user preferences, but also the rest of the spatio-textual objects that are relevant to those queries.

Spatial keyword search. Spatial keyword search has been well studied during the recent years and several index structures have been introduced for efficient search. A detailed evaluation of existing spatio-textual indexes can be found in [19]. Felipe et al. [36] introduced the IR^2-tree index, which integrates a bitmap signature on each node of an R-tree describing the textual content of the subtree rooted at the node. Cong et al. [24] introduced the IR-tree and its variants. The IR-tree is based on the R-tree structure as well. Each node of the tree is associated with an inverted index containing the textual information of the children of the node. Rocha et al. [91] proposed the S2I index, which uses different strategies for frequent and infrequent terms. The spatial distribution of a frequent term is stored in an aggregated R-tree (aR-tree), where each node contains an aggregated value of the impact of the term on the objects contained in the subtree rooted at the node. Cao et al. [15] introduced the concept of *prestige*, where a spatio-textual object has a higher prestige if it is collocated with other textually similar objects. They calculate the prestige of a spatio-textual object based on a graph, where each node corresponds to an object and two nodes are connected if and only if their textual similarity and spatial proximity exceed certain thresholds. Deng et al. [26] suggested an approach of finding a set of spatio-textual objects that are relevant to a spatio-textual query and at the same time they fulfill a desired spatial property. In particular, their aim is to identify a keyword-cover of optimal score, where a keyword cover is defined a set of objects where each object is associated with exactly one term of the spatio-textual query.

Ying Lu et al. [76] and Jiaheng Lu et al. [74] studied the problem of reverse spatial and textual k nearest neighbor search, where, given a query point q, the objective is to locate the set of spatio-textual objects, for which q is among the k nearest neighbors. The distance between the objects is a linear combination of the textual and the euclidean distance of the objects. The authors introduce the IUR-tree, which is an adaptation of the IR-tree. Each node of the IUR-tree contains the union and the intersection of the terms contained in the objects in the subtree rooted at the node. Our approach is different, as we do not evaluate the similarity between elements of a set of spatio-textual objects, but our aim to increase the relevance and therefore the visibility of an object against a set of user preferences, which constitutes a different set from that of the spatio-textual objects that our query object belongs.

Wu et al. [117] proposed the W-IR-tree, which is similar to the IR-tree but it differs in the way it is constructed. While the IR-tree places the objects in leaf nodes based on their distance, the W-IR-tree partitions the objects based primarily on their textual relevance. The W-IR-tree shows improved performance for batch queries, where objects are considered relevant to the query only if they contain all terms of the query. The W-IR-tree cannot be applied in our case, as we consider it possible for a spatio-textual object to be relevant to a user query even if it does not contain all terms of the query.

7.3 Preliminaries

Let D be a set of *spatio-textual* objects, where each object o is represented by a tuple of the form $o = \langle o.T, o.L \rangle$, where $o.T$ is a set of keywords describing the features of o and L is a point in $\mathbb{R}^2$ describing the location of o. We denote as $\mathcal{A} = \bigcup_{o \in D} o.T$ to be the set of all keywords in D. For a given object o, we consider the *size* of o to be equal to $|o.T|$, namely the size of an object is the number of terms it contains.

7.3.1 Top-k spatial keyword queries

Let u be a user preference query on D, where u is represented by the a tuple $u = \langle u.T, u.L, \alpha \rangle$, $u.T \subseteq \mathcal{A}$ is the text describing the user's desired features, $u.L \in \mathbb{R}^2$ denotes the desired location, and $\alpha \in [0, 1]$ denotes the importance of location over matching the desired features. Given a preference u, we can assign a score to each object using the following equation:

$$f(o, u) = \alpha \times \delta(o.L, u.L) + (1 - \alpha) \times \theta(o.T, u.T) \tag{7.1}$$

where $\delta(o.L, u.L)$ is the spatial distance, and $\theta(o.T, u.T)$ is the textual distance between the object o and the user preference u. Given an integer k, we can return the top-k spatio-textual objects according to their score. We assume that lower scores are better, both spatial and textual distances are normalized in the interval $[0, 1]$ and $f(o, u) = 1.0$ if $\theta(o.T, u.T) = 1$. The latter assumption implies that objects that are not textually relevant to the query cannot be considered as a valid result.

The textual relevance we employ is the normalized intersection of terms between the description of a spatio-textual object $o.T$ and a user preference keyword set $u.T$, i.e., $\theta(o.T, u.T) = 1 - |o.T \bigcap u.T||u.T|^{-1}$. Although in large documents different textual similarity functions are more appropriate, the intersection is more representative in cases of feature selection. For instance if a user is looking for a hotel with a restaurant and a pool, any hotel offering more features (e.g. restaurant, pool, bar) than the ones specified by the user should not be less textually relevant than a hotel that offers only the features specified by the user preference (restaurant, pool).

The cardinality of the $RTOP_k$ set of a query-object q is called *influence score* of the object and we denote it as $f_k^I(q)$. The influence score indicates the number of users to whom q is visible.

7.3.2 IR-tree

We employ a state-of-the-art index structure to process spatial keyword queries, namely the IR-tree [24]. The IR-tree is an R-tree, where each node is associated with an inverted index of the objects contained in the respective sub-tree rooted at the node. In more detail, each leaf node contains an inverted index of the spatio-textual objects contained in the node. The leaf node is characterized by a spatio-textual pseudo-object that consists of a *minimum bounding rectangle* (MBR) enclosing all objects of the node and a pseudo-document consisting of the union of all the terms contained in the children of the node. Each non-leaf node contains an inverted index of the spatio-textual pseudo-objects of the children nodes it contains. Non-leaf nodes are also characterized by spatio-textual pseudo-objects, which are constructed similarly to the pseudo-objects of the leaf nodes.

7.4 Problem definition

Let D be a set of spatio-textual objects and U be a set of preferences. We assume that the location of the spatio-textual objects cannot change. Therefore, the problem of maximizing the influence score of a spatio-textual object q can be viewed as the problem of selecting a set of terms that maximize the influence score of q by increasing the textual similarity between q and the user preferences in U. Ideally, according to the textual relevance function described in Section 7.3 the textual description of q should be enhanced with the addition of the entire vocabulary of the user queries. This is in most cases practically impossible, therefore, we study the problem of finding a set of b terms, which when added to the textual description of q, they maximize the influence score of q. We refer to this problem as *Best-terms query.*

Definition 7.1. ***Best-terms query.*** *Given a set D of spatio-textual objects, a set of terms $\mathcal{A} = \bigcup_{o \in D} o.T$, a set of queries U, a scoring function f, an integer k, a spatio-textual object $q = \langle q.T, q.L \rangle$, and an integer b, the set* BT *is a set of terms such that* $\mathrm{BT} \subseteq \mathcal{A}$, $\mathrm{BT} \bigcap q.T = \emptyset$, $|\mathrm{BT}| \leq b$ *and* $\forall T \subseteq \mathcal{A}, |T| \leq b$ *it holds that* $f_k^I(q_1) \geq f_k^I(q_2)$ *where* $q_1 = \langle q.T \bigcup \mathrm{BT}, q.L \rangle$ *and* $q_2 = \langle q.T \bigcup T, q.L \rangle$.

The Best-terms problem is NP-hard. We show that by studying a special case of a Best-terms query, namely the respective decision problem of finding whether there exists a set of terms T with $|T| \leq b$ such that $f_k^I(\langle q.T \bigcup T, q.L \rangle) = |U|$.

Definition 7.2. ***Best-terms (decision problem).*** *Given a set D of spatio-textual objects, a set of terms $\mathcal{A} = \bigcup_{o \in D} o.T$, a set of queries U, a scoring function f, an integer k, and a spatio-textual object $q = \langle q.T, q.L \rangle \in D$, decide if there is a set* BT *such that* $\mathrm{BT} \subseteq \mathcal{A}$, $\mathrm{BT} \bigcap q.T = \emptyset$, $|\mathrm{BT}| \leq b$ *for which it holds that* $f_k^I(q_1) = U$ *where* $q_1 = \langle q.T \bigcup \mathrm{BT}, q.L \rangle$

We will show that the decision problem of Definition 7.2 is NP-complete by reducing the set cover problem to the decision problem using the restriction technique [39].

Definition 7.3. ***Set cover problem.*** *Let U be a set of elements (universe) and $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a collection of sets where $\bigcup_{i=1}^n T_i = U$. The* set cover problem *decides if there is a subset of $\mathcal{T}$, $\mathcal{T}' \subseteq \mathcal{T}$ of size $|\mathcal{T}| \leq b$ such that $\mathcal{T}'$ is a cover of U.*

Theorem 7.1. *The decision problem of Best-terms is NP-complete.*

Proof. Let an oracle machine select the BT set for a query object q. We set $p = \langle q.T \bigcup \mathrm{BT}, q.L\rangle$ and by performing a TOP_k query for each user preference we can calculate the $RTOP_k(p)$ set and the influence score $f_k^I(p)$ of object p in polynomial time. Therefore the solution can be verified in polynomial time and our problem belongs to the NP class.

We set U a to be a set of users and $D = \{q\}$, where $q.T = \emptyset$. We define a collection $\mathcal{T} = \{T_1, \ldots, T_{|A|}\}$ of sets, one for each term t_i in A where a user u belongs in T_i only if $t_i \in u.T$. If we consider $k = 1$, then, for all users that $q.T \bigcap u.T = \emptyset$ it holds that $q \notin TOP_k(u)$ since q is not relevant to $u.T$. If $q.T \bigcap u.T \neq \emptyset$ then $q \in TOP_k(u)$ as it is the only object. Therefore any selection of a term t_i is equivalent of selecting a subset of T_i of U. The set cover problem is consequently reduced to Problem 7.2, as it can be seen as a special case of Problem 7.2. Problem 7.2 is therefore NP-complete. Best-terms is at least as hard as Problem 7.2, which leads us to the conclusion that the Best-terms problem is NP-hard. ■

7.5 The best term first (BTF) algorithm

Since the Best-terms problem is NP-hard, an exact solution is infeasible, even for medium-sized datasets. Motivated by this observation, in this section we describe a greedy algorithm, termed *Best Term First* (BTF), that provides an approximate solution to the Best-terms problem. BTF operates in an iterative way consisting of b steps, and on each step it adds to the query object the term that induces the highest increase in influence score.

7.5.1 Algorithm description

Algorithm 15 describes the BTF approach in more detail. BTF takes as input an IR-tree index containing the set of spatio-textual objects D, and an IR-tree index containing the set of user preferences U. BTF works in b iterations, and in each iteration the best term (i.e., the term that induces the maximum increase in the influence of q) is selected and added to the terms of the query object.

Initially, BTF creates a pseudo-preference q' defined by q and using $\alpha = 1$, which indicates that q' uses only distance to data objects, not textual similarity, for ranking. The role of q' is to enable traversing the preference dataset solely based on distance to the query object q. This imitates a sorted access to the preferences, yet this is achieved by means of the IR-tree index on U, without having to sort U.

Algorithm 15 Best Term First (BTF) Algorithm

```
Input: $U$:set of users
       $D$: set of objects
       $q$:query point
       $b$ : number of new terms
Output: BT: set of new terms
 1: $C \leftarrow \emptyset$, buffer $\leftarrow \emptyset$
 2: $q' \leftarrow \langle q.T, q.L, 1\rangle$
 3: bestCandidate$\leftarrow q$
 4: for $i = 0; i < b; i++$ do //repeat until b new terms have been found
 5:    for all $t \in \mathcal{A} - q.T$ do
 6:       $C \leftarrow C \bigcup \{\langle \text{bestCandidate}.T \bigcup \{t\}, \text{bestCandidate}.L\rangle\}$
 7:    end for
 8:    $u \leftarrow$next(U,$q'$)
 9:    while $u \neq null$ do
10:       $\tau \leftarrow \max\limits_{p \in \text{buffer}}(f(p,u))$ //in the first iteration buffer is empty, so we set $\tau \leftarrow \infty$
11:       if $\exists c \in C : f(c,u) \leq \tau$ then
12:          buffer$\leftarrow$ TOP$_k(u)$
13:          $\tau \leftarrow \max\limits_{p \in \text{buffer}}(f(p,u))$
14:          for all $c \in C$ do
15:             if $f(c,u) \leq \tau$ then
16:                $I(c) \leftarrow I(c) + 1$
17:             end if
18:          end for
19:       end if
20:       $u \leftarrow$next(U,$q'$)
21:    end while
22:    bestCandidate$\leftarrow \underset{c}{\text{argmax}}(I(c))$
23: end for
24: BT$\leftarrow$ bestCandidate.T-$q$.T
25: return BT
```

In each iteration, BTF first creates a set C of candidate spatio-textual objects, one for each term that can be added to q. The size of C is equal to $|\mathcal{A} - q.T|$. In lines 10,11 the algorithm exploits the sorted access to the preference dataset, in order to avoid processing some top-k queries. More accurately, given the current user preference u, the score of the last retrieved spatio-textual objects is compared with the scores of the candidate objects C, and if no candidate object has a better score than the k-th ranked spatio-textual object, the user preference is ignored (*pruning condition*) as no candidate object can be in its TOP_k set. Otherwise, the top-k query needs to be executed and its TOP_k result set is stored in the buffer. All candidate objects that are no worse than the k-best element of the calculated TOP_k set belong also to the TOP_k set of u and therefore their influence score is increased. When all user preferences have been examined, the object with the highest influence score is selected and a new set of candidate objects is created based on that object. The procedure is repeated b times until an object with b new terms is created. The b terms that were selected constitute the resulting BT set.

Although BTF adopts a greedy technique to select the b terms, the use of sorted access to dataset U together with the pruning condition reduce the number of processed top-k queries, thereby saving computational costs.

7.5.2 Complexity analysis

The cost of the BTF algorithm is determined by the cost of selection of each of the b terms. As the main factors that affect the cost of term selection are the construction of set C with cost $O(|\mathcal{A}|)$, and the cost C_{topk} of processing a top-k query which in worst case will be processed $|U|$ times. Thus, the overall complexity of BTF is:

$$C_{BTF} = O(b(|\mathcal{A}| + |U|C_{topk}))$$

However, in practice the number of processed top-k queries is much smaller than $|U|$.

7.6 Graph based term selection

BTF extends the textual description of a spatio-textual object iteratively, which forces the algorithm to scan the preferences set U multiple times. In this section we present a novel algorithm, named Graph Based Term Selection (GBTS), which examines the set of preferences only once and creates a graph of terms that provides an estimation of the influence gain any combination of terms may provide.

Essentially, GBTS consists of two separate algorithms. The first algorithm, named Graph Construction (GC), creates a graph connecting the terms that when added to the spatio-textual query object q, they can induce an increase in its influence score. The second algorithm, named Best Subgraph Selection (BSS), traverses the graph in a deliberate manner, in order to identify the sets of terms that will induce the highest increase in the influence score of q.

7.6.1 Graph construction algorithm

Given a set of objects D, a set of user preferences U and a spatio-textual object q, we denote as $\widehat{U}(q)$ the subset of all preferences u ($\widehat{U}(q) \subseteq U$) for which q is does not belong in the $TOP_k(u)$ set and at most b terms are needed for q to be added to $TOP_k(u)$. The Graph Construction algorithm builds a weighted graph $G = (V, E)$, where each node of the graph represents a candidate term, and the weights on edges indicate the maximum increase in the influence score of q that can be induced, if the respective set of terms is added to q.

In more detail, for each examined user preference u, the algorithm adds to graph G a node for each previously unseen term. The edges connecting the nodes and the weights of the edges are determined by the number of terms λ that need to be added to u for it to be included in RTOP$_k(q)$. The value of λ is calculated based on Equation 7.2, where $f_{max}(q, u)$ is the worst score that q is required to have in order to be in the $TOP_k(u)$ set and derives directly from Equation 7.1.

$$f_{max}(q, u) = \alpha \times \delta(q.L, u.L) + (1 - \alpha) \times \frac{|q.T \bigcap u.T| + \lambda}{|u.T|} \tag{7.2}$$

- If λ=1, the algorithm adds a loop edge with weight equal to 1 to each term t that is not contained in q. If the edge already exists, the weight is simply added to the weight of the existing edge.

- If $\lambda > 1$, thus more than one terms are necessary for q to be included to $TOP_k(u)$, the procedure is slightly different. Let $T = u.T - q.T = \{t_1, \ldots, t_n\}$ be the terms that are included in u but not in q. For each pair of terms in $u.T - q.T$, the algorithm adds an edge with weight w_e. As before, if an edge already exists, the weight is added to the existing edge.

Since we add λ terms that correspond to $\frac{\lambda(\lambda-1)}{2}$ pairs of terms, the weight of each edge w_e is set to $2\left(\lambda(\lambda-1)\right)^{-1}$, which is a normalization that makes the sum of weights added equal to 1. Intuitively, we add a total weight of 1 to each subgraph $G' = (V', E')$ where $V' \subseteq T$ and $|V'| = \lambda$, indicating the potential increase in the influence score of q if the terms contained in G' were added to q.

Algorithm 16 Graph Construction (GC) Algorithm

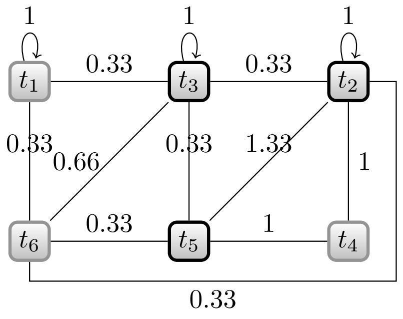
```
Input: $U$:set of users
       $D$: set of objects
       $q$:query point
       $b$ : number of new terms
Output: $G = (V, E)$: resulting graph
 1: $V = \emptyset, E = \emptyset$, buffer$\leftarrow \emptyset$,$G = (V, E)$ //graph initialization
 2: $q' \leftarrow \langle q.T, q.L, 1\rangle$
 3: $u \leftarrow next(U, q')$
 4: while $u \neq null$ do
 5:   buffer$\leftarrow$ TOP$_k(u)$
 6:   $\tau \leftarrow \max_{p \in \text{buffer}} (f(p, u))$
 7:   if $f(q, u) > \tau$ then //if $q \notin TOP_k(u)$
 8:     $T \leftarrow u.T - q.T$
 9:     $V \leftarrow V \bigcup T$
10:     $\lambda \leftarrow \max\left(1, \left\lceil \left(1 - \frac{\tau - a\delta(q,u)}{1-a}\right) |u.T| - |q.T \bigcap u.T| \right\rceil\right)$ //from Eq. 7.2
11:     if $\lambda \leq 1$ then
12:       $E \leftarrow E \bigcup \{e = (t_i, t_i, 1) : t_i \in T\}$
13:     else if $1 < \lambda \leq b$ then
14:       $E \leftarrow E \bigcup \left\{e = \left(t_i, t_j, \frac{2}{\lambda(\lambda-1)}\right) : \forall t_i, t_j \in T \text{ and } t_i \neq t_j\right\}$
15:     end if
16:   end if
17:   $u \leftarrow$next(U,$q'$)
18: end while
19: return $G$
```

Algorithm 16 describes the construction of the term graph G. Similarly to Algorithm 15, GC traverses the preferences based on their distance to q. For each user preference u, if q is not in the $TOP_k(u)$ set, GC updates the node set of G and calculates λ, the number of terms that need to be added in q for it to be included in the $TOP_k(u)$ set (line 10). A non-positive value of λ indicates that u is located near q but $q.T \bigcap u.T = \emptyset$ and therefore q is not included in the $TOP_k(u)$ set. The addition of any term will allow q to be added to TOP$_k(u)$ set and therefore one loop edge is added to each term t for which it holds $t \in u.T - q.T$. If more than one terms are necessary to be added in

User terms

user	**terms**	min terms to be added
u_1	t_1, t_2, t_3	1
u_2	t_2, t_4, t_5	2
u_3	t_2, t_3, t_5, t_6	3
u_4	t_1, t_3, t_6	3

(a) The term-sets for the users



(b) The resulting graph

Figure 7.1: Example Graph: The nodes of the suggested solution are colored with light gray

q ($\lambda > 1$), GC adds all necessary edges in the graph. The algorithm continues until all user preferences have been examined.

The size of the graph depends on the number of distinct terms contained in $\widehat{U}(q)$. The terms correspond to the features extracted from the textual descriptions of spatio-textual objects that describe the offered facilities. In practice, we have noticed that the vocabulary for the targeted applications is limited and therefore the graph is expected to fit in the main memory.

Example 7.1. *As an example, let the user preferences in Figure 7.1(a) be the $\widehat{U}(q)$ set for $b = 3$, i.e., the set of user preferences that can be added to the $RTOP_k(q)$ set if 3 more terms are added to the spatio-textual object q. We also assume that the shown terms for each user preference are not included in q. The first step of the algorithm is the evaluation of the user preference u_1. The algorithm adds to the graph the nodes t_1, t_2, and t_3. Since only one term needs to be added to q for u_1 to be added to $RTOP_k(q)$, it adds one loop edge with*

Algorithm 17 Best Subgraph Selection (BSS) Algorithm

```
Input: $G = (V, E)$: graph
       $b$: number of desired terms
Output: BT:set of new terms
 1: $Q \leftarrow \emptyset$ //Priority Queue
 2: BT $\leftarrow \emptyset$
 3: for $i = 0; i < b; i++$ do
 4:     $t_i \leftarrow$ next node of $G$ with the highest degree
 5:     $G_{t_i} \leftarrow$ expandNode($t_i$)
 6:     Q.add(sumOfWeights($G_{t_i}$),$G_{t_i}$)
 7: end for
 8: while |BT| $\leq b$ do
 9:     $G_S \leftarrow$Q.pop()
10:     add to BT the $b - |\mathrm{BT}|$ highest degree nodes from $G_S$
11: end while
12: return BT
```

weight 1 to all terms. On the next step u_2 is processed and two more nodes (t_4, t_5) are added to the graph. For each pair of terms contained in u_2 we add an edge to the graph with weight equal to $2(\lambda(\lambda-1))^{-1}$, where λ is equal to 2, which is the number of terms needed to be added to q for u to be in $RTOP_k(q)$. When u_3 is processed, t_6 is added to the graph, and for each pair of terms in u_3 an edge with weight $1/3$ is added to the graph. Finally, u_4 is processed and the graph is updated accordingly.

7.6.2 Best subgraph selection algorithm

When the graph has been created, the Best Subgraph Selection algorithm (BSS) chooses as seed nodes the b nodes (terms) of the graph with the highest degree and creates b subgraphs, each containing one node. Next, each subgraph is expanded by adding at each step the highest degree node that is adjacent to a node of the subgraph. The expansion of each subgraph is continued until each subgraph has b nodes or the subgraph cannot be expanded. Finally, the subgraph with the highest sum of edge weights is selected as the solution and the set of terms included in the subgraph are the ones that constitute the BT set.

Algorithm 17 describes the algorithm of the term selection. Initially an empty priority queue (Q) is constructed. Subsequently, at line 4 the algorithm chooses as seed the highest degree node t_i that has not yet been selected and constructs the subgraph G_{t_i} (line 5). The subgraph is constructed by repeatedly selecting the highest degree node adjacent to the G_{t_i} until $|G_{t_i}| = b$ or until no nodes can be added to G_{t_i}. When each subgraph is constructed, it is pushed to Q. The sorting key of Q is the sum of weights of the edges in the subgraph. The BT set is constructed by selecting the subgraph with the highest sum of edges and adding the terms of the subgraph to BT. If the subgraphs contain less than b terms, more subgraphs are pulled from the priority queue until BT contains b terms. In such cases we add from each subsequent subgraph to BT the $b - |\mathrm{BT}|$ highest degree nodes of the subgraph.

Example 7.2. *Continuing the previous example, during the execution of BSS, 3 subgraphs are created with seed nodes the terms t_2, t_5 and t_3. We denote the respective subgraphs as G_{t_i} where t_i is the seed node of the subgraph. Each subgraph G_{t_i} is extended to the highest degree node adjacent to G_{t_i}. In the case of G_{t_2}, the subgraph is expanded by adding first node t_5, which is the node with the highest degree adjacent to t_2 and subsequently with node t_3 which is the highest degree node adjacent to either t_2 or t_5. After the addition of t_3, the size of G_{t_2} becomes equal to 3 and therefore the expansion stops and the next subgraph is processed. In the case of the example all subgraphs produce the same result which includes the light gray nodes in Figure 7.1(b). The nodes contained in the result are the ones to be added to q.*

7.6.3 Complexity analysis

The overall complexity of the Graph Based Term Selection is determined by the two algorithms that comprise it.

$$C_{GBTS} = C_{GC} + C_{BSS}$$

GC consists of two parts: the processing of $|U|$ top-k queries and the addition of edges $\widehat{U}(q)$ times. The addition of an edge is done in constant time and therefore the cost of GC is equal to: $C_{GC} = O(|U| \cdot C_{topk} + \widehat{U}(q))$.

BSS also consists of two parts: the construction of b subgraphs, and the selection of nodes (terms) from the best of these subgraphs. The main cost of BSS is the construction of the b subgraphs, which is $O(b \cdot (b^3 + logb))$. The cost of expanding a single-node subgraph b times and finding the highest degree node is equal to $O(b^3)$, while the cost of insertion to the priority queue is equal to $O(logb)$. The node selection is in worst case $O(b \cdot logb)$, though in practice it is $logb$. Hence, we derive: $C_{BSS} = O(b \cdot (b^3 + logb))$.

Consequently, the overall complexity of Graph Based Term Selection is:

$$C_{GBTS} = O(|U| \cdot C_{topk} + \widehat{U}(q) + b \cdot (b^3 + logb))$$

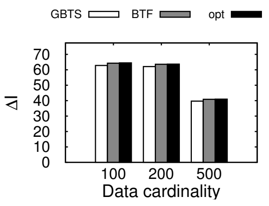
7.7 Experimental evaluation

In this section, we present the results of the experimental evaluation. All algorithms were implemented in Java and the experiments were executed on an AMD Opteron 4130 Processor (2.60GHz), with 32GB of RAM and 2TB of disk.

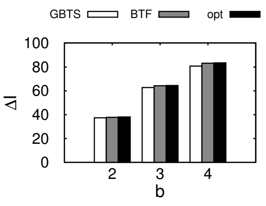
Datasets and metrics. For the dataset D of spatio-textual objects, we used a set of 200000 descriptions of hotels from the site of Booking.com[1]. The dataset contains 188 distinct features. The set of preferences U was generated using a uniform distribution for creating the location and the α parameter of each preference, while the terms were randomly chosen from the vocabulary generated by processing the set of hotels. The location of the user preferences was bounded in the MBR defined by the set of hotels. We also tested our algorithm against a Zipfian distribution of terms. We used the Zipfian distribution generator provided by the Apache Commons project[2]. The metrics under which we evaluated the implemented algorithms were: a) increase in the influence score ΔI, b) number of I/O's performed by each algorithm, and c) processing time.

Experimental procedure. Both datasets D and U were indexed using an IR-tree, where the maximum capacity of each node was 100 entries. We employed a buffer that was fixed at the size of 4MB, both for the tree index and for the inverted files. The performance of the proposed algorithms was evaluated through a series of experiments varying the parameters of a) the cardinality of D in the interval $[10K, 200K]$, b) the cardinality of U, $[10K, 200K]$, c) the number of returned results per user preference k, $[5, 50]$, d) the maximum size of user preferences, $[1, 5]$, and d) the number of returned terms for a query object b, $[2, 5]$. For the Zipfian distribution we varied the value of the characteristic exponent s in the interval $[0.1, 1.0]$. The default setup for the experiments was: $|D| = 20K$, $|U| = 20K$, $k = 10$, $b = 3$ and each the maximum preference size

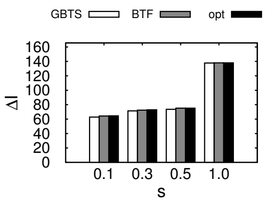
[1]http://www.booking.com
[2]http://commons.apache.org/proper/commons-math/



(a) Varying $|D|$



(b) Varying b



(c) Varying s

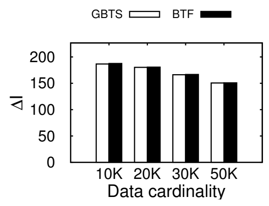
Figure 7.2: Evaluating the quality of results

was set to 5. For each experiment a random set of 20 query objects was selected from D.

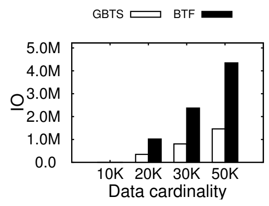
Quality evaluation. We compared the proposed algorithms against an exhaustive algorithm, which examines all $\binom{|A-q.T|}{b}$ term combinations[3] and calculates the optimal set of terms BT. Due to the high processing cost of the exhaustive algorithm even for small values of b, we employed datasets of limited size. The default setting for this series of experiments was $|D| = 100, |U| = 1000$, and $b = 3$. The set of objects D, consisted of a random set of hotels from the area of Catalonia in Spain, and they were selected from Booking.com. The set of preferences U, follows a uniform distribution in Figures 7.2(a) and 7.2(b), and a Zipfian distribution in Figure 7.2(c). Figure 7.2 indicates that both algorithms achieve an increase in the influence score that is very close to the optimal value. The execution time of the exhaustive algorithm was in all cases orders of magnitude larger than the execution time of BTF and GBTS.

Varying |D|. Figure 7.3 illustrates the performance of the algorithms as we vary the number of spatio-textual objects. Figure 7.3(a) indicates that both algorithms perform similarly with respect to the increase of the influence score. As the number of objects increases, the gain in influence score drops as more spatio-textual objects compete for the same number of user-preferences, and therefore it becomes harder for a query object to increase its influence score. Figures 7.3(b) and 7.3(c) indicate that the I/O accesses and the processing time for both algorithms increase when the dataset size raises. As the dataset size increases, the cost of a single TOP_k query increases as well and therefore both

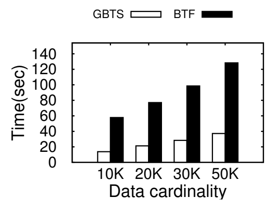
[3]Based on the adopted similarity function, the addition of a term does not have a negative effect on the influence score. In the general case, an exact algorithm should examine $2^{|A|}$ term combinations.



(a) ΔI

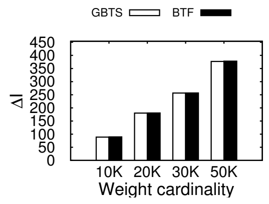


(b) I/O

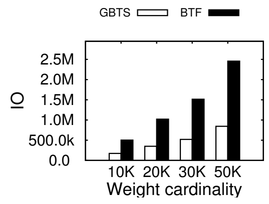


(c) Time

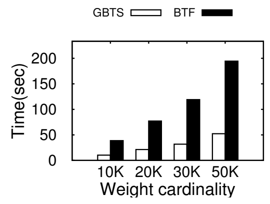
Figure 7.3: Varying data cardinality



(a) ΔI



(b) I/O



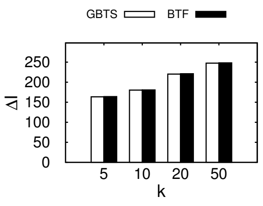
(c) Time

Figure 7.4: Varying preferences cardinality

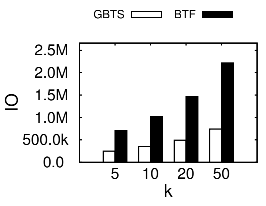
algorithms are affected by the dataset size. The effect on BTF is larger than in GBTS, as BTF accesses the data multiple times in order to create the set of new terms.

Varying |U|. Figure 7.4 depicts the performance of both algorithms as more preferences are processed. When the number of preferences increases there are more user preferences that can be added to the RTOP$_k$ set of an object with an addition of a new set of terms, and therefore the gain in influence score increases as well. The processing cost for both algorithms is expected to raise for a larger number of user preferences, as more preferences have be to examined. Both processing time and I/O cost raise faster for BTF than for GBTS. In particular the processing cost for BTF grows almost by a factor of b faster than GBTS, because BTF has to process the set of preferences b times in order to identify the set of new terms.

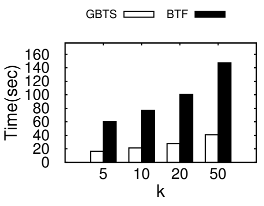
Varying k. When the size of the TOP_k set of each preference increases, the cost of a single TOP_k query increases as well. Figure 7.5 indicates that the increased I/O and processing cost of a TOP_k query affects both algorithms but



(a) ΔI

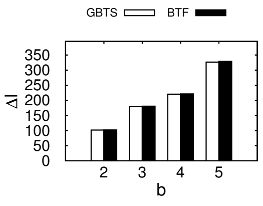


(b) I/O

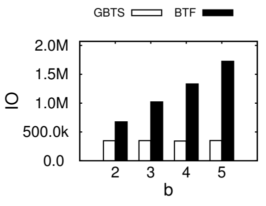


(c) Time

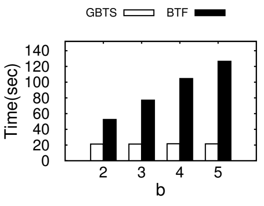
Figure 7.5: Varying k



(a) ΔI



(b) I/O



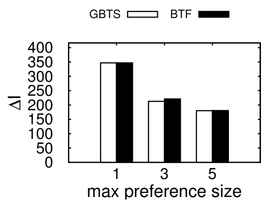
(c) Time

Figure 7.6: Varying b

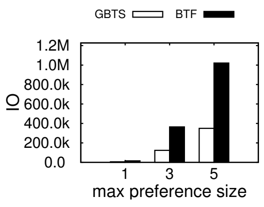
similarly to the increase of datasets' size the effect on BTF is magnified by a factor of b. The influence score gain increases as well, since with the increase of k more objects can be included in the TOP_k set of a user preference, and the necessary increase in the text similarity for a query object q to be added to a TOP_k set of a user preference u becomes smaller.

Varying b. Figure 7.6 illustrates the performance of the algorithms as we vary the number of new terms added to each query object. It is noteworthy the fact that both algorithms behave similarly with respect to the increase of the influence score. The cost of BTF raises linearly with respect to b, which is expected as it has to process the data b times before returning the resulting BT set. On the other hand, GBTS remains unaffected by the increase of the b parameter as it has to access the preferences set only once.

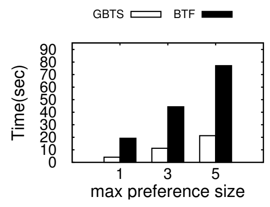
Varying max preference size. Figure 7.7 indicates that as the maximum preference size increases, the possible gain of influence score for a spatio-textual object drops. The reason lies in the fact that for a large user preference u, more terms are required to be added to a spatio-textual object q, for q to enter



(a) ΔI

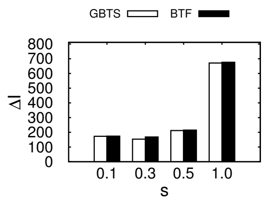


(b) I/O

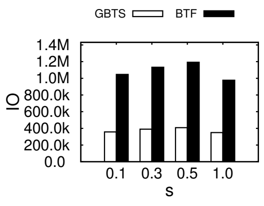


(c) Time

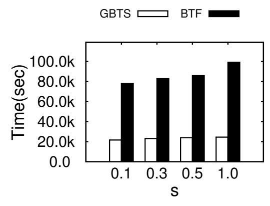
Figure 7.7: Varying max preference size



(a) ΔI



(b) I/O

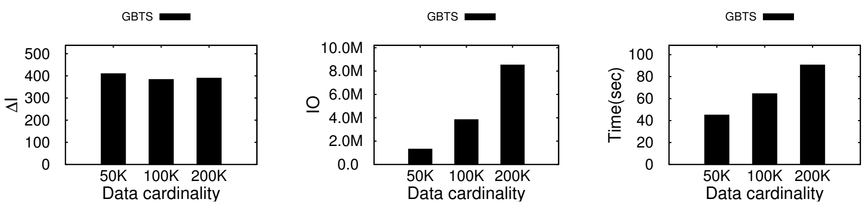


(c) Time

Figure 7.8: Varying zipf distribution

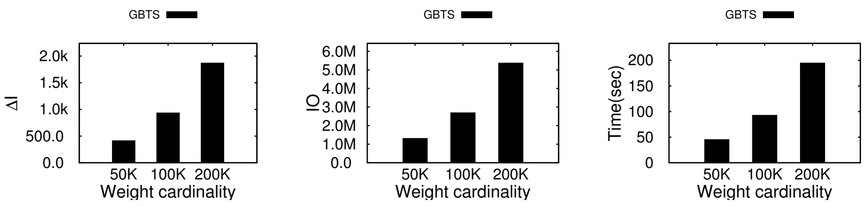
the $TOP_k(u)$ set. Larger queries require more complex TOP_k queries on the indexes and consequently the performance of both algorithms is affected. As expected, the increased cost of the TOP_k queries affects BTF by a larger degree than GBTS.

Zipfian distribution. It is quite usual for the terms of user-preferences to follow a Zipfian distribution. We tested our algorithms against a set of user preferences where the occurrences of terms follow a Zipfian distribution. Figure 7.8 illustrates the experimental results. Similarly to the uniform distribution, GBTS outperforms BTF in terms of I/O accesses and processing time while producing the same gain in influence score. In cases where the exponent of the Zipfian distribution takes high values the gain in influence score raises significantly. Such behavior is expected, because when a small number of distinct terms appear in a large number of user preferences, adding those terms to a spatio-textual object will result in a significant increase of its influence score, since the addition of those terms will allow it to enter the TOP_k set of many user preferences.



(a) ΔI (b) I/O (c) Time

Figure 7.9: Varying data cardinality



(a) ΔI (b) I/O (c) Time

Figure 7.10: Varying data cardinality

Scalability analysis. We evaluated the performance of GBTS against larger datasets to evaluate the scalability of our approach. BTF is not included in the results as it needed excessive time to produce results. The experimental results shown in Figure 7.9 indicate that the processing time of GBTS grows logarithmically with respect to the size of the dataset while the I/O cost increases linearly. In the first TOP_k queries we have an increased number of I/Os, however after a certain number of queries, several nodes of the IR-tree are buffered and as a result the subsequent TOP_k queries induce a limited number of I/O accesses. Figure 7.10 illustrates the performance of GBTS with respect to the cardinality of user preferences set. Both the processing time and the I/O increase linearly with respect to time.

7.8 Conclusion

In this chapter, we address the challenging problem of increasing the influence of a spatio-textual object, by enriching its textual description with at most b carefully selected keywords. In this way, the spatio-textual object's textual relevance to user queries is increased, with the ultimate objective being for the object to become part of the top-k result for many different users. We provide a formal problem statement that is novel and relies on concepts related to top-k and reverse top-k queries. We show that the problem is NP-hard, and we present a greedy solution to the problem. Then, we propose a more efficient algorithm that achieves results of comparable quality, but with significantly lower processing cost. We demonstrate the performance gains of the proposed approach by means of a thorough experimental evaluation that includes real data.

Part IV

Closing Remarks

In this part, we discuss the conclusions drawn in this thesis and directions for future research.

Chapter 8

Conclusion

In this chapter, we summarize the thesis and its contributions, and we present directions for future research.

8.1 Contributions

In this thesis, we have studied the problem of enhancing the visibility of database objects through exploratory search and exploratory analysis of preference queries. We have described algorithms, which instead of presenting to the users a list of results, they organize search results into groups, helping users to have a wide overview of the data content related to their query. In addition, we proposed algorithms for exploratory analysis of preference queries that reveal objects or features that have a potential of being attractive to a wide range of users. This information can be used by companies or organizations that wish to attract new users and make their products or data more visible to their user-base. In summary, the main contributions of this thesis are the following:

- We proposed a framework for summarizing and grouping keyword search results on relational data based on their content and the temporal data of the tuples constituting the results. The search results are organized into non-overlapping periods according to their content. The number of the defined periods is determined by a hierarchical agglomerative algorithm. Each group is described by a set of characterizing terms, which functions as a summary of the results. The framework was tested against a group of users who evaluated its usefulness in understanding the results.

- We introduced algorithms for answering preference queries using combinations instead of single items. The proposed algorithms create combinations of variable size between main and accessory objects. The user is presented combinations consisting of different main objects, while she has the ability to refine her search by focusing to the combination or item she is most attracted and discover more combinations of the same main object. The efficiency of our algorithms was proven theoretically and demonstrated through a thorough experimental evaluation.

- We presented algorithms for the identification of objects that are constantly attractive for a large number of users over a specified period of time. We capitalized on reverse top-k queries to identify influential objects and we presented algorithms which can be combined with any top-k or reverse top-k technique for the efficient calculation of *continuously influential objects*. The proposed algorithms focus on the early identification of the most continuous influential object while they support incremental retrieval of the next continuous influential objects. The performance of the algorithms was evaluated through a detailed experimental procedure.

- We studied the problem of diversity as a problem of identifying objects that appear on the results of users with diverse preferences, i.e., objects with diverse RTOP_k sets. We proposed algorithms that can efficiently calculate sets of diverse objects and we studied their efficiency, performance and scalability in a variety of experimental settings.

- We studied the problem of maximizing the influence score of a spatio-textual object by enhancing its textual description. We showed that selecting a set of terms which maximize the influence score of a spatio-textual object is NP-hard, and we presented two approximate algorithms. The performance of the presented algorithms was evaluated through a detailed experimental study.

8.2 Future work

Our work can be extended in multiple directions. In the following, we present an outline of the open challenges regarding exploratory search and analysis.

Exploratory search in spatial and spatio-textual data. The evolution of geo-location technologies and Internet providing services has led to the generation of an abundance of data containing spatio-textual information. The available information varies from points of interest such as hotels or gasoline stations, to public transportation data and detailed navigation services. This increasing amount of data can be exploited to provide to users detailed information about the places they are interested in. Many structures and algorithms have been proposed for processing spatio-textual [24, 91] and spatial-preference queries [92], while spatio-textual joins of two or more relations have also been studied [14, 73]. Current techniques however, provide limited search capabilities as they return lists of single objects or fixed-structure combinations. Usually, these lists are ranked according to their distance from the query location or a specific attribute (e.g., price). However, user preferences are quite complex, consisting of multiple criteria, which are not all equally important and possibly are better satisfied with a combination of objects. For instance, a tourist visiting a city may be looking for a cheap, clean, central hotel with breakfast. If the tourist does not find a central hotel satisfying her criteria, a possible solution could be a central hotel with a coffee-shop nearby, or a less central hotel close to a metro station. Such options include combinations of spatial objects that the user may not be aware of and she might need several queries to find a solution that fits her needs. Exploratory tools need to take into account the complexity of the users' preferences and the multiple ways spatio-textual objects can be combined and provide the users a comprehensive overview of the available solutions.

Exploratory search on dynamic data and data streams. An important factor that should be taken into account is the increasing volume of generated, stored and processed data. Nowadays, users create a large number of information such as posts, reviews and comments. This user generated data can be exploited in order to provide the users a better overview of the content they are exploring. In order to be able to handle efficiently the increase volume of data, a large number of data management systems use NoSQL databases. However NoSQL databases offer limited search capabilities in order to provide high data availability and most existing approaches assume that documents are static, i.e., the documents do not change over time [114]. Consequently, a significant part of the generated information is not readily available to the users who are searching the database content. Additional limitations arise from the distributed nature of those systems and the lack of database schemas, facts that induce a high computational cost during the generation of object combinations. We believe that current techniques need to be modified and new techniques should be developed to ensure efficiency and scalability in exploratory search in distributed and highly dynamic environments.

Analyzing exploratory queries. Exploratory search poses new challenges in data analysis as well. During the analysis of user preferences, it should be taken into account the possibility that users may be presented with object combinations instead of single objects. This consideration changes utterly the way the visibility of an object is determined and estimated, since an object may not be highly ranked as a single object, but it can participate in a highly ranked combination. Our plans are to examine how current techniques for processing reverse top-k queries and influential top-m queries can be improved and extended in order to take into account the possibility of evaluating combinations in addition to single objects.

Exploring Big Data. Today, a large number of enterprises and organizations receive vast amounts of data, generated by traditional sources, social media and user queries. A significant part of this data contains spatio-temporal and textual information. Due to the large volume, the high velocity and possibly the short life-span of the incoming data, it is hard to create an index that stores spatial, temporal and textual information and is efficient at the same time in search, updates and resource usage. A recent study [19] is indicative of the fact that search efficiency and index size in spatio-textual data are conflicting with one another. These restrictions impede the exploration and analysis of the available data. Our aim is to propose new algorithms and structures that will allow users to explore large volumes of highly dynamic data and enable enterprises to analyze both data and user queries in order to make their content more visible and easily accessible among a vast amount of information.

8.3 Outlook

Today, information is generated in an accelerating rate by a wide range of sources varying from enterprises and organizations to simple users. Users have difficulties exploring the available data and finding the information they are interested in, fact that has a direct effect on enterprises since their database content is not visible to the users. In this thesis, we have proposed algorithms for exploratory search that allow users locate objects they are interested in. In addition we presented algorithms for exploratory analysis that allow enterprises and organizations to identify objects and object properties that are attractive to users.

The modern era poses new challenges in data exploration as data become larger in volume and richer in context, while user preferences become more detailed and complicated containing often space- and time-related data. It is therefore necessary to develop new techniques for exploratory search and exploratory analysis that will enable efficient exploration of large collections and highly dynamic streams of data containing spatial, temporal and textual information.

References

[1] Parag Agrawal and Jennifer Widom. Confidence-aware join algorithms. In *Proceedings of ICDE*, pages 628–639, 2009.

[2] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: recommending advertiser bid phrases for web pages. In *Proceedings of WWW*, pages 13–24, 2013.

[3] Rakesh Agrawal and Edward L. Wimmers. A framework for expressing and combining preferences. In *Proceedings of SIGMOD*, pages 297–306, 2000.

[4] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A system for keyword-based search over relational databases. In *Proceedings of ICDE*, pages 5–16, 2002.

[5] David C. Anastasiu, Byron J. Gao, and David Buttler. A framework for personalized and collaborative clustering of search results. In *Proceedings of CIKM*, pages 573–582, 2011.

[6] Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of SIGMOD*, pages 781–792, 2011.

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

[8] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *Proceedings of VLDB*, pages 564–575, 2004.

[9] Ori Ben-Yitzhak, Nadav Golbandi, Nadav Har'El, Ronny Lempel, Andreas Neumann, Shila Ofek-Koifman, Dafna Sheinwald, Eugene J. Shekita, Benjamin Sznajder, Sivan Yogev, and Sivan Yogev. Beyond basic faceted search. In *Proceedings of WSDM*, pages 33–44, 2008.

[10] Thomas Bernecker, Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz, Shiming Zhang, and Andreas Züfle. Inverse queries for multidimensional spaces. In *Proceedings of SSTD*, pages 330–347, 2011.

[11] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proceedings of ICDE*, pages 431–440, 2002.

[12] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[13] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of ICDE*, pages 421–430, 2001.

[14] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. Spatio-textual similarity joins. *Proceedings of the VLDB Endowment*, 6(1):1–12, 2012.

[15] Xin Cao, Gao Cong, and Christian S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *Proceedings of the VLDB Endowment*, 3(1):373–384, 2010.

[16] Yuan-Chi Chang, Lawrence D. Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith. The Onion technique: Indexing for linear optimization queries. In *Proceedings of SIGMOD*, pages 391–402, 2000.

[17] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Transactions on Database Systems*, 31(3):1134–1168, 2006.

[18] Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *Proceedings of VLDB*, pages 397–410, 1999.

[19] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: An experimental evaluation. *Proceedings of the VLDB Endowment*, 6(3):217–228, 2013.

[20] Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. Indexing reverse top-k queries in two dimensions. In *Proceedings of DASFAA*, pages 201–208, 2013.

[21] Yejin Choi, Marcus Fontoura, Evgeniy Gabrilovich, Vanja Josifovski, Maurício R. Mediano, and Bo Pang. Using landing pages for sponsored search ad selection. In *Proceedings of WWW*, pages 251–260, 2010.

[22] Susan Cholette, Özgür Özlük, and Mahmut Parlar. Optimal keyword bids in search-based advertising with stochastic advertisement positions. *Journal of Optimization Theory and Applications*, 152(1):225–244, 2012.

[23] Jan Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466, 2003.

[24] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *Proceedings of the VLDB Endowment*, 2(1):337–348, 2009.

[25] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. Answering top-k queries using views. In *Proceedings of VLDB*, pages 451–462, 2006.

[26] Ke Deng, Xin Li, Jiaheng Lu, and Xiaofang Zhou. Best keyword cover search. *IEEE Transactions on Knowledge and Data Engineering*, 27(1):61–73, 2014.

[27] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1/2):143–175, 2001.

[28] Christos Doulkeridis, Akrivi Vlachou, Kjetil Nørvåg, Yannis Kotidis, and Neoklis Polyzotis. Processing of rank joins in highly distributed systems. In *Proceedings of ICDE*, pages 606–617, 2012.

[29] Marina Drosou and Evaggelia Pitoura. Diversity over continuous data. *IEEE Data Engineering Bulletin Issues*, 32(4):49–56, 2009.

[30] Marina Drosou and Evaggelia Pitoura. ReDRIVE: result-driven database exploration through recommendations. In *Proceedings of CIKM*, pages 1547–1552, 2011.

[31] Marina Drosou and Evaggelia Pitoura. DisC diversity: result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment*, 6(1):13–24, 2012.

[32] Marina Drosou and Evaggelia Pitoura. Dynamic diversification of continuous data. In *Proceedings of EDBT*, pages 216–227, 2012.

[33] Erhan Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

[34] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[35] Georgios John Fakas. A novel keyword search paradigm in relational databases: Object summaries. *Data and Knowledge Engineering*, 70(2):208–229, 2011.

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

[37] Jonathan Finger and Neoklis Polyzotis. Robust and efficient algorithms for rank join evaluation. In *Proceedings of SIGMOD*, pages 415–428, 2009.

[38] Ariel Fuxman, Panayiotis Tsaparas, Kannan Achan, and Rakesh Agrawal. Using the wisdom of the crowds for keyword generation. In *Proceedings of WWW*, pages 61–70, 2008.

[39] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Macmillan Higher Education, 1979.

[40] Shen Ge, Leong Hou U, Nikos Mamoulis, and David W. Cheung. Efficient all top-k computation - a unified solution for all top-k, reverse top-k and top-m influential queries. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1015–1027, 2013.

[41] Orestis Gkorgkas, Kostas Stefanidis, and Kjetil Nørvåg. A framework for grouping and summarizing keyword search results. In *Proceedings of ADBIS*, pages 246–259, 2013.

[42] Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Exploratory product search using top-k join queries. *(Under submission).*

[43] Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Discovering influential data objects over time. In *Proceedings of SSTD*, pages 110–127, 2013.

[44] Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Efficient processing of exploratory top-k joins. In *Proceedings of SSDBM*, pages 35:1–35:4, 2014.

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

[46] Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Maximizing influence of spatio-textual objects through keyword selection. To appear in *Proceedings of SSTD*, 2015.

[47] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. In *Proceedings of VLDB*, pages 419–428, 2000.

[48] Xi Guo and Yoshiharu Ishikawa. Multi-objective optimal combination queries. In *Proceedings of DEXA*, pages 47–61, 2011.

[49] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. Fixed-precision estimation of join selectivity. In *Proceedings of PODS*, pages 190–201, 1993.

[50] Dirk Habich, Wolfgang Lehner, and Alexander Hinneburg. Optimizing multiple top-k queries over joins. In *Proceedings of SSDBM*, pages 195–204, 2005.

[51] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. BLINKS: ranked keyword searches on graphs. In *Proceedings of SIGMOD*, pages 305–316, 2007.

[52] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[53] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proceedings of VLDB*, pages 850–861, 2003.

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

[55] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proceedings of VLDB*, pages 670–681, 2002.

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

[57] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB Journal*, 13(3):207–221, 2004.

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

[59] Jeffrey Jestes, Jeff M. Phillips, Feifei Li, and Mingwang Tang. Ranking large temporal data. *Proceedings of the VLDB Endowment*, 5(11):1412–1423, 2012.

[60] Wen Jin, Martin Ester, Zengjian Hu, and Jiawei Han. The multi-relational skyline operator. In *Proceedings of ICDE*, pages 1276–1280, 2007.

[61] Wen Jin, Michael D. Morse, Jignesh M. Patel, Martin Ester, and Zengjian Hu. Evaluating skylines in the presence of equijoins. In *Proceedings of ICDE*, pages 249–260, 2010.

[62] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of VLDB*, pages 505–516, 2005.

[63] Mohamed E. Khalefa, Mohamed F. Mokbel, and Justin J. Levandoski. Prefjoin: An efficient preference-aware join operator. In *Proceedings of ICDE*, pages 995–1006, 2011.

[64] Werner Kießling. Foundations of preferences in database systems. In *Proceedings of VLDB*, pages 311–322, 2002.

[65] Jia-Ling Koh, Chen-Yi Lin, and ArbeeL.P. Chen. Finding k most favorite products based on reverse top-t queries. *VLDB Journal*, pages 1–24, 2013.

[66] Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos. Continuous top-k dominating queries. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):840–853, 2012.

[67] Georgia Koutrika, Zahra Mohammadi Zadeh, and Hector Garcia-Molina. Data clouds: summarizing keyword search results over structured data. In *Proceedings of EDBT*, pages 391–402, 2009.

[68] Mong-Li Lee, Wynne Hsu, Ling Li, and Wee Hyong Tok. Consistent top-k queries over time. In *Proceedings of DASFAA*, pages 51–65, 2009.

[69] Cuiping Li, Beng Chin Ooi, Anthony K. H. Tung, and Shan Wang. DADA: a data cube for dominant relationship analysis. In *Proceedings of SIGMOD*, pages 659–670, 2006.

[70] Feifei Li, Ke Yi, and Wangchao Le. Top-k queries on temporal data. *VLDB Journal*, 19(5):715–733, October 2010.

[71] Chen-Yi Lin, Jia-Ling Koh, and Arbee L. P. Chen. Determining (k)-most demanding products with maximum expected number of total customers. *IEEE Transactions on Knowledge and Data Engineering*, 25(8):1732–1747, 2013.

[72] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting stars: The k most representative skyline operator. In *Proceedings of ICDE*, pages 86–95, 2007.

[73] Sitong Liu, Guoliang Li, and Jianhua Feng. Star-join: spatio-textual similarity join. In *Proceedings of CIKM*, pages 2194–2198, 2012.

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

[75] Jiaheng Lu, Pierre Senellart, Chunbin Lin, Xiaoyong Du, Shan Wang, and Xinxing Chen. Optimal top-k generation of attribute combinations based on ranked lists. In *Proceedings of SIGMOD*, pages 409–420, 2012.

[76] Ying Lu, Jiaheng Lu, Gao Cong, Wei Wu, and Cyrus Shahabi. Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search. *ACM Transactions on Database Systems*, 39(2):13, 2014.

[77] Nikos Mamoulis, Man Lung Yiu, Kit Hung Cheng, and David W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Transactions on Database Systems*, 32(3):19, 2007.

[78] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251 – 266, 1995.

[79] Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems*, 29(2):319–362, 2004.

[80] Davide Martinenghi and Marco Tagliasacchi. Cost-aware rank join with random and sorted access. *IEEE Transactions on Knowledge and Data Engineering*, 24(12):2143–2155, 2012.

[81] Muhammed Miah, Gautam Das, Vagelis Hristidis, and Heikki Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *Proceedings of ICDE*, pages 356–365, 2008.

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

[83] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *Proceedings of VLDB*, pages 281–290, 2001.

[84] Panagiotis Papadimitriou, Hector Garcia-Molina, Ali Dasdan, and Santanu Kolay. Output URL bidding. *Proceedings of the VLDB Endowment*, 4(3):161–172, 2010.

[85] Aditya G. Parameswaran and Hector Garcia-Molina. Recommendations with prerequisites. In *Proceedings of RecSys*, pages 353–356, 2009.

[86] Jaehui Park and Sang-goo Lee. Keyword search in relational databases. *Knowledge and Information Systems*, 26(2):175–193, 2011.

[87] Zhaohui Peng, Jun Zhang, Shan Wang, and Lu Qin. Treecluster: Clustering results of keyword search over databases. In *Proceedings of WAIM*, pages 385–396, 2006.

[88] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Keyword search in databases: the power of RDBMS. In *Proceedings of the SIGMOD*, pages 681–694, 2009.

[89] Sekharipuram S. Ravi, Daniel J. Rosenkrantz, and Giri K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[90] Sujith Ravi, Andrei Z. Broder, Evgeniy Gabrilovich, Vanja Josifovski, Sandeep Pandey, and Bo Pang. Automatic generation of bid phrases for online advertising. In *Proceedings of WSDM*, pages 341–350, 2010.

[91] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Proceedings of SSTD*, pages 205–222, 2011.

[92] João B. Rocha-Junior, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Efficient processing of top-k spatial preference queries. *Proceedings of the VLDB Endowment*, 4(2):93–104, 2010.

[93] Senjuti Basu Roy, Sihem Amer-Yahia, Ashish Chawla, Gautam Das, and Cong Yu. Constructing and exploring composite items. In *Proceedings of SIGMOD*, pages 843–854, 2010.

[94] Senjuti Basu Roy, Haidong Wang, Gautam Das, Ullas Nambiar, Mukesh K. Mohania, and Mukesh K. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proceedings of CIKM*, pages 13–22, 2008.

[95] Karl Schnaitter and Neoklis Polyzotis. Evaluating rank joins with optimal cost. In *Proceedings of PODS*, pages 43–52, 2008.

[96] Karl Schnaitter and Neoklis Polyzotis. Optimal algorithms for evaluating rank joins in database systems. *ACM Transactions on Database Systems*, 35(1), 2010.

[97] Joachim Selke, Christoph Lofi, and Wolf-Tilo Balke. Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. *Proceedings of the VLDB Endowment*, 5(6):538–549, 2012.

[98] Abraham Silberschatz, Alexander Tuzhilin, and Alexander Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *Proceedings of KDD*, pages 275–281, 1995.

[99] Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB Journal*, 17(1):117–149, 2008.

[100] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. *You May Also Like* results in relational databases. In *Proceedings of PersDB*, pages 37–42, 2009.

[101] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. PerK: personalized keyword search in relational databases through preferences. In *Proceedings of EDBT*, pages 585–596, 2010.

[102] Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei. Distance-based representative skyline. In *Proceedings of of ICDE*, pages 892–903, 2009.

[103] Yufei Tao, Vagelis Hristidis, Dimitris Papadias, and Yannis Papakonstantinou. Branch-and-bound processing of ranked queries. *Information Systems*, 32(3):424–445, 2007.

[104] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of VLDB*, pages 648–659, 2004.

[105] George Tsatsaronis, Matthias Reimann, Iraklis Varlamis, Orestis Gkorgkas, and Kjetil Nørvåg. Efficient community detection using power graph analysis. In *Proceedings of LSDS-IR*, pages 21–26, 2011.

[106] Leong Hou U, Nikos Mamoulis, Klaus Berberich, and Srikanta J. Bedathur. Durable top-k search in document archives. In *Proceedings of SIGMOD*, pages 555–566, 2010.

[107] Vijay K Vaishnavi and William Kuechler Jr. *Design science research methods and patterns: innovating information and communication technology.* CRC Press, 2007.

[108] George Valkanas, Apostolos N. Papadopoulos, and Dimitrios Gunopulos. SkyDiver: a framework for skyline diversification. In *Proceedings of EDBT*, pages 406–417, 2013.

[109] Marcos R. Vieira, Humberto Luiz Razente, Maria Camila Nardini Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina Jr., and Vassilis J. Tsotras. DivDB: a system for diversifying query results. *Proceedings of the VLDB Endowment*, 4(12):1395–1398, 2011.

[110] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Monochromatic and bichromatic reverse top-k queries. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1215–1229, 2011.

[111] Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Monitoring reverse top-k queries over mobile devices. In *Proceedings of MobiDE*, pages 17–24, 2011.

[112] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Identifying the most influential data objects with reverse top-k queries. *Proceedings of the VLDB Endowment*, 3(1):364–372, 2010.

[113] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *Proceedings SIGMOD*, pages 481–492, 2013.

[114] Christian von der Weth and Anwitaman Datta. Multiterm keyword search in nosql systems. *IEEE Internet Computing*, 16(1):34–42, 2012.

[115] Qian Wan, Raymond Chi-Wing Wong, Ihab F. Ilyas, M. Tamer Özsu, and Yu Peng. Creating competitive products. *Proceedings of the VLDB Endowment*, 2(1):898–909, 2009.

[116] Ryen W. White and Resa A. Roth. *Exploratory Search: Beyond the Query-Response Paradigm.* Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.

[117] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. Joint top-k spatial keyword query processing. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1889–1903, 2012.

[118] Tianyi Wu, Yizhou Sun, Cuiping Li, and Jiawei Han. Region-based online promotion analysis. In *Proceedings of EDBT*, pages 63–74, 2010.

[119] Tianyi Wu, Dong Xin, Qiaozhu Mei, and Jiawei Han. Promotion analysis in multi-dimensional space. *Proceedings of the VLDB Endowment*, 2(1):109–120, 2009.

[120] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood. Breaking out of the box of recommendations: from items to packages. In *Proceedings of RecSys*, pages 151–158, 2010.

[121] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood. Efficient rank join with aggregation constraints. *Proceedings of the VLDB Endowment*, 4(11):1201–1212, 2011.

[122] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood. Efficient top-k query answering using cached views. In *Proceedings of EDBT*, pages 489–500, 2013.

[123] Dong Xin, Chen Chen, and Jiawei Han. Towards robust indexing for ranked queries. In *Proceedings of VLDB*, pages 235–246, 2006.

[124] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, and Yuguo Chen. Efficient maintenance of materialized top-k views. In *Proceedings of ICDE*, pages 189–200, 2003.

[125] Albert Yu, Pankaj K. Agarwal, and Jun Yang. Processing a large number of continuous preference top-k queries. In *Proceedings of SIGMOD*, pages 397–408, 2012.

[126] Oren Zamir and Oren Etzioni. Grouper: A dynamic clustering interface to web search results. *Computer Networks*, 31(11-16):1361–1374, 1999.

[127] Wangda Zhang, Reynold Cheng, and Ben Kao. Evaluating multi-way joins over discounted hitting time. In *Proceedings of ICDE*, pages 724–735, 2014.

[128] Weinan Zhang, Dingquan Wang, Gui-Rong Xue, and Hongyuan Zha. Advertising keywords recommendation for short-text web pages using Wikipedia. *ACM Transactions on Intelligent Systems and Technology*, 3(2):36, 2012.

[129] Zhao Zhang, Cheqing Jin, and Qiangqiang Kang. Reverse k-ranks query. *Proceedings of the VLDB Endowment*, 7(10):785–796, 2014.