INTERACTIVE DATA ANALYSIS AND ITS APPLICATIONS ON MULTI-STRUCTURED DATASETS

FENG ZHAO

NATIONAL UNIVERSITY OF SINGAPORE

2013

National University of Singapore

Doctoral Thesis

Interactive Data Analysis and Its Applications on Multi-structured Datasets

Author:
Feng Zhao

Supervisor:
Prof. Anthony K.H. Tung

A thesis submitted
for the degree of Doctor of Philosophy

in the

Department of Computer Science
School of Computing

2013

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Feng Zhao

July, 2013

Acknowledgement

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this research. I would like to express my gratitude to all of them.

Foremost, I would like to express my sincere gratitude to my advisor Professor Anthony K. H. Tung for the continuous support of my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. He has been my inspiration as I hurdle all the obstacles during my entire period of Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Professor Chee-Yong Chan and Professor Roger Zimmermann, for their encouragement, insightful comments, and suggestions to improve the quality of the thesis.

I am grateful to my project supervisor Professor Beng Chin Ooi. He set a good example to me in my research as well as in my life. As he said, it is ourselves who determine our path. His attitude inspired me to work hard and overcome all the difficulty during the last five years. My sincere thanks also goes to Professor Gautam Das, Professor Kian-Lee Tan, for collaborating with me on my research papers and giving many insightful comments on my work.

I thank my fellow labmates in iData Group: Bingtian Dai, Chen Liu, Meiyu Lu, Zhan Su, Nan Wang, Xiaoli Wang, Shanshan Ying, Dongxiang Zhang, Jingbo Zhang, Zhenjie Zhang, Wei Kang, Jingbo Zhou and Yuxin Zheng, for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all

the fun we have had in the last five years. Also I thank all my colleagues in Database Research Laboratories and many friends in Singapore as we shared a wonderful time in Singapore together.

Last but not the least, I would like to thank my family: my parents Lihang Zhao and Jingping Guo, for giving birth to me at the first place, taking care of me and supporting me spiritually throughout my life.

I am particularly grateful to my dearest Wenyi Chen for all the insightful thoughts and helping in the journey of life, proving her love and support during the whole course of this work.

Contents

Summary

Data analytics in databases has received a lot of attention in the database community as it is an effective process of inspecting, cleaning, transforming, and modeling data with the goal of highlighting useful information, suggesting conclusions, and supporting decision making. However, as dataset cardinality increases dramatically nowadays, it remains a challenge to make the analytical process scalable as well as keep the process interactive, visual intuitive and user controllable. As such, it is important to provide a framework to support data interactive analytics in a scalable manner.

This thesis first addresses a user preference query on top of multi-dimensional datasets. We propose to elicit the preferred ordering of a user by utilizing skyline objects as the representatives of possible orderings. With the notion of *order-based representative skylines*, representatives are selected based on the orderings that they represent. To further facilitate preference exploration, a hierarchical clustering algorithm is applied to compute a denogram on the skyline objects. By coupling the hierarchical clustering with visualization techniques, this framework allows users to refine their preference weight settings by browsing the hierarchy.

To further extend the interactive data analytics, we propose to apply the hierarchical browsing approach in the application of keyword search in databases. To this end, we implement a novel system allowing users to perform diverse, hierarchical browsing on keyword search results. It partitions the answer trees in the keyword search results by selecting k diverse representatives from the answer trees, separating the answer trees into k groups based on their similarity to the representatives and then recursively applying the partitioning for each group. By constructing summarized result for the answer trees in each of the k groups, we provide a visual interface for users to quickly locate the results that they desire.

Finally, we introduce a novel subgraph concept to capture the cohesion in social interactions, and propose an I/O efficient approach to discover cohesive subgraphs. In addition, we develop an analytical system which allows users to perform intuitive, visual browsing on a large scale social networks. We hierarchically visualizes the subgraph out on orbital layout, in which more important social actors are located in the center. By summarizing textual interactions between social actors as the tag cloud, users can quickly locate active social communities and their interactions in a unified view.

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid development of database system research, modern database systems can process terabytes to petabytes of data, or incorporate non-structural data and multi-structured data sources and types. However, despite the considerable advancements in high performance, large storage, and high computation power, there is a lack of attention in identifying, clustering, classifying, and interpreting a large spectrum of the underlying information, knowledge and intelligence. Database researchers recently realized that making database usable deserves more attention [67]. It is very important to design better approaches to retrieve what users need effectively and intuitively, due to the large scale of datasets and complex data types in existing database applications. In view of this, we introduced the interactive data analysis into database research.

Data analysis is an effective process of inspecting, cleaning, transforming, and modeling data with the goal of highlighting useful information, suggesting conclusions, and supporting decision making [76], which is widely used in different domains, such as business, science, and policy. In general, it can be divided into three major phases: data cleaning, initial data analysis and main data analysis [2]. Data cleaning is a procedure during which the data are inspected and erroneous data are corrected without information loss. The initial data analysis is the next phase which does not directly aim at answering the original research question, but takes quality of data and measurements as its main concern and performs initial transformations of data. In the main analysis phase, analysis aims at answering the research question as well as

any other relevant analysis. In this thesis, we focus on the main data analysis phase, with the assumption that the data we need to analyze is already cleaned and stored in database systems with the format we need. As such, based on different database applications on various multi-structured datasets, we propose different analyzing solutions to extract information out of data and to show results to users in an interactive manner.

There are various of data analysis methods, some of which include data mining, text analytics, business intelligence, and data visualizations. One important branch is data mining, which is the computational process of discovering patterns in large data sets. Related to data mining, text mining, roughly equivalent to text analytics, extracts and classifies information from textual sources, a species of unstructured data. Business intelligence is commonly applied in the business area that relies heavily on aggregation, focusing on business information. In statistical applications, data analysis is divided into descriptive statistics, exploratory data analysis (EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data while CDA on confirming or falsifying existing hypotheses. My research topic specializes in interactive data analysis in databases, close to the data mining and data visualization. Differently, we are more interested in querying and searching problems on the large scale indexed datasets and try to implement visualized systems to capture the most important information with respect to users' interests.

To better explain the blueprint of the thesis, we depict the overall framework as in Figure 1.1. In general, it can be divided into three layers, including data storage layer, data analysis engine and data visualization interface. In this thesis, we make use of the data storage layout to organize the data with respect to different data types and my study focuses on the above two layers. We propose different data analyzing techniques for different problems and visualize them in visualization interface, so that users can interact with the system and quickly understand the meaning of the analyzing results.

In the subsequent sections, an overview of the scope of study for this thesis is presented first. Then, we describe the research aims, the general methodology, the contributions and the outline of the thesis.

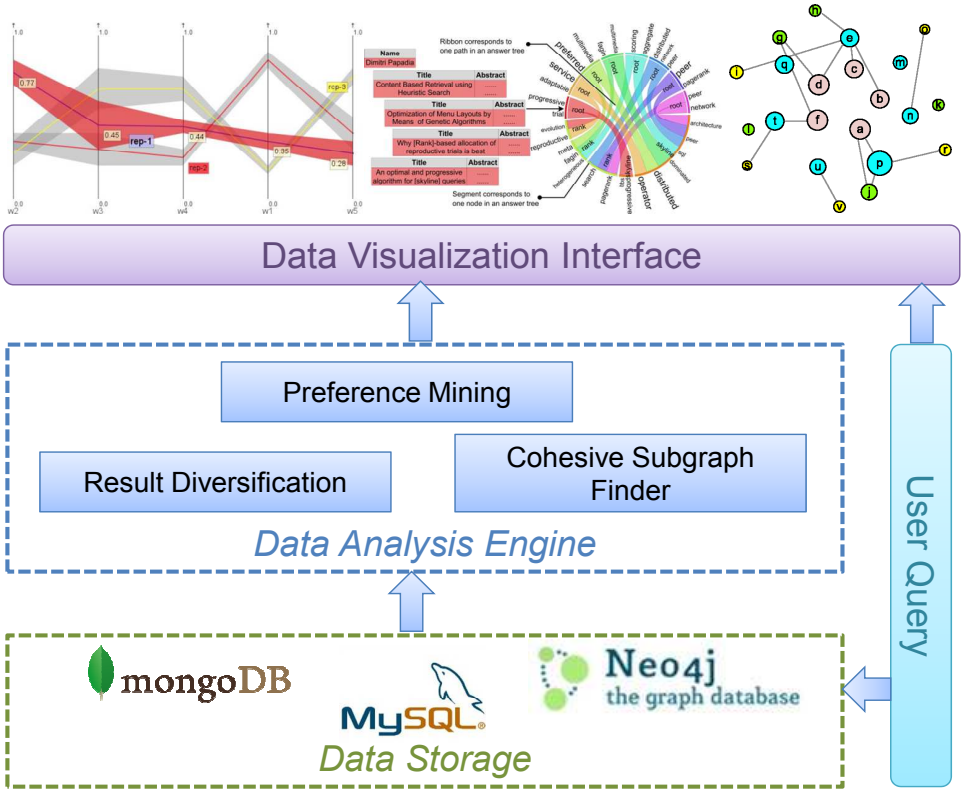


Figure 1.1: The Overview Framework.

1.1 Scope of Study

Since interactive data analysis in databases is a very broad area, my study will focus on the following key topics. A brief introduction is given below and in-depth discussion will be found in subsequent chapters.

1.1.1 Preference Mining

The notion of preference occurs naturally in every context where one talks about human decision or choice. In the context of database queries, faced with information overload, database users seek ways to obtain not necessarily all answers to queries but rather the best, most preferred answers [70]. Personalization of e-services poses new challenges to database technology, demanding a powerful and flexible modeling technique for complex preferences. Preferences, treated as soft constraints, are utilized in multi-criteria decision situations to identify the preferred results. A common

approach assumes that a monotonic ranking (or preference) function $P(\cdot)$ is provided and the user will specify his/her preference by setting a set of weights to rank the importance of data objects. In this thesis, we aim at eliciting a users preference by adopting this preference mining setting.

Computing preference queries have been a well studied problem in the database community [70, 28, 68, 89]. Among various possible problem settings, a common one [68, 89] assumes that a monotonic ranking (or preference) function $P(\cdot)$ is provided and the user will specify his/her preference by setting a set of weights $w = \{w_1, w_2, \ldots, w_d\}$ which are used within the preference function to rank the importance of data objects. Each of the weight w_i represents the importance of an attribute A_i describing the objects and thus w_1, ..., w_d describe the importance of d attributes A_1,..., A_d. In such a problem setting, it is also assumed that the order of preference for the domain values of each attribute are known. As such, if the user is able to specify the settings of the weights correctly, then the objects will be ranked in the correct order of his/her preference and then the problem becomes one of retrieving the objects efficiently based on the order. However, if the user is unsure of his/her preference (which is typically the case), it is crucial to interact with the user to obtain a correct set of weights that represent his/her preference. Designing an effective mechanism to elicit the preference of the user is exactly what we set to do in this work.

To elicit an user's preference, a common approach is to present the user with a set of objects, and based on his/her choice of the objects, we can potentially infer the correct weights. To ensure that all possible choices are well covered, the set of objects being presented must be carefully selected. More often than not, this involves clustering the objects into different groups and a representative from each group will be presented to the user. By stating the preference for a particular representative, he/she implicitly provides an approximate setting for the set of weights and also indicates that he/she prefers the group associated with the representative. Further refinement can then be made by repeating the procedure on the selected group and selecting more representatives from the group. However, such an approach will bring about a catch-22 situation. In a typical clustering operation, an appropriate similarity function will be required to determine the similarity between the objects. Such a similarity function will usually be determined by weighting the importance of the attributes based on the user's input. The user, unfortunately, is relying on the

clustering results to help him/her determine the importance of these attributes in the preference function!

In view of this, much research has been done on the problem of skyline computation [17, 29, 98, 72, 94, 74]. An object p dominates another object q if p is better or equal to q in all attributes and at least better than q in one. The skylines objects are objects that are not dominated by any other objects in the set. Based on this definition, it can be shown that the set of skyline objects for a dataset is insensitive to (1) the weight assigned to each attribute and (2) the preference function being adopted. More importantly, given any monotonic preference function, it is guaranteed that the top one will always be a skyline object. More formally, let $\pi_w(D)$ denote the preferred ordering of a set of objects given weight setting w and $\pi_w(D)[i]$ denote the i^{th} object in this ordering, then $\pi_w(D)[1]$ must be a skyline object. In this sense, we will refer to $\pi_w(D)[1]$ as a **representative** of $\pi_w(D)$ and thus every possible ordering based on different weight settings will be represented by one of the skyline objects.

Since the set of skyline objects is insensitive to the setting of weights and gives full coverage as representatives of $\pi_w(D)$, it thus makes sense to present the skylines to the user for selection and infer the weight setting that represents the user's preference based on his/her selection[1]. However, it has been shown in [98] that the expected number of skyline objects is $\Theta(\ln^{d-1} n/(d-1)!)$ for a random dataset where d is the dimensionality of the data. The large number of skyline objects for high dimensional dataset is ironical since this is the situation in which users have the most difficulty determining their preferences and comparing products. Various efforts have been made [80, 112] to overcome this problem by selecting k representatives from a large set of skylines. While we will discuss these later, it suffices to point out here that none of these works tries to bring the preference function and its ordering of the objects back into the picture.

1.1.2 Keyword Search in Databases

It has become highly desirable to provide users with flexible ways to query/search information over databases as simple as keyword search like Google search [126].

[1] Note that since multiple settings of w can be represented by the same skyline object, this inference is only approximate.

Keyword search over databases focuses on finding structural information among objects in a database using a set of keywords. Such structural information to be returned can be either trees or subgraphs representing how the objects, that contain the required keywords, are interconnected in a relational database or an XML database. The structural keyword search is completely different from finding documents that contain all the user-given keywords. The former focuses on the interconnected object structures, whereas the latter focuses on the object content. However, keyword search queries can often return too many complex answers. As a result, exploring and understanding keyword search results can be time consuming and not user-friendly. In this thesis, we expect to make the keyword search in databases more intuitive to use to finding desired answers.

With an increasing amount of textual data being stored in relational databases, keyword search is well recognized as a convenient and effective approach to retrieve results without knowing the underlying schema or learning a query language [3, 64, 69, 61]. The result of a keyword query is often modeled as a compact substructure, such as a tree or a graph, which connects keyword tuples to include all the keywords. Potentially, a user could discover underlying relationships and the semantics based on structural answers.

However, keyword search queries can often return too many answers. This is because the semantics captured in a keyword query is limited, and the tuples that keywords are located in might come from different tables and connect with each other in many ways. As a result, exploring and understanding keyword search results can be time consuming and not user-friendly. To illustrate this, we describe a simple example on CiteSeerX[2] dataset. Figure 1.2 shows the schema graph G_S, in which nodes are associated with tables and edges indicate foreign key references.

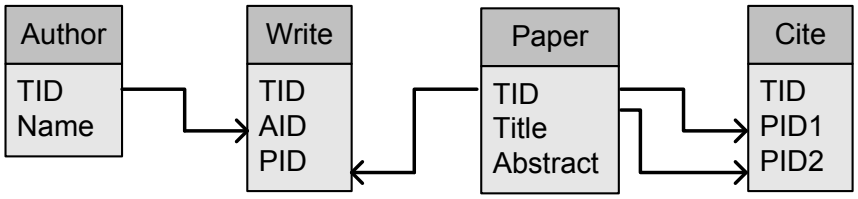


Figure 1.2: CiteSeerX Schema Graph

[2]http://citeseerx.ist.psu.edu/

Example 1 *Consider a keyword query on "skyline" and "rank" over the CiteSeerX dataset. There are* 78 *tuples containing the keyword "skyline", and* 729 *tuples containing the keyword "rank". A snapshot of keyword tuples are presented in Table 1.1, and part of the answers related to these tuples are shown in Figure 1.3. For clear illustration, we use "a" to denote an author and "p" to denote a paper. It can be seen that the relationship between them varies a lot even for fixed keyword tuples. Presenting and exploring the results of this keyword query will be difficult.*

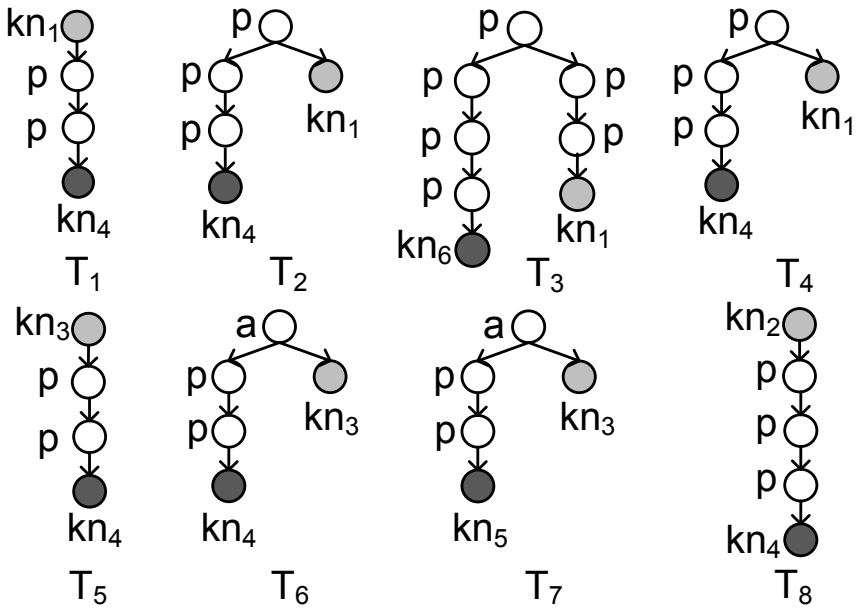


Figure 1.3: Search Result Examples

Table 1.1: The Snapshot of Keyword Tuples

ID	Content Excerpt
kn_1	The [Skyline] Operator
kn_2	[Skyline] with Presorting
kn_3	An Optimal and Progressive Algorithm for [Skyline] Queries
kn_4	Merging [Ranks] from Heterogeneous Internet Sources
kn_5	Why [Rank]-Based Allocation of Reproductive Trials is Best
kn_6	The PageRank Citation [Rank]ing

A typical solution for massive keyword search results is to return top-k answers according to relevant scores [61]. Sophisticated ranking strategies have been developed to attempt to capture the search intention of a user. Without knowing the schema, however, it is hard for a user to explicitly express the preference. For instance, the

query {skyline, rank} aims to discover the relationship between them, but it is difficult to indicate which keyword is more important or what types of path connections are meaningful before a user realizes what can be found in the dataset. Even if it is possible to estimate users' preference, the top-k results usually include many overlapped answers that are redundant to present. As an extreme case in Example 1, T_2 and T_4 share two keyword nodes and even an identical answer structure.

Ideally, the results for keyword query would properly account for the interests of the overall user population [31]. In view of this, result diversification has been well studied in information retrieval community [31, 52, 5]. More explicitly, they try to put documents with broad information and different semantics in the first page of search interface. Consequently, the search engine improves users' satisfaction since each user has a high possibility of efficiently finding interesting documents. The aim here is to adapt this idea to select diversified answer trees for keyword search over databases. For instance, we may choose T_1 and T_7 in Figure 1.3 since they represent different keyword tuples, and the connection structures are distinct as well.

1.1.3 Social Network Analysis

Social network analysis [71] has emerged as a key technique in modern sociology due to a large and rapidly growing social network companies nowadays, such as Facebook and Twitter. Social network analysis views social relationships in terms of network theory, consisting of nodes (representing individual actors within the network) and ties (which represent relationships between the individuals, such as friendship, kinship, organizational position, sexual relationships, etc.) [95]. One fundamental problem is how to efficiently to identify groups of social actors that are highly connected with each other, represented by a cohesive subgraph, in which analysts may discover interesting structural patterns among social actors, and normal users can know what happening in their neighborhood. Moreover, visual representation of social networks is important to understand the network data and convey the result of the analysis. Many of the analytic software have modules for network visualization. Exploration of the data is done through displaying nodes and ties in various layouts, and attributing colors, size and other advanced properties to nodes. Visual representations of networks may be a powerful method for conveying complex in-

formation. In this thesis, we combine the cohesive subgraph discovery and social network visualization to build a novel system for social network visual analysis.

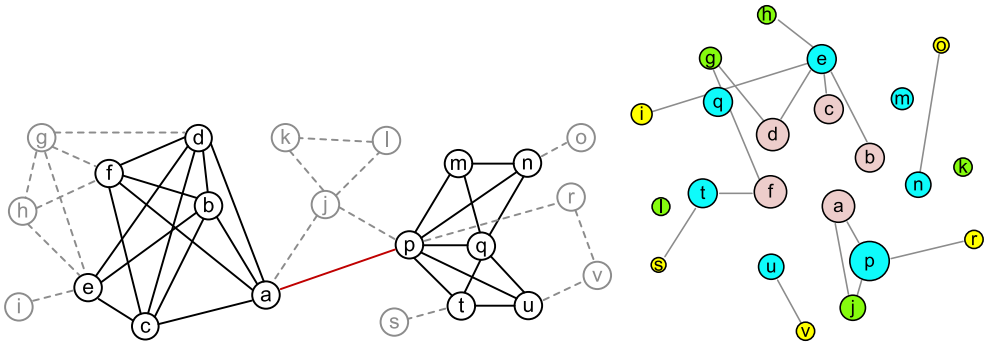
Graphs play a seminal role in social network analysis nowadays. A large and rapidly growing social network companies store social data as graph structures, such as Facebook[3] and Twitter[4]. In a social graph, vertices represent social actors, while edges represent relationships or interactions between actors. One fundamental operation on the social graph is to identify groups of social actors that are highly connected with each other, represented by a cohesive subgraph, in which analysts may discover interesting structural patterns among social actors, and normal users can know what happening in their neighborhood.

Cohesive subgraph discovery is an intriguing problem and has been widely studied for decades. One fundamental structure is the clique in which every pair of vertices is connected. Finding cliques is NP-Hard [45] and many work tries to relax the clique problem to improve efficiency [83, 8, 103, 102, 117, 115]. However, these methods do not directly take the characteristics of social network into consideration. For example, in Figure 1.4a, we emphasize the 3-core in solid edges and connected vertices, in which every vertex v inside it satisfies $d(v) \geq 3$. However, g is not cohesive enough as a whole. Considering cliques inside g, we can find a 5-clique (a, b, c, d, f) and a 4-clique (c, d, e, f) on the left, as well as two 4-cliques $\{(m, n, p, q), (p, q, t, u)\}$ on the right. But vertex a and p are not tightly coupled since they only share one common neighbor j, so the subgraph g is better viewed as two separate cohesive groups.

This phenomenon, denoted as the tie strength concept, is well studied in the sociological area. Note that tie is same as edge in a social graph. Mark Granovetter in his landmark paper [55] indicates that two actors A and B are likely to have many friends in common if they have a strong tie. In another state-of-the-art sociological paper, White et al. [121] observe that a group is cohesive to the extent that pairs of its members have multiple social connections, direct or indirect, but within the group, that pull it together. One intuitive real life example is that you and your intimate friends in Facebook may have a high possibility to share lots of mutual friends. However, this observation has been missing from many of the cohesive subgraph definitions,

[3] https://www.facebook.com
[4] https://www.twitter.com

which drives us to define a "**mutual-friend**" structure to capture the tie strength in a quantitative manner for social network analysis. Assume we consider a tie in Figure 1.4 valid if and only if it is supported by at least two mutual friends. With only supported by one mutual friend j, the tie (a, p) should be disconnected according to the mutual-friend concept, and we successfully separate subgraph g to two groups. We will formally define the problem and compare it to other definitions in details in Chapter 5.



(a) Before Layout (b) After Layout

Figure 1.4: Cohesive Graph Example

1.2 Research Aims

It has recently been asserted that the usability of a database is as important as its capability [67]. The authors study why database systems today are so difficult to use, and identify a set of five pain points in the current database systems. Inspired by this work, the most important objective of this thesis is to improve the usability of the modern database management system.

However, the focus of the database usability paper is on issues in the data model and database design, while the focus of this thesis is the data analysis and data visualization in databases. In general, my research interests span across the whole process of converting data into intelligence, such as the multi-dimensional data in preference mining, structural data in keyword search over databases and graph data in social network analysis. We view data as sources of intelligence and aim to extract knowledge from data and information in an efficient and effective manner so that the

knowledge can be utilized to create intelligent systems with applications in real life problems. To this end, we not only propose new data analyzing problems and design algorithm to efficiently solve them, but also build real systems to support users to browse the analysis results in visualized and interactive manner. The results of my interactive data analytical study should shed light on the database usability that are not available so for.

1.3 Methodology

In contrast to the common sense that we tackle a difficult problem with a "high powered" techniques, in data analysis the real "trick" is to simplify the problem and the best data analyst is the one who gets the job done, and done well, with the most simple methods. The major difficulties for the large scale data analysis in databases are twofold. On one hand, handling the datasets with large cardinality and high dimension is problematic. On the other hand, the result representations are too complex to understand. In this section, we briefly present various key techniques to perform interactive data analysis in databases, and the detailed solutions will be presented in Chapter 3 to Chapter 5 respectively.

To begin with, since we need to deal with large scale database applications, one fundamental strategy is to provide summary view for the complex data analysis results, so that users can understand the result in the broad way. The summarization in this thesis is the approach to extract the most important characteristics of the analyzed data but not the details. It is a simple yet effective approach to many large scale data analyzing problems. There are various approaches to achieve the summarization. Sampling is widely used in statistical analysis because analyzing a well selected subset of data gives similar results to analyzing all of the data. It caters for large scale applications since sampling is a lightweight approach with high efficiency. In data mining, clustering is one common used approach to discover representatives for multi-dimensional datasets. In information retrieval, search results diversification [88] emerges in order to discover relevant but distinguished results to cover more information. Based on the social network data, researchers proposed various metrics to highlight and summarize different aspects for social network analysis.

But data analysis is not about data — it uses them. Even if we could present data in a summary view, we still need to propose an effective approach to help users find what exactly they need in the complex results. Especially when deal with large scale dataset, it is a big challenge to keep the analysis visual intuitive and user controllable, which is very important for users to understand the result and find out what is interesting to investigate. Ranking is one common used strategy to list the results. However, different users have different preferences. Without knowing the data well, it is hard for a user to explicitly express the preference for effective ranking. To solve it, we propose a hierarchical browsing approach to couple with the summarization techniques we discussed above. Hierarchical browsing is an effective approach to interact with users and can be elegantly supported by summarization techniques. By grouping the large result set with respect to the representatives, we enable users to efficiently locate desired results by drilling down to relevant answers incrementally on top of the visual interface instead of a global ranking.

1.4 Contributions

Next, we summarize various topics this thesis contributes towards the interactive data analysis in database area.

Elicit Users' Preference In this work, we address a user preference query on top of multi-dimensional dataset. We propose to elicit the preferred ordering of a user by utilizing skyline objects as the representatives of the possible ordering. With the notion of *order-based representative skylines*, representatives are selected by means of sampling based on the orderings that they represent. To further facilitate preference exploration, a hierarchical clustering algorithm is applied to compute a denogram on the skyline objects. By coupling the hierarchical clustering with visualization techniques, this framework allows users to refine their preference weight settings by browsing the hierarchy.

Diversified Keyword Search in Databases We next apply the hierarchical browsing approach in the application of keyword search in databases. To this end, we implement a novel system allowing users to perform diverse, hierarchical browsing on keyword search results. It partitions the answer trees in the

keyword search results by selecting k diverse representatives from the answer trees, separating the answer trees into k groups based on their similarity to the representatives and then recursively applying the partitioning for each group. By constructing summarized result for the answer trees in each of the k groups, we provide a visual interface for users to quickly locate the results that they desire.

Social Network Visual Analysis We finally introduce a novel subgraph concept to capture the cohesion in social interactions, and propose an I/O efficient approach to discover cohesive subgraphs. Besides, we propose an analytic system which allows users to perform intuitive, visual browsing on a large scale social networks. We hierarchically visualizes the subgraph out on orbital layout, in which more important social actors are located in the center. By summarizing textual interactions between social actors as the tag cloud, we provide a way to quickly locate active social communities and their interactions in a unified view.

Parts of the materials of this thesis on interactive data analysis in preference mining, keyword search in databases and social network analysis were previously published in [132, 134, 133] respectively.

1.5 Outline of the Thesis

The rest of the thesis is organized according to the three topics that we have introduced and the approaches we developed to perform interactive data analysis on these topics. To begin with, we review the literatures in chapter 2 about the data analysis and data visualization techniques, which are the context and the background knowledge for the study in this thesis.

Chapter 3 presents the interactive data analysis in preference mining in database. In chapter 4, we propose the interactive data analysis for keyword search in databases. Next, we tackle the problem of interactive data analysis in social network in chapter 5. For each of the above topics, we first show the motivation and the importance of data analysis in this topic. Then, based on the limitations of interactive data analysis

in each topic, we propose a new problem and describe the methodology we proposed to solve it efficiently. Furthermore, we implement interactive visualization systems to make it user friendly. Last but not the least, we describe the experiments to show the effectiveness and the efficiency of our methods and summarize each work.

Finally, we conclude the whole thesis and indicating the future research directions in chapter 6.

Chapter 2

Literature Review

In recent years, interactive data analytics in databases has been a hot topic in database community. In the following discussions, we first review the general data analysis and data visualization techniques in Section 2.1, which form the foundation of our solutions to interactive data analysis in databases. Then, we classify the related work of interactive data analytics in databases in terms of their similarities/differences with three key topics respectively. In particular, we first review the related work of eliciting users' preference in Section 2.2. Second, we examine how to perform keyword search in databases efficiently in Section 2.3. Third, we investigate the study in social network analysis and social network visualization in Section 2.4.

2.1 Interactive Data Analysis Techniques

We first review the state-of-the-art interactive data analysis techniques that are adopted in or highly related to the solutions in the three key topics in this thesis, according to the introduction in Section 1.3. The first part is about summarization techniques, while the second part is about visualization techniques.

2.1.1 Summarization Techniques

Summarization is the approach to extract the most important characteristics of the analyzed data but not the details, which is a simple yet effective approach to many large scale data analyzing problems. There are various approaches to achieve the summarization. In statistical analysis, sampling is concerned with the selection of a subset of individuals within a statistical population to estimate characteristics of the whole population. It is widely used because its low cost and fast data collection. Sampling methods can be classified as probability methods or nonprobability methods. A probability sampling is one in which every unit in the population has a chance of being selected in the sample, including random sampling [124], systematic sampling [15] and so on. A non-probability sampling is one in which members are selected from the population in some nonrandom manner. These include snowball sampling [53], judgment sampling [36] and so on. The advantage of probability sampling is that sampling error can be calculated, while the degree to which the sample differs from the population remains unknown in nonprobability sampling.

In data mining, clustering is one common used approach to discover representatives for multi-dimensional datasets. It has plenty of variations and can be categorized based on their cluster model, such as connectivity models, connectivity model, density models, subspace models and graph-based models. For example, the k-means algorithm [85] belongs to the connectivity models, which represents each cluster by a single mean vector. DBSCAN [39] and OPTICS [10] defines clusters as connected dense regions in the data space, which belongs to the density models. Since there are so many different models suitable for different applications, many toolkits were developed to help users find the best clustering method for a specific problem. The most widely used one is WEKA [58], which is an open source platform providing a collection of machine learning algorithms for data mining tasks.

Result diversification is emerging data summarization technique where the result consists of a set of objects representing the whole result set or distinguished from each other. In contrast to the ranking query, this query type is useful for users to fast discovering results they are interested in from a large result set, so that it plays an important role in many different contexts nowadays, such as representative skyline finding, search result diversification and so forth. Representative skyline finding is

proposed to solve the too many skyline results in high dimensional space, which we will introduce in the subsequence sections. Search result diversification is a powerful approach to enhance user satisfaction in the IR community [88, 31, 5, 52, 37]. They developed various diversity measures for documents, and effectively solved the diversity problem based on different diversification objectives. However, their diversity measures are designed for documents, so the approaches are not applicable to keyword search in databases with structural answer set.

Based on the social network data, researchers proposed various metrics to highlight and summarize different aspects for social network analysis. In general, these metrics can be divided into three categories. The first category is based on the connections.One example metric belong to this category is homophily [86], which is the tendency of individuals to associate and bond with similar others. The second category is based on the distributions. The most common used one is centrality, which refers to a group of metrics that aim to quantify the "importance" or "influence" of one node within a network [120]. Examples of centrality measures include betweenness centrality [120], degree centrality [93] and so on. The last category is based on the segmentations. For example, the clustering coefficient [59], a measure of the degree to which nodes in a graph tend to cluster together, is one metric belong to this category.

In this thesis, we take advantage of the above summarization techniques and adopt them according to different data analytic problem settings. The detailed explanations will be presented later in independent chapters.

2.1.2 Visualization Techniques

A common approach for making large datasets tractable for interactive exploration is through a browseable hierarchy. Smith et al. [106] grouped and visualized the search results based on the rich categories. Abello et al. [1] described a node-link-based graph visualization that allows clustering and navigation of large graphs. Balzer et al. [13] developed the Voronoi treemaps for the visualization of software metrics. In this thesis, we couple this technique with the summarization techniques to better capture the complex results in an interactive manner. As such, users can better perceive results in an intuitive way and find out the results they desired efficiently.

Recently, researchers have developed a variety of toolkits for facilitating visualization design. Stanford Vis Group devises an outstanding framework named Protovis [18, 63], advocating for declarative, domain specific languages (DSLs) for visualization design. By decoupling specification from execution details, declarative systems allow language users to focus on the specifics of their application domain, while freeing language developers to optimize processing. Similar to Protovis, they further proposed D3 [19] with a declarative framework for mapping data to visual elements. However, unlike Protovis, D3 does not strictly impose a toolkit-specific lexicon of graphical marks. Instead, D3 directly maps data attributes to elements in the document object model (DOM). Inspired by their framework, I will integrate the proposed hierarchical browsing visual analytical system as a toolkit, in order to support flexible customizing the visualization and browsing the result as they need.

2.2 Elicit Users' Preference

2.2.1 Skyline Query

The skyline query was introduced into the database community by Borzsonyi et al. [17]. Given a set of points in a multidimensional space such as a set of digital cameras in the space of price, resolution, and the average user review score, the skyline operator [17] returns the points that are not dominated by any other points in the set. The skyline operator and its efficient computation have received a lot of attention in the database community [17, 74, 29, 98, 72, 94] mainly due to the importance of skyline computation in multi-criteria decision making applications and preference-based query answering. Firstly, we define the skyline query formally.

Given a space S defined by a set of d dimensions $\{D_1, \ldots, D_d\}$ and a dataset D on S, a point $p \in D$ can be represented as $p = (p_1, p_2, \ldots, p_d)$ where every p_i is a value on dimension D_i.

Definition 2.2.1 *Domination*
A point $p \in D$ is said to dominate *another point $q \in D$ on S, denoted by $p \prec q$, if (1) on every dimension $D_i \in S$, $p_i \leq q_i$; and (2) on at least one dimension $D_j \in S$, $p_j < q_j$. For $r, s \in D$, they are said to be* not comparable *if $r \not\prec s$ and $s \not\prec r$.*

Definition 2.2.2 *Skyline Query*
A point $p \in D$ is a skyline *point in S if p is not dominated by any other point $q \in S$. We denote $SL(S)$ as all data points that are not dominated by any other points in S, i.e., $SL(S) = \{p \in S | \nexists q \in S, q \prec p\}$. Skyline query is the process to find $SL(S)$.*

There is extensive research works focus on improving the efficiency of the skyline computation. The efficiency was first improved by Chomicki *et al.*[29] and Godfrey *et al.*[98] significantly by means of sorting. By exploiting index structures, the efficiency of skyline query processing can be further improved. Kossmann *et al.*[72] presented a nearest neighbor search algorithm and Papadias *et al.*[94] proposed a branch-and-bound algorithm (BBS). Both methods are based on R-tree structure [56]. This operator has been studied in the context of distributed systems [12], P2P networks [119, 118], parallel environment [122], data streams [101], microeconomic data analysis [77, 78, 131] and processing queries with minimum communication [129].

The skyline query in different environments is also a hot topic recent years. The operator has been studied in distributed systems[12], P2P networks[119], parallel environment[122] and data streams[101]. Parallel and distributed computational environments post both opportunities and challenges for skyline computation. To address the challenges in skyline computation on distributed data sources, Balke et al. [12] proposed an algorithm for vertically distributed data, i.e., the attribute values of a data point are distributively stored in different data sources. Suppose the values of all data points on an attribute are stored in a data source. Independently a sorted list of each attribute is built. Then, the algorithm continuously probes all dimensions in the preference descending order until it retrieves all dimensions of a data point which is identified as a skyline point immediately. Then, all other data points which have not been accessed in any dimension are filtered out. Such a process continues until all skyline points are retrieved. The method can reduce the number of pairwise comparisons between data points.

Several interesting variations were derived from the concept of skyline query. Spatial Skyline Queries (SSQ) [104] returns the set of data objects that can be the nearest neighbors of any object in a given query set. Formally, given a set of data points P and a set of query points Q, each data point has a number of derived spatial attributes

each of which is the distance from the data point to a query point. An SSQ retrieves those points of P which are not dominated by any other point in P considering their derived spatial attributes. The main difference with the regular skyline query is that this spatial domination depends on the location of the query points Q. SSQ has application in several domains such as emergency response and online maps. In this paper, the authors proposed two algorithms B^2S^2 and VS^2 for static query points and one algorithm, The B^2S^2 can be defined as a special case of BBS algorithm presented in [94]. While BBS is a nice general algorithm, since it has no knowledge of the geometry of the problem space, it is not as efficient as B^2S^2 algorithms for the spatial case. On the other hand, VS^2 algorithm makes use of the Voronoi diagram. The Voronoi diagram can fast retrieval the nearest neighbor in a spatial environment, so the VS^2 algorithm utilizes it to find the candidate objects and discovers all the spatial skyline objects efficiently. Moreover, they presented VCS^2 algorithm for streaming Q whose points change location over time. VCS^2 exploits the pattern of change in Q to avoid unnecessary re-computation of the skyline and hence efficiently perform updates.

The most related variation of skyline query is targeting on the problem of having too many skylines in high dimensional space, which were first highlighted by us in [130, 24, 25] and solutions were proposed in the form of *strong*, *frequent* and *k-dominant* skyline respectively. Subsequently, [80] proposed *representative skylines* where k representative skyline objects must be found such that they together dominate the most objects. From a ranking point of view, this ensures that the representatives will somehow not rank too low since the dominated objects will never rank higher than them with any weight settings. Next, *distance-based representative skylines* [112] grouped the skyline objects into k clusters based on Euclidean distance and the medoid of each cluster is selected as a representative skyline. Spatial proximity, however, does not necessary means similarity in ordering. Two points spatially closer to each other may not rank close since it is sensitive to the ranking function. Besides, it is well known that the distance-based method can never avoid the curse of dimensionality, in the sense that the Euclidean distance of a given skyline object from its nearest and farthest neighbor tends to converge [4]. In contrast, we consider using an order-based approach to solve the too many skylines in high dimensional space in this thesis, in order to apply it to the preference elicitation problem. The order-based approach is robust to the increase in dimensionality, which is

more suitable for high dimensional context.

2.2.2 Preference Elicitation

Preference query is one effective query type in many applications, such as recommendation system, information retrieval and so forth. We will introduce preference elicitation in database area and quantitative preference elicitation area respectively, and indicate a different angle of this work.

Preference discovery and mining have been investigated in the database community recently. Kießling [70] modeled various preference constructors and integrates them into database systems. The framework considers preferences in a multidimensional space. They presented a strict partial orders preference model tailored for database systems. The extensible preference model both unifies and extends existing approaches for non-numerical and numerical ranking and opens the door for a new discipline called preference engineering. Also, their model can easily extend to complex preferences by means of various preference constructors. To better integrate the preference query into database systems, they proposed the Preference SQL and Preference XPATH. Here are some typical examples:

Sample Preference SQL query:
SELECT * FROM used_cars WHERE make = 'Opel'
PREFERRING (category='cabriolet' ELSE category ≠ 'roadster')
AND price AROUND 40000 AND HIGHEST(power)
s AND mileage BETWEEN 20000,30000;
Sample Preference XPATH query:
/CARS/CAR #[(@fuel_economy) HIGHEST AND (@mileage) LOWEST
PRIOR TO (@color) IN ("black", "white") AND (@price) AROUND 10000]#

Based on the preference construction approach aforementioned, Jiang et al. [68] introduced the scenario of mining preferences using superior and inferior examples. That is, in a multidimensional space where the user preferences on some categorical attributes are unknown, from some superior and inferior examples provided by a user, can we learn about the user's preferences on those categorical attributes? To solve this problem, preferences are modeled as skyline relations. The authors focus

on mining minimal (in terms of relation size) finite atomic preference relations. They show that the problem of existence of such relations is NP-complete, and the problem of computing them is NP-hard. They also provide two heuristics for computing such preferences.

Recently, Denis et al. [89] proposed a framework called *p-skylines* which is short for prioritized skylines. They presented two drawbacks of skyline query. One important deficiency of the skyline framework is its inability to represent differences in the relative importance of attributes. Another drawback of the skyline framework is that the size of a skyline may be exponential in the number of attribute preferences involved. Therefore, they proposed the framework called p-skylines which enriches skylines with the notion of attribute importance. It turns out that incorporating relative attribute importance in skylines allows for reduction in the corresponding query result sizes. They proposed an approach to discovering importance relationships of attributes, based on user-selected sets of superior and inferior examples. It is shown that the problem of checking the existence of and the problem of computing an optimal p-skyline preference relation covering a given set of examples are NP-complete and FNP-complete, respectively. However, they restricted the discovery problem (using only superior examples to discover attribute importance), which can be solved efficiently in polynomial time.

These works differ from ours in two ways. First, their main aim is to elicit the preference of categorical values within some categorical attribute domains. Second, they focus on finding unknown **atomic** preferences, i.e. an attribute is either more important, less important or incomparable to other attributes. Our work involves the concept of weighted attributes which can model tradeoffs between the attributes. For example, we can model the fact that a user is willing to take a notebook with a CPU that is 20% slower if 50% more memory is given.

In quantitative preference elicitation [26], the attribute priorities are similarly represented as weight coefficients in numeric utility functions. Given the fact that utility function elicitation over a large amount of outcomes is typically time-consuming and tedious, many preference elicitation systems have made various assumptions concerning preferences structures. The normally applied assumption is additive independence, where the utility of any given outcome can be broken down to the sum of individual attributes. The assumption of independence allows a high-dimensional

utility function to be decomposed into a simple combination of lower dimensional sub-utility functions. Then, it is based on the Analytic Hierarchy Process (AHP) [99] to elicit the weight coefficients. The AHP has been accepted as a robust and flexible decision support tool to solve multi-criteria decision problems. It uses a multi-level hierarchical structure of objectives, criteria, subcriteria, and alternatives. However, this AI methodology adopts the query-answer model based on the attributes of outcomes, and learns the utility function and saves as much of an user's effort as possible. Instead of learning the explicit weight coefficients, our work directly elicits preferred objects by presenting the k representatives to the user, who can quickly browse through these objects and discover the preferred ones through a hierarchical process.

2.2.3 Ranking Related Query

Order information is well studied by the database and data mining communities. There are several kinds of ranking queries highly related to preference mining. We will introduce the ranking aggregation and the order learning respectively.

Rank aggregation addresses the problem of computing a "consensus" ranking of the alternatives, given the individual ranking preferences of several judges. While the philosophical aspects of rank aggregation have been debated extensively during this period, the mathematics of rank aggregation has gained more attention in the last eighty years, and the computational aspects are still within the purview of active research. In computer science, rank aggregation has proved to be a useful and powerful paradigm in several applications including meta-search, combining experts, synthesizing rank functions from multiple indices, biological databases, similarity search, and classification.

In [38], they mainly focused on one important practice of rank aggregation in the web applications. They formulated precisely what it means to compute a good consensus ordering of the alternatives, given several (partial) rankings of the alternatives. Specifically, they identified the method of Kemeny, originally proposed in the context of social choice theory, as an especially desirable approach, since it minimizes the total disagreement between the several input rankings and their aggregation. The definition of Kemeny criterion is as follows:

Definition 2.2.3 *Kemeny Optimal Ranking*
Given n candidates and k permutations of the candidates, $\{\pi_1, \pi_1, \ldots, \pi_k\}$*, a Kemeny optimal ranking of the candidates is the ranking* π *that minimizes a "sum of distances",* $\sum_i^k d(\pi, \pi_i)$*, where* $d(\pi_j, \pi_k)$ *denotes the number of pairs of candidates that are ranked in different orders by* π_j *and* π_k*.*

However, the optimal solutions based on Kemeny's approach is NP-hard, even when the number of rankings to be aggregated is only 4. Therefore, they provided several heuristic algorithms for rank aggregation and evaluated them in the context of Web applications.

Fagin et al. [42, 43, 40, 41] and Ailon et al. [6, 7] solved many challenges for rank aggregation in databases. On one hand, in database-centric applications, we are often interested in only the top few answers of the aggregation. This feature leads to the quest for algorithms that quickly obtain the top result(s) of aggregation, perhaps in sub-linear time, without even having to read each ranking in its entirety. The author in [42] mainly solved the problem how to define reasonable and meaningful distance measures between top k lists. Specifically, they introduced various distance measures between "top k lists", which are "almost" a metric satisfying the a relaxed version of the triangle inequality. On the other hand, while many database attributes are usually numeric, there are attributes that are inherently non-numeric. The number of distinct values in such non-numeric attributes is often very small. Therefore, when one sorts according to values this attribute can take, the resulting rank ordering of the objects is not a permutation any more; it is an ordering with ties, also known as a partial ranking. Thus, one important feature of rank aggregation in database applications is that, due to preference criteria on few-valued attributes, we need to deal with partial rankings rather than full rankings. Motivated by this scenario, Fagin et al. [40, 41] proposed several metrics to compare partial rankings and handle ties, presented algorithms that efficiently compute them, and proved that they are within constant multiples of each other. In [6], they improved constant factor approximation algorithms for aggregation of full rankings and generalized them to partial rankings for all the metric introduced by Fagin. Furthermore, they paid remarkable attention to the more general p-ratings problem, i.e., a mapping from the ground set V to a rank universe U of fixed size p.

Moreover, there are several other important applications of rank aggregation. In

[43], the authors proposed a novel rank aggregation based approach to performing efficient similarity search and classification in high dimensional data. In their approach, a small number of independent ”voters” rank the database elements based on similarity to the query. These rankings are then combined by a highly efficient aggregation algorithm. On the theoretical side, this method has a high probability to produce a result that is a $(1+\epsilon)$-factor approximation to the Euclidean nearest neighbor. On the practical side, it turns out to be extremely efficient with sorted access to a small portion of data. In [7], the authors extended the idea of rank aggregation to clustering. Consensus clustering or ensemble clustering is the problem of integrating possibly contradictory clusterings from existing data sets into a single representative clustering. This problem can be applied to remove noise and incongruities from data sets or combine information from multiple classifiers. In this paper, the authors provided an unified method to approximately solve the ranking aggregation and consensus clustering efficiently.

Learning to rank is a new and popular topic in machine learning. In [32], an algorithm was developed to learn a linear preference function. In this algorithm, feedbacks are iteratively given by users in the form of "p is preferred to q" and the weights are iteratively adjusted based on the feedbacks. In specific, they developed the following two-stage approach to learning how to order. In stage one, they learn a preference function, a two-argument function PREF(u, v) which returns a numerical measure of how certain it is that u should be ranked before v. In stage two, they use the learned preference function to order a set of new instances U; to accomplish this, they evaluate the learned function PREF(u, v) on all pairs of instances $u, v \in U$, and choose an ordering of U that agrees, as much as possible, with these pairwise preference judgments. However, finding a total order that agrees best with a preference function is NP-complete, so they described a simple greedy algorithm that is guaranteed to find a 2-approximation result. In another paper [47], they introduced and studied an efficient learning algorithm called RankBoost for combining multiple rankings or preferences. This algorithm is based on AdaBoost algorithm and its recent successor developed by Cohen et al. [32]. The algorithm they presented uses a similar framework, but avoids the intractability problems. Furthermore, as opposed to the on-line algorithm, RankBoost is more appropriate for batch settings where there is enough time to find a good combination. Thus, the two approaches complement each other.

There is another way to learning the rank based on probability models. In [22], Burges et al. investigated gradient descent methods for learning ranking functions; They proposed a simple probabilistic cost function, and introduced RankNet, an implementation of these ideas using a neural network to model the underlying ranking function. They employed gradient descent as an algorithm to train the neural network model. Zhe et al. [23] proposed the listwise approach, in which document lists instead of document pairs are used as instances in learning. Likewise, they utilized a probabilistic method to calculate the listwise loss function. Specifically they transformed both the scores of the documents assigned by a ranking function and the explicit or implicit judgments of the documents given by humans into probability distributions. Then, the ListNet was proposed using the listwise loss function, with neural network as a model and gradient descent as an algorithm. This method further improved the quality of the ranking function. This approach is different with ours in that we focus on skylines as a representative of orders and provide a hierarchical visualization framework to elicit the preference of users systematically [1].

2.3 Diversified Keyword Search in Databases

2.3.1 Keyword Search in Databases

Keyword search in databases is a convenient and effective approach in information retrieval, without the need for users to know the underlying data schema and query language. This technique has been widely applied in various domains, including web documents, relational database, XML documents, and graph databases. Current approaches can be classified into two categories: schema-based ones and graph-based ones. The schema-based methods [64] generated join expressions based on database schema and produced the resulting tuple trees through SQL queries. The graph-based approaches [3, 69] materialized the database as a graph in which each node corresponds to a tuple. They discovered compacted substructures based on heuristic graph search. Many of recent works [65, 81] developed different ranking strategies in order to improve the search effectiveness. Hristidis et al. [65] adapted

[1] In many ways, our approach is similar to how we judge the results of a search engine; We conclude that the search ranking is useful if the first few results are good.

IR-style document-relevance ranking strategies to the problem of processing free-form keyword queries over RDBMSs. Liu et al. [81] further propose a sophisticated IR style ranking strategies with four new factors that are critical to the problem of search effectiveness in relational databases.

Among all these data models, the graph-structured model is among the most well accepted as it is rather general and can even model unstructured, semi-structured and structured data together in one graph [79]. Given a set of query keywords, most of the existing keyword search systems [69] aim to find minimal connected trees that contain all the keywords, which is in essentially the Steiner tree problem. It has the advantage of ensuring the tightness of the result so that the keywords are closely related. Such an approach however can have two drawbacks. First, some interesting information may be missed due to the *minimal* property. Missing nodes and edges that are not included in the result could contain interesting information although they do not contain any keywords. Second, when querying frequent keywords, large number of result trees could be return with a large amount of overlaps in nodes and edges between these trees. To tackle the two problems, [96] proposed to find communities with a center node, where the distance from all the keyword nodes to the center is within a threshold radius. This however have its own drawback. First, a suitable radius is difficult to tune. Second, it is not reasonable to treat all keyword nodes equally important within the same radius; Moreover, the community structure may be hardly interpretable for some complex graph structure.

2.3.2 Result Diversification in Databases

Database researchers studied the result diversification recently. Yu et al. [125] introduced the notion of explanation-based diversity in recommendation systems. Vee et al. [113] diversified the query results by applying an inverted-list algorithm. Liu et al. [82] developed a feature selection algorithm in order to highlight the differences among structural XML data.

There are three recent works considering keyword search diversity in relational databases. Golenberg et al. [51] and Stefanidis et al. [109] studied the answer tree diversification, while DivQ [35] solved another problem to discover diversified schemas. However, their diversity measures are all derived from the Jaccard distance, which fails

to capture both textual information and structural information as shown in the experiments. Instead, we proposed a novel kernel distance suitable for structural answers. Furthermore, we developed an interactive system which allows users to diversely, hierarchically browse the whole answer set instead of top-*k* of them.

2.4 Social Network Visual Analysis

2.4.1 Social Network Analysis

Modeling a cohesive subgraph mathematically has been extensively studied for decades. One of the earliest graph models was the clique model [84], in which there exists an edge between any two vertices. However, the clique model idealizes cohesive properties so that it seldom exists and is hard to compute. Alternative approaches are suggested that essentially relaxes the clique definition in different aspects. Luce [83] introduces a distance based model called *k*-clique and Alba [8] introduces a diameter based model called *k*-club. Although these models relax the reachability among vertices from 1 to *k*, one limitation of these works is they are still NP-complete which cannot be applied to large social graphs. Another line of work focuses on a degree based model, like *k*-plex [103] and *k*-core [102]. The *k*-plex is still NP-Complete since it restricts the subgraph size, while *k*-core further relaxes it to achieve the linear time complexity with respect to the number of edges. However, the *k*-core definition is too loose to capture the cohesive structure of the social graphs. A new direction based on the edge triangle model, like DN-Graph [117] and truss decomposition [115], is more suitable for social network analysis since it captures the tie strength between actors inside the subgroup. Our proposed mutual friend concept belongs to this model and we will compare it with the above two concepts in Chapter 5 in details. Recently, database researchers have tried to scale up the disk based cohesive subgraph discovery. Cheng et al. [27] propose a partition based solution for massive *k*-core mining. They also develop a disk based triangulation method [30] as a fundamental operation for cohesive subgraph discovery. In this research, we store the social graph in graph database that is more scalable for graph traversal-based algorithms.

Community detection is another approach to discover a group of people in addition to the dense subgraph discovery. Leskovec et al. [75] summarize the state-of-the art community detection methods and compare them empirically. Typical approach of community detection is to choose an objective function that captures a set of vertices with better internal connectivity than external connectivity, such as betweenness centrality [50] and modularity [91]. The goal of community detection approach is close to cohesive subgraph discovery: discover the nodes of the network that can be easily grouped into sets of nodes such that each set of nodes is densely connected internally. However, they deal with the problem from different angles. The community detection is like the clustering approach on the nodes of the network. As such, they are concerned with how to define a better objective function to determine whether nodes belonging to one community or not. On the other hand, the cohesive subgraph discovery views the subgraph as a whole, i.e. try to find the subgraphs that satisfy certain properties. Our social network analysis belongs to the latter category, in which we find all the subgraphs in the social graph with the *k*-mutual-friend property.

In addition, social network characteristics has been well investigated in sociology communities. The most related one is the tie strength theory, which is introduced by Mark Granovetter in his landmark paper [55]. Recently, many social network researchers investigate this important theory in online social network, such as the user behaviors in Facebook [49, 11] and Twitter [54]. Their conclusions show that the strength of tie, which is the basis of the mutual-friend subgraph definition in this thesis, is still a tenable theory in social media. However, to the best of our knowledge, there is no previous work makes use of this theory to discovery cohesive subgraphs in social network analysis.

2.4.2 Social Network Visualization

After discovering cohesive subgraphs, how to visually represent these subgraphs is another important component of this research. Graph structure visualization and analysis has received a great deal of attention from both sociology and computer science communities. Freeman [46] summarizes the use of graphic imaging in social network analysis from the sociology perspective. Researches from the computer science perspective put more efforts into the graph representation and exploration of

social networks. Wang et al. [116] proposes a linear plot based on graph traversal to capture the dense subgraph distribution in the whole graph. Zhang et al. [128] extends it to compare the pattern changing between two graph snapshots. Another approach of placing vertices in concentric circles with different levels is a popular way to visualize graph structures, such as k shell decomposition [9], centralities visualization [33] and so on. We leverage the circular idea and devise the orbital layout to visualize k-mutual-friend subgraphs in an interactive manner. In our method, the orbital layout is perpendicular to linear plot. Using the approach proposed by Wang et al., linear plot for global subgraph distribution and the orbital layout for local subgraph representation could be seamlessly integrated. Moreover, Arnetminer [111] provides comprehensive search and mining services for academic social networks. It is a full fledged framework with nice visual exploring the function like the relationship graph between two researchers. However, the focus in Arnetminer is to show the connections between two researchers. More information along the importance of individuals in the cohesive subgraphs needs to be uncovered.

Chapter 3

Hierarchically Elicit Users' Preference

3.1 Overview

In this chapter, we propose to elicit the preferred ordering of a user by utilizing skyline objects as representatives of the possible ordering. Our approach tries to find k representative skylines that best capture the orderings that are associated with other skyline objects. This brings about two challenges:

1. Given a dataset D, let $W_p(D)$ denote the set of weight settings such that for every $w \in W_p$, $\pi_w(D)[1] = p$, $p \in D$. In this case, $W_p(D)$ is a set of weight settings in which the object p will be ranked first and such a set could potentially have infinite memberships. As such, comparing the ordering represented by two skyline objects becomes difficult.

2. Given that $\pi_w(D)$ represents a ranking with a large number of objects, comparing any instance of the rankings represented by two skyline objects will require computationally efficient solutions to be developed.

In order to overcome these problems, we propose an indirect notion of similarity between the orderings that are represented by two skyline objects p and q. We claim

that the ordering of q is close to p if q has a high probability of ranking high whenever p is ranked first in the ordering. Based on this notion which we will formally define later, we make various contributions towards eliciting users' preference based on hierarchical browsing of skylines:

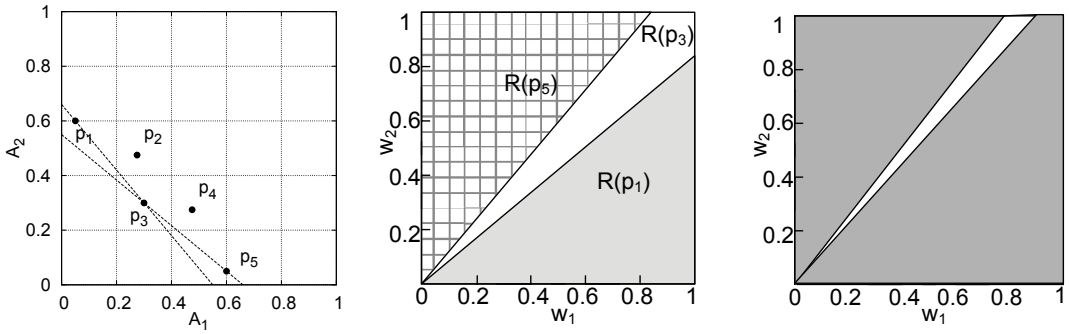
- We introduce the notion of **order-based representative skylines** which selects representative skylines based on the ordering that they represent. Unlike previous work, we bring the preference function back into the picture when determining representative skylines since our aim is to elicit the preference of the user based on these representatives.
- To handle the two problems that we presented earlier, we define a notion of similarity that avoids explicit comparison of the orderings that are represented by two skyline objects. Based on this similarity measure, we develop sampling techniques that allow us to efficiently and accurately estimate the similarity between any two skyline objects. The similarity measure also allows us to define a goodness measure for clustering skyline objects, and a k-partitioning clustering algorithm is developed to cluster skyline objects based on this goodness measure.
- By applying the k-partitioning algorithm recursively, we create a hierarchical clustering of the skyline objects. By coupling hierarchical clustering with visualization techniques, we enable users to refine their preference weight settings by browsing the hierarchy.
- We conducted extensive experiments, and the results show that our approach is both effective and efficient.

The remainder of this chapter is organized as follows. Section 3.2 gives our new definition of representative skylines and shows the defects of existing methods. Section 3.3 presents the efficient sampling algorithm, and hierarchical browsing to elicit users' preference is described in Section 3.4. Results of our extensive experimental study are reported in Section 3.5. Finally, we summarize this chapter in Section 3.6.

3.2 Preliminary

3.2.1 Problem Definition

We have a database D of n objects. Each object is described by d attributes A_1, ..., A_d. We will use $p.A_i$ to refer to the value of an attribute A_i for an object p. For ease of discussion, we assume that all of these attributes are numerical attributes ranging from 0 to 1 [1] and that a smaller value indicates better score. As such, we say p dominates q if $p.A_i < q.A_i$ for at least one value of i and $p.A_i \leq q.A_i$ for $1 \leq i \leq d$. The skyline set $S \subseteq D$ consists of all objects in D which are not dominated by any other objects in D. We also have a monotonic ranking (or preference) function $P(\cdot)$ which is provided by the application domain and users will specify their preference by providing a set of weights $w = \{w_1, w_2, \ldots, w_d\}$, $0 \leq w_i \leq 1$. Given set of weights, the user can easily define any monotonic ranking function as $P(\vec{w}, \overrightarrow{f(\cdot)}) =< \vec{w}, \overrightarrow{f(\cdot)} >$, i.e. the dot product of weight vector $\vec{w}(w_1, w_2, \ldots, w_d)$ and monotonic function vector $\overrightarrow{f(\cdot)}(f_1(\cdot), f_2(\cdot), \ldots, f_d(\cdot))$. $f_i(\cdot)$ can be any monotonic function on objects, such as linear function, product function or exponential function.



(a) Data Space (b) Weight Space (c) $R_4(p_2)$ on Weight Space

Figure 3.1: Example of Data Space and Weight Space

Given the above setting, we deals with two multi-dimensional spaces. First, we have the **data space**, which is the d-dimensional space that is formed from A_1, ..., A_d. Second, we have the **weight space**, which is another d-dimensional space formed from w_1,...,w_d, i.e. the i^{th} dimension of this space represents the weight w_i. Any

[1] This can be obtained by mapping the attribute values of some application domain to a score from 0 to 1

point in the weight space thus corresponds to a particular setting of the weights. For any skyline object $p \in S$, we will use $R(p)$ to refer to the region in the weight space such that $\pi_w(D)[1] = p$ as long as w is within the region $R(p)$. Since the weight space is normalized to the unit range, we can treat the volume of $R(p)$ as a probability that p is the top object in any possible ordering. Figure 3.1 illustrates the data space and weight space for a set of skyline objects when the preference function is a simple dot linear product between the weights and the attribute values. As can be seen in Figure 3.1(a), p_1, ..., p_5 are all skyline objects since they do not dominate each other. However, if we look at the weight space in Figure 3.1(b), we can see only $R(p_1)$, $R(p_3)$ and $R(p_5)$ since based on the dot linear product preference function, p_2 and p_4 can never be ranked first regardless of the weight setting.

If we use $V(\cdot)$ as a function that calculates the volume of any given region, then the probability of p being a top object will be denoted as $V(R(p))$. To generalize this further, we will use $R_m(p)$ to denote the region in the weight space such that p is among the top-m objects when compared to other skyline objects. We are now ready to define a similarity measurement between two skyline objects p and q.

Definition 3.2.1 $\mathcal{SIM}_m(p, q)$
Given $p, q \in S$ and m, we measure how well p can represent q by

$$\mathcal{SIM}_m(p, q) = V(R(p) \cap R_m(q))/V(R(p))$$

It is easy to see that $\mathcal{SIM}_m(p, q)$ is in fact the probability that q is within the top-m skyline objects whenever p is ranked first. Intuitively, we are saying that if p has orderings that are very similar to q, then $\mathcal{SIM}_m(p, q)$ will be high and thus p can represent q well. Note that $\mathcal{SIM}_m(p, q)$ is in fact not a metric since it is not symmetric and also does not follow triangular inequality. This however does not affect our k-partitioning clustering algorithm. Unlike most clustering applications in which members in the same cluster must be similar, our sole aim here is that the representatives of each cluster can represent its members accurately while other members in the cluster need not be similar to each other.

Given S, our aim is to select a k representative set K, such that $K \subseteq S$, $|K| = k$ and other non-representative skylines are somehow represented by K in terms of

the ordering that they represent. Intuitively, K should satisfy two criteria. First, its $\bigcup_{p \in K} V(R(p))$ should cover a sufficiently large region of the weight space so that all possible rankings are covered as much as possible. Second, K should somehow represent other skylines that are not within K.

The first criteria is relatively easier to satisfy with the observation that $\{R(p) \cap R(q) = \emptyset, p, q \in S \wedge p \neq q\}$. Since there are no overlap between the regions, it is enough to ensure that $V(R(p))$ is sufficiently large for each $p \in K$ so that $\bigcup_{p \in K} V(R(p))$ is large. For the second criteria, we will propose a measure of goodness.

Definition 3.2.2 *Quality*(K, S)

$$Quality(K, S) = \frac{\sum_{q \in S} \max_{p \in K} \mathcal{SIM}_m(p, q)}{|S|}$$

As can be seen, *Quality*(K, S) is a goodness measure that is similar to those used in a k-partitioning algorithm, i.e. the average similarity between each skyline object and its best representative. Correspondingly, we define our order-based representative skyline problem as follow:

Definition 3.2.3 *Order-based Representative Skylines*
Given S, p, m and threshold α, find a set of representative $K \subseteq S$ such that:

1. *For each $p \in K$, $V(R(p)) \geq \alpha$.*

2. *Quality*(K, S) *is maximized.*

3.2.2 Problem Analysis

Note that like all clustering problem, finding the optimal order-based representative skylines is a NP-hard problem.

Lemma 3.2.1 *Finding K with k skyline objects maximizing Quality*(K, S) *is NP-hard.*

Proof 3.2.1 (sketch) *We construct a polynomial reduction from one NP-hard problem: the decision version of vertex cover problem. Given the undirected graph $G(V, E)$, if there exists one edge e between node p and node q, $\mathcal{SIM}_m(p, q)$ and $\mathcal{SIM}_m(q, p)$ are set to 1. Moreover, $\mathcal{SIM}_m(p, p), p \in V$ are all set to 1 since the node p covers itself. Other $\mathcal{SIM}_m(p, q)$ are all set to 0. If we find an optimal set of order-based representative skylines K making Quality$(K, S) = 1$, K is the set of k nodes covering the graph G. This completes the polynomial reduction.*

While the proof for Lemma 3.2.1 assumes that the $R(p)$ and $R_m(p)$ with their corresponding volumes can be easily computed, we will show here that this is not the case. First, one important property about the $R(p)$ is presented in the following lemma.

Lemma 3.2.2 *For any skyline object p, $R(p)$ is either empty or a convex polytope.*

Proof 3.2.2 *Based on linear programming theory, if p is the i^{th} object p_i in the skyline, the computation of $R(p_i)$ can be directly transformed to the satisfaction of the following inequations:*

$$\begin{cases} P(\vec{w}, \overrightarrow{f(p_i)}) \leq P(\vec{w}, \overrightarrow{f(p_1)}) \\ \dots \\ P(\vec{w}, \overrightarrow{f(p_i)}) \leq P(\vec{w}, \overrightarrow{f(p_{i-1})}) \\ P(\vec{w}, \overrightarrow{f(p_i)}) \leq P(\vec{w}, \overrightarrow{f(p_{i+1})}) \\ \dots \\ P(\vec{w}, \overrightarrow{f(p_i)}) \leq P(\vec{w}, \overrightarrow{f(p_n)}) \\ w_i \in [0, 1] \end{cases}$$

The above inequations are a set of linear constraints on the weight space, because each $P(\cdot)$ is the linear function with respect to weight vector $\vec{w}$ given the $\overrightarrow{f(\cdot)}$ and the attributes for each $p \in S$. Therefore, computing the $R((p))$ is equivalent to solving the feasible range of linear constraints. The boundary theory of linear programming[100] proves that each inequality specifies a half space in an n-dimensional Euclidean space, and their intersection is the set of all feasible values the variables can take. The region is either empty, unbounded, or a convex polytope. In our case,

the region is either empty, or a convex polytope, because it is bounded by weight space with $w_i \in [0, 1]$.

According to Lemma 3.2.2, these regions can be determined by computing their boundaries. Ideally, we first discover vertices of $R(p)$ or $R_m(p)$ for each skyline object p, and then derive $\mathcal{SIM}_m(p, q)$. However, the cost of this method is too expensive. For a convex polytope, there are at most $(u!)/(v!(u - v)!)$ vertices, where u is the number of inequations and v is number of variables[100]. Accordingly, the $R_m(p)$ can also be viewed as a union of all possible combinations of linear constraints that p is smaller than at least $|S| - m$ skyline objects. As illustrated in Figure 3.1(c), the shaded region, which is $R_4(p_2)$, is the union of two separate parts. This simple example shows that the computation of $R_m(p)$ is much more complicated than the computation of $R(p)$ in general. Therefore, we conclude that finding the exact boundary of top region and top-m region is unrealistic.

3.3 Methodology

According to earlier analysis, finding the exact regions for $R(p)$ and $R_m(p)$ for all $p \in S$ can be very computationally intensive. Since we are only interested in $V(R(p))$ and $V(R_m(q) \cap R(p))$, we can adopt a sampling approach to estimate these values. This is done by performing a uniform sampling in the weight space and generating a set of weight settings W. For each $w \in W$, we find $\pi_w[i]$ for $1 \leq i \leq m$ and keep a count on the occurrences of the skyline objects. Once the sampling is complete, we can simply estimate $V(R(p))$ by $count(\{w|w \in W, \pi_w[1] = p]\}/|W|$, i.e. the number of instances w in which p is ranked top and divide it by $|W|$. Likewise, $V(R_m(q) \cap R(p))$ is $count(\{w|w \in W, \pi_w[1] = p, \pi_w[i] = q, i \leq m\})/|W|$, the number of instances in which q is ranked among top-m whenever p is the top object and normalize it by $|W|$. Finally, we obtain the following formula according to the definition 3.2.1:

$$\mathcal{SIM}_m(p, q) = \frac{count(\{w|w \in W, \pi_w[1] = p, \pi_w[i] = q, i \leq m\})}{count(\{w|w \in W, \pi_w[1] = p]\}}$$

There are two remaining issues. First, we need to ensure that generating these samples is efficient. Second, we need to ensure that our estimation based on these samplings have certain accuracy. We will address these two issues in the next two subsections. Once these issues are resolved, we will then move on to present our clustering algorithm based on our measure of similarity.

3.3.1 Generating Samples

Instead of computing the ordering for individual samples, we conduct the sampling in batches and apply the TA algorithm[44] concurrently for all samples within the same batch. Assuming that the main memory can handle b samples and we want to have a total of s samples, then the TA algorithm will be applied $\lceil s/b \rceil$ times.

The sampling method is shown in Algorithm 1. For all the sample weight settings, it only needs to discover top-m skyline objects from the disk once using the TA algorithm. Here, m is set to be (number of skyline objects)/k based on the assumption that the skyline objects have uniform probability of appearing in the top-m list of any of the (eventual) k representative objects and thus setting m to this value ensures that each of the objects has a non-zero probability of appearing in the top-m list of one of the k representatives. This m can then be fixed for processing future batches of samples.

In order to perform TA algorithm, we further need to store d sorted lists in the disk. In τ_i, the skyline objects are sorted from the smallest to the largest based on the values on dimension i. Because the score function is monotonic, we perform sorted access and random access on d ranking lists to find the top-m skyline objects efficiently. According to these top-m lists, we can calculate the $V(R(p))$ and $V(R(p) \cap R_m(q))$ and derive $\mathcal{SIM}_m(p,q)$ for every $p, q \in S$.

Intuitively, the approximation of region computation has high precision based on random uniform sampling if the sampling size is sufficiently large. Thus, we approximately achieve region computation as well as derive $\mathcal{SIM}_m(p,q)$ according to definition 3.2.1. As in Line 12-15 in Algorithm 1, we only keep in memory the

counting information of skyline objects which can be the top object. Since the skyline set could be too large to fit in the memory, this strategy greatly reduce the memory consumption. In addition, after performing the TA based algorithm, the batch of samples can be safely discarded to free up the memory for the next batch. Let the top object set T_0 satisfy $\{T_0|p$, if $p \in S$ and $V(R(p)) > 0\}$. Accordingly, T_α is defined as $\{T_\alpha|p$, if $p \in S$ and $V(R(p)) > \alpha\}$. Assume the skyline set size is n, Algorithm 1 utilizes $O(|T_0|n)$ instead of $O(n^2)$. $|T_0|$ is determined by the monotonic function, which is much smaller than n. Therefore, this improves the scalability of our algorithm. Besides the probability information, the regions of $p \in T_0$ defined below are incrementally updated based on samples. These information is critical for hierarchical processing in Section 3.4.

Definition 3.3.1 *Object Coverage*
Given $\{W|$*weight setting* $w \in W$ *if* $\pi_w[1] = p\}$*, the coverage of* p *is the minimal bounding rectangle(MBR) of* W *on weight space.*

The *MBR* for the object p is the minimal bounding rectangle that encloses all the $w \in W$ whenever $\pi_w[1] = p$. However, to determine a sufficient number of samples is challenging. The sample space is infinite and the definition of sufficiency is unclear. Before finding the k representative skylines, we first show what is the quantitative relationship between sampling size and sampling accuracy and how to calculate the sampling size s in Line 2 of Algorithm 1.

3.3.2 The Analysis of Sampling Accuracy

The sampling size determines the tradeoff between accuracy and efficiency. Intuitively, we expect the approximations of $V(R(p))$ and $\mathcal{SIM}_m(p,q)$ to be close to the accurate values if the values are larger than certain thresholds. The constraint of $V(R(p))$ refers to definition 3.2.1. Furthermore, $\mathcal{SIM}_m(p,q)$ should be accurate if it is no smaller than the user-defined threshold β. Taking these two thresholds into consideration, we can derive the following bound for $V(R(p) \cap R_m(p))$:

$$V(R(p) \cap R_m(p)) = V(R(p)) \cdot \mathcal{SIM}_m(p,q) > \alpha\beta$$

Algorithm 1: **SamplingTopM**

Input: # representatives k and # samples b
Output: $2d$ array $\mathcal{SIM}_m$ to store $\mathcal{SIM}_m(p, q)$

$m \longleftarrow$ # skyline objects/k
Calculate the required sampling size s
while $s > 0$ **do**
    Generate next b random uniform samples W
    $s \longleftarrow s - b$
    `// TA based method for b samples`
    while $score_i(m^{th}$ *item on* $heap_i) > \delta_i, i \in [1, w]$ **do**
        Round-robin sorted access on $\tau_1, \ldots, \tau_d$
        Update thresholds $\theta_1, \ldots, \theta_b$ for each sample
        Random access to get next skyline object p
        if $score_i(p) < score_i(m^{th}$ *item on* $heap_i)$ **then**
            Swap p with m^{th} item on heap_i
    foreach *skyline object q in top-m list when p is the top object* **do**
        if $p \notin \mathcal{SIM}_m$ **then** new array $\mathcal{SIM}_m[p]$
        Update the region for $R(p)$
        $\mathcal{SIM}_m[p][q]$++
foreach *skyline object* $p \in \mathcal{SIM}_m$ **do**
    $Count[p] \longleftarrow count(\{w | w \in W, \pi_w[1] = p\})$
    foreach *skyline object q* **do**
        `// calculate the probablity`
        $\mathcal{SIM}_m[p][q] \longleftarrow \frac{\mathcal{SIM}_m[p][q]}{Count[p]}$
return $\mathcal{SIM}_m$;

Therefore, we will focus on the accuracy of $V(R(p) \cap R_m(p))$ to satisfy the required sampling quality. Next, we provide guidelines for the choice of sampling size using statistical analysis.

Let $V(R(p) \cap R_m(p))$ be the unknown value that we are trying to estimate. For simplicity, we utilize probability P representing $V(R(p) \cap R_m(p))$ to do the analysis. Then, the $\overline{P}$ stands for the complementary set of $V(R(p) \cap R_m(p))$. Assume that we have N samples and find that $X = \tilde{P}N$ of these samples satisfy q does not appear in the top-m lists when p is the top object. Given a sufficiently large number of samples, we expect $\overline{P}$ to be close to $\tilde{P}$ as much as possible. Furthermore, to ensure sufficiently large coverage, P should be larger than or equal to $\alpha\beta$, which is equivalent to $\overline{P} \leq 1 - \alpha\beta$. We formally express the problem as follows.

Problem 1 *Given thresholds α, β, confidence interval $1-\gamma$ and margin of error δ, how to determine the sampling size N to ensure*

$$Pr(\overline{P} \in [\tilde{P}-\delta, \tilde{P}+\delta]) > 1-\gamma$$

when $\overline{P} \leq 1-\alpha\beta$.

Obviously, we want both the interval size 2δ and the error probability γ to be as small as possible. Since the sampling process can be viewed as the Bernoulli Trials on the weight space, $X = \tilde{P}N$ satisfies a binomial distribution with parameters N and $\overline{P}$. Therefore, we can apply Chernoff bounds[57] to compute

$$\begin{aligned} & Pr(\overline{P} \notin [\tilde{P}-\delta, \tilde{P}+\delta]) \\ = \; & Pr(X < N\overline{P}(1-\delta/\overline{P})) + Pr(X > N\overline{P}(1+\delta/\overline{P})) \\ < \; & e^{-N\overline{P}(\delta/\overline{P})^2/2} + e^{-N\overline{P}(\delta/\overline{P})^2/3} \end{aligned}$$

The bound in above equation is meaningless if the value of $\overline{P}$ is unknown. A simple relaxation is based on the fact that $\overline{P} \leq 1-\alpha\beta$, yielding

$$Pr(\overline{P} \notin [\tilde{P}-\delta, \tilde{P}+\delta]) < e^{-N\delta^2/2(1-\alpha\beta)} + e^{-N\delta^2/3(1-\alpha\beta)}$$

Setting $\gamma = e^{-N\delta^2/2(1-\alpha\beta)} + e^{-N\delta^2/3(1-\alpha\beta)}$, we obtain the tradeoff between these parameters. Given the requirements on α, β, γ and δ, the above equation calculates the minimum number of sampling size N to guarantee the sampling accuracy.

3.3.3 Finding Order-based Representative Skylines

Since the $\mathcal{SIM}_m(p,q)$ has already been calculated approximately, the next step is to discover the order-based representative skylines. Our goal is to maximize the quality of the representative set K as well as cover a sufficiently large size of the weight space. As the proof in Section 3.2, this problem is NP-hard, thus we adopt the k-medoids clustering algorithm, as presented in Algorithm 2, to efficiently solve the

problem. By partitioning the skyline objects into groups, we choose the medoid as representative for each group and other skyline objects are assigned to the closest representative.

The *k*-medoids clustering is derived from CLARANS [92], which is the state-of-art *k*-medoids clustering inspired by local search idea. One noticeable difference of Algorithm 2 is the filtering method. As in line 4, it sifts out candidate set C as T_α. The default setting of α is $1/|S|$, the average volume of $R(p)$ for $p \in S$ on weight space. This is reasonable since the volume of representative skylines should be at least no worse than the average situation, otherwise the objects can be safely pruned. User also has the flexibility to adjust the threshold in order to achieve the tradeoff between the importance of K and the quality to represent other skyline objects. This is not only beneficial to finding better *k* representative skylines, but also further reducing the candidate size, especially for skyline objects with skew top region sizes. Moreover, the *swapcost* is the difference between $Quality(K, S)$ and $Quality(K', S)$. Line 13 guarantees that the cluster K is updated only if the new cluster K' has better quality. Finally, the clustering algorithm finds *numlocal* *k*-medoids sets with local best quality, and chooses the best of them as the final order-based *k* representative skylines.

3.4 Eliciting Users' Preference

In this section, we further extend our work to support skyline browsing and visualization in order to elicit users' preference effectively and efficiently. In general, it is a hierarchical navigation approach to locate user's preferred region on the weight space. A visual interface is developed to support this exploration.

3.4.1 Hierarchical Browsing

Hierarchical browsing is an effective way to interact with the user. As shown in Algorithm 3, this process can be viewed as iterative refinements based on a combination of sampling and clustering. First, shown with the initial *k* representatives, the

Algorithm 2: **FilterClustering**

Input: # representatives k,threshold α and $\mathcal{SIM}_m$
Output: k order-based representative skylines

Candidate set $C \longleftarrow \emptyset$
k-medoids set $K \longleftarrow \emptyset$
foreach *skyline objects* $p \in T_0$ **do**
    if $V(p) > \alpha$ **then** $C \longleftarrow C \cup p$
// setting according to paper [92]
maxneighbor $\longleftarrow \max(250, k \times (|C| - k) \times 1.25\%)$
numlocal $\longleftarrow 2$
bestquality $\longleftarrow 0$, *bestcluster* $\longleftarrow \emptyset$
for $i = 1$ *to numlocal* **do**
    $K \longleftarrow$ randomly choose k objects from $|C|$
    for $j = 1$ *to maxneighbor* **do**
        Randomly select p from K and q from $C - K$ to swap
        Calculate *swapcost* using p
        if *swapcost* < 0 **then**
            $j \longleftarrow 1$, update K
    if *Quality*$(K, S) >$ *bestquality* **then**
        bestrepresentative $\longleftarrow K$
        bestquality $\longleftarrow$ *Quality*(K, S)
return *bestrepresentative*

users will then select a subset of them as object/s of interested. Second, re-sampling is performed on the region covered by the subset and related clusters. This focused sampling will allow us to have more accurate sampling result on the area of interest. We first define the **cluster coverage** as follows.

Definition 3.4.1 *Cluster Coverage*
Given a cluster c, the cluster coverage r_c of c on the weight space is the minimal bounding rectangle(MBR) that bounds the object coverage of all skyline objects in c.

Therefore, the area of interest is the *MBR* covering all the clusters generated by the selected skyline subset. The clustering algorithm will then be applied on the new samples so that the next level of k representatives can be found. This procedure will iterate until the user reaches the skyline object that is of interest to him/her or when

$|T_0|$ is smaller than k. As shown in Line 5 of the algorithm, the hierarchical process terminates and displays the final results to the user if one of the above two conditions is satisfied.

Algorithm 3: **HierarchicalBrowsing**

```
   Input: w, k, α, SIM_m
1  p ⟵ SamplingTopM(w)
2  K ⟵ FilterClustering(k, α, SIM_m)
3  Output K to user
4  User chooses interesting subset H and sets k
5  while H ≠ ∅ and |T_0| ≥ k do
6      sampleregion ⟵ ∅
7      Candidate set C ⟵ ∅
8      foreach object p ∈ H do
9          Calculate cluster region r_c from the cluster c with medoid p
10         Update sampleregion covering the r_c
11     Update C as all the objects in the cluster c with medoid p ∈ H
       // sampling on sampleregion
12     p ⟵ SamplingTopM(m, w)
13     K ⟵ FilterClustering(k, α, p)
14     Output K to user
15     User chooses interesting subset H and sets k
16 Output final set of skyline objects user preferred
```

3.4.2 Visualization

To support our hierarchical browsing process, we provide a visualization tool to ensure that users can easily see the difference between the representatives and select the representatives that are of interest to them.

Parallel coordinates[66] is a common way of visualizing high-dimensional geometry and analyzing multivariate data. To show a set of objects in a d-dimensional space, this technique represents data dimensions as d parallel lines spaced equally. Data object in d-dimensional space is represented as a polyline linking n vertices on each axes. The i^{th} vertex is mapped to position on i^{th} axis proportional to its value for that dimension. For our purpose, each of these axis represents a dimension in the weight space and users can thus indirectly indicate their preferred weight setting by

selecting the clusters based on the visualization. For each cluster, we take the *MBR* that represent its coverage and plot the bounding values along each dimension on the corresponding axes. To enhance the visualization further, we take the average values of the samples along each dimension and plot a line that goes through these averages for each dimension. Users can estimate the average weight setting for each dimension by looking at this line. Generally, the range of different clusters overlapping and crossing each other, which renders the graphic representation unclear. Instead, we approximate display the cluster coverage. For the cluster c, the weight setting w belongs to it if $\pi_w[1] = p \wedge p \in c$. Let the mean value of all the weight settings belongs to cluster c on the i^{th} dimension be $\mu_i(c)$, and the standard deviation of them be $\sigma_i(c)$. To ameliorate the visualization, we restrict the range as $[\mu_i(c) - \sigma_i(c), \mu_i(c) + \sigma_i(c)]$ on the i^{th} dimension to control the size of the cluster c.

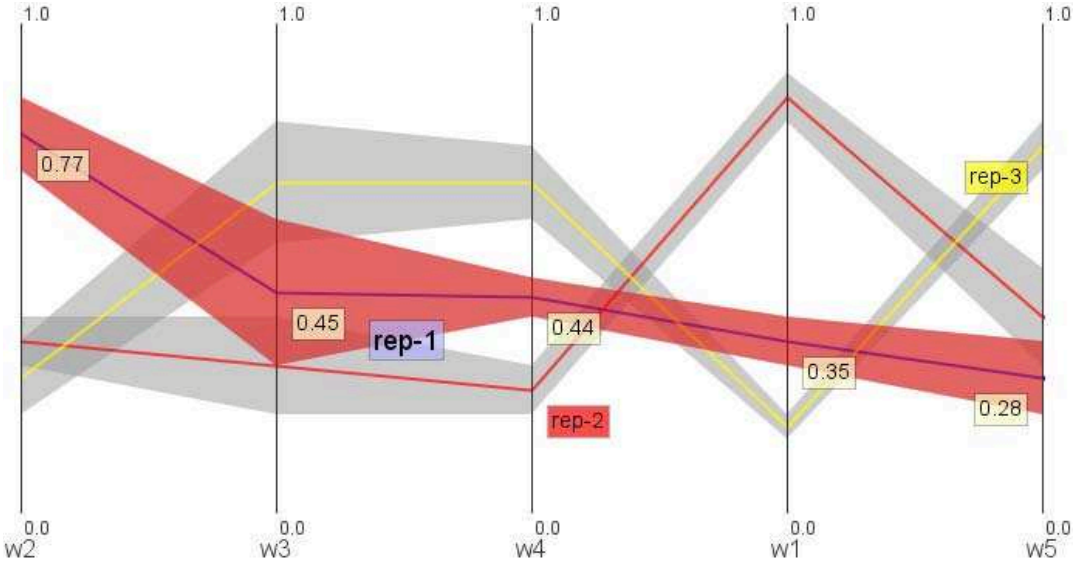


Figure 3.2: Visualization Example

Figure 3.2 gives an example of our visualization technique. According to the definition of weight space, all dimensions are within the range of 0.0 to 1.0. In the diagram, three clusters are represented with three representatives: rep-1, rep-2 and rep-3. Each cluster is visualized as a polygon. In each dimension, the cluster region is restricted by the upper bound and the lower bound respectively. Furthermore, the polyline in the middle of each region is shown for users to estimate the average weight settings of the cluster in each dimension. In addition, the values of each of the representative skyline in the data space are also presented for users to link their preferences back to the actual domain that is familiar to them. To distinguish differ-

ent clusters, the polyline and representative label belonging to the same cluster are colored similarly, while the colors are different between clusters. This framework supports highlighting cluster as well. For instance, the cluster with representative rep-1 is highlighted with red color in Figure 3.2.

Ordering of Axes: Another simple but powerful feature of our visualization tool is that it supports dynamic ordering of the axes based on the selected cluster c. The dimensions are arranged from left to right following the order $w_1, w_2, \ldots, w_d$ if $u_1(c) \geq u_2(c) \geq \ldots \geq u_d(c)$. By looking at the order of these dimensions, users can quickly assess the strength of the cluster by looking at the relative ranking of the dimensions and compare these ranking against what they preferred.

Furthermore, the gradient of the polyline after the ordering give a good indication of the tradeoff between clusters' attributes. A steep gradient indicates that the tradeoff between the attributes is high while a gentle gradient indicates that the importance of attributes are almost the same. For example, the dimensions of Figure 3.2 are reordered to be $\{w_2, w_3, w_4, w_1, w_5\}$. The underlying meaning is that this cluster of skyline objects ranks high mainly because of the dominance on attributes w_2. Furthermore, the steep gradient of this cluster demonstrates that the quality on attribute A_1 and A_5 must drop substantially in the cluster to sustain the strength on A_2.

Alternatively, users will also be allowed to rank the dimensions themselves. Once they ordered the dimensions, they can identify clusters of interest to them by looking for polylines that are approximately decreasing from left to right. Among all those that are decreasing, they can also assess the tradeoff by looking at the gradients.

3.5 Experiments

We now present the experimental study to evaluate order-based representative skylines. For simplicity, we refer to our algorithm as *SampleClus*. In Section 3.5.1, the proposed algorithm is measured with respect to the efficiency as well as effectiveness on synthetic datasets. Section 3.5.2 further illustrates its performance on the real *NBA* dataset[90]. At last, the effect of different monotonic functions and the process of hierarchical elicitation are evaluated in Section 3.5.3. All experiments are

executed on the Windows operating system with Intel Core-2 Duo processor and 4 GB RAM.

3.5.1 Synthetic Data

The synthetic datasets are created using the anti-correlated distribution according to the classical method[17]. Every attribute on each dimension is normalized to [0,1]. Table 3.1 shows the range and default values (in bold) of the parameters. In each experiment, we adjust a single parameter while keeping the rest at their default values. Note that the confidence interval $1-\gamma$ and margin of error δ are two variables for controlling suitable sampling size. Due to the space constraint, other two parameters α and β are fixed in our experimental settings. The default value for α is 0.01 as we expect the representative skyline object covers at least one percent of the weight space. The default value of β is set to be $1/k$, i.e. $\mathcal{SIM}_m(p,q) \geq \beta = 0.1$, since the closest representative should be better than the average case to represent the non-representative objects. The experimental study on α and β are reported and can be found in the full technical report[2]. The evaluation is based on the dot linear product preference function.

Table 3.1: Parameter Settings

Parameter	**Range**
γ	0.1,**0.2**,0.3,0.4
δ	0.01,**0.02**,0.03,0.04
Dimensionality	2,**3**,4,5
k	4,6,8,**10**,12
Data Size(100K)	2,4,6,8,**10**

Table 3.2: Varying γ

γ	0.1	0.2	0.3	0.4
Sampling Size	4,774	3,618	2,956	2,492
Sampling time(ms)	1,676	1,254	1,025	865

We measure the performance of the algorithm in seven aspects. The first five refer to # top objects, # replacements, $Quality(K,S)$, $V(R(K))$, and $Er(K,S)$, which evaluate

[2]This file cannot be cited though because of anonymous requirement.

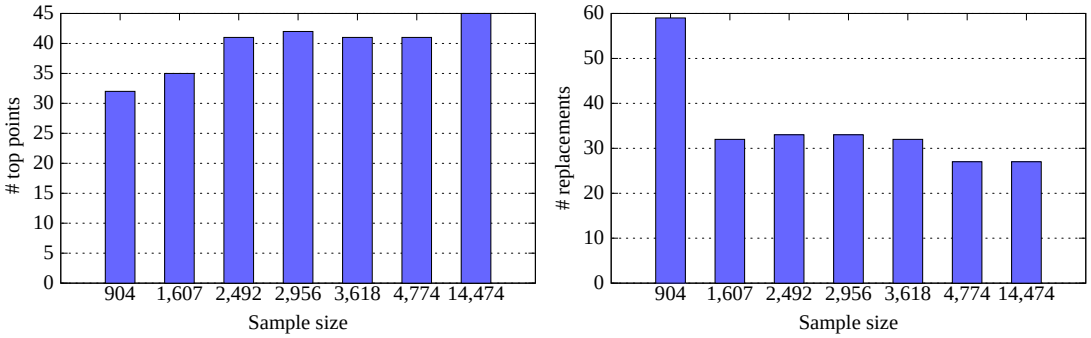
Table 3.3: Varying δ

δ	0.01	0.02	0.03	0.04
Sampling Size	14,474	3,618	1,607	904
Sampling time(ms)	5,056	1,254	559	330

the effectiveness of the algorithm. The first one reflects how many distinct skyline objects appear as top objects in the samples. The # replacements is utilized to evaluate the robustness of the clustering algorithm, which records the number of objects varying from one cluster to another due to the alteration of sampling size. Recall that the $Quality(K, S)$ is described in definition 3.2.1. The $V(R(K))$ is the summation of all $V(R(p))$ as long as $p \in K$ since they are mutually exclusive. For $Er(K, S)$, it indicates the representative error to measure the distance between the representative skylines and the other skyline objects[112]. The remaining two aspects, # IO and CPU time, assess the efficiency of the algorithm. # IO consists of two parts, the random access times (RA) and the sorted access times (SA). Moreover, CPU time is also divided into sampling time and clustering time for better understanding the performance of our *SampleClus* algorithm. The breakdown of execution time provides a deeper and clearer view of the experimental result. Furthermore, due to the random nature of the sampling output, we repeat each experiment ten times and report the average measurements.

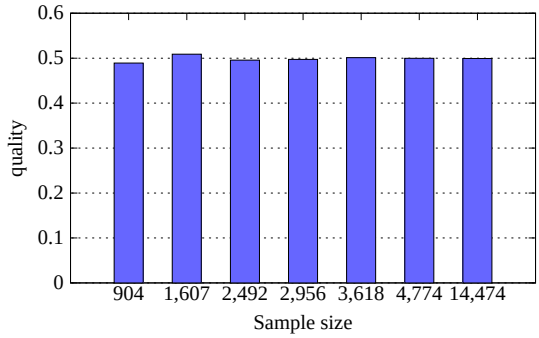
We first investigate how the confidence interval $1-\gamma$ and margin of error δ affect the performance of *SampleClus*. Since the α and the β are fixed, the sampling size N is determined by these two parameters. Table 3.2 shows how sampling size varies as γ changes from 0.1 to 0.4 while fixing $\delta = 0.02$. On the other hand, by increasing δ from 0.01 to 0.04 with $\gamma = 0.2$, we derive the sampling size in Table 3.3. Because the sampling quality is directly related to the sampling size, we continue the analysis based on the sampling size. To begin with, we generate the initial k-partitioning clusters using 4326 samples, which is the mean of all the sampling sizes in Table 3.2 and 3.3. Additionally, by varying the sampling size, we record # top objects, # replacements, $Quality(K, S)$ and $V(R(K))$, and display them in Figure 3.3. Generally, the trend of # top objects suggests that it increases with growing sampling size. However, the increase ratio tends to converge to the real # top objects, which is larger at the beginning while smaller at the end. Concerning the # replacements, the large amount of replacements for sampling size 904 is because of its low accuracy.

Other than this, the tiny difference is mainly due to the randomness of the sampling process. Most importantly, the *Quality*(K, S) and $V(R(K))$ of the clustering results demonstrate *SampleClus*'s robustness although the change in sampling size is noticeable. The *stdev* of *Quality*(K, S) is 0.007 and that of $V(R(K))$ is 0.006. This is primarily determined by the small number of replacements and the stability of the k representatives to incorrect approximation of $\mathcal{SIM}(p, q)$ when it happens with low probability.

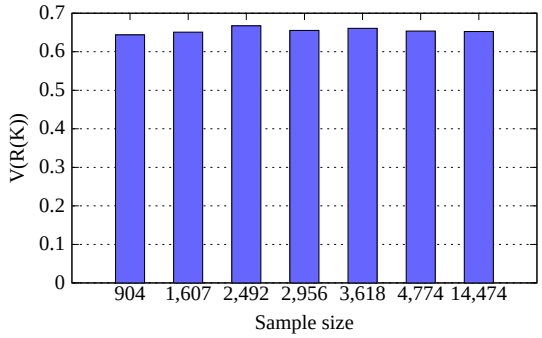


(a) # Top Points

(b) # Replacements



(c) Quality(K,S)

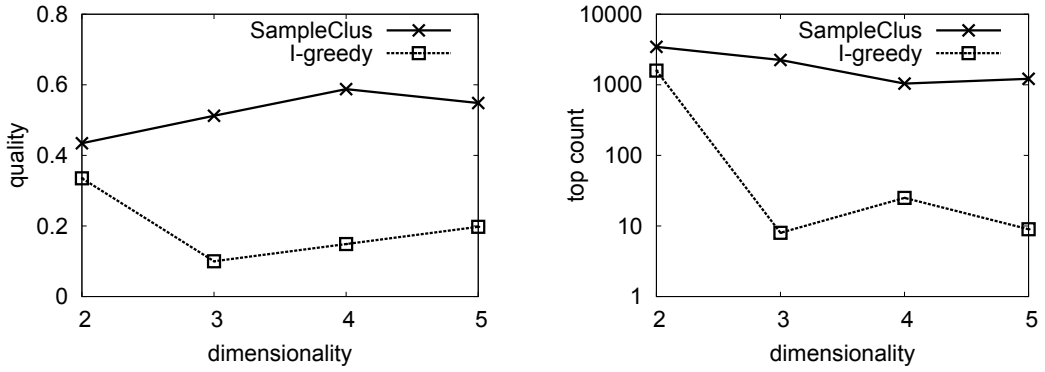


(d) V(R(K))

Figure 3.3: Robustness vs. Sampling Size

Furthermore, the sampling time is linear related to the sampling size as shown in Tables 3.2 and 3.3, since finding the top-m skyline objects for each sample almost costs the same amount of time. Taking both the clustering robustness and sampling time into consideration, we conclude that moderate size of samples is enough for good clustering outputs. Based on this observation, we choose $\gamma = 0.2$ and $\delta = 0.02$, which determine the sampling size to be 3618.

Figure 3.4 shows the comparison between *SampleClus* and *I-greedy* with respect to dimensionality. Note that we generate ten sample sets other than the one used in *SampleClus* to test the representative skylines of two algorithms. The figure suggests that *SampleClus* is superior to *I-greedy* both for $Quality(K, S)$ and $V(R(K))$ in any dimensionality. The $V(R(K))$ is multiplied by the sampling size for clearer display. The closeness of the two algorithms in two dimensional cases is because the number of skyline objects is 57, which is in the same order of magnitude as the number of representatives. Other than this, the distance-based representative skylines can hardly represent the order information as analyzed in the Section 2. Furthermore, the distance based metric is sensitive to the dimensionality. The goal of *I-greedy* algorithm is to minimize the $Er(K, S)$. Accordingly, we define a relative representative error $NormEr(K, S)$ as $Er(K, S)/\sqrt{d}$, where $\sqrt{d}$ is the maximal possible distance between two objects in d dimensional normalized space. By varying dimensionality from 2 to 5, as shown in Table 3.4, $Er(K, S)$ as well as $NormEr(K, S)$ increases along with the rise in dimensionality. It suggests that this goodness function deteriorates in high dimensional cases.



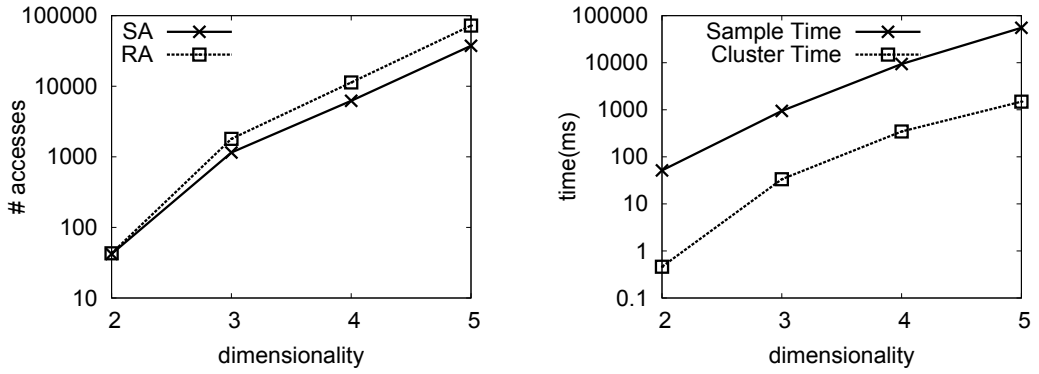
(a) Quality(K,S)

(b) V(R(K))· Sample Size

Figure 3.4: Effectiveness vs. Dimensionality

Figure 3.5 shows the efficiency measures as a function of dimensionality. As dimensionality varies from 2 to 5, the $|S|$ increases dramatically because of the property of anti-correlated distribution. The corresponding skyline sizes equal to 57, 990, 7745, 36290 for dimensionalities 2 to 5 respectively. Therefore, both # IO and CPU time rise linearly with respect to dimensionality.

Next, we vary the number of representatives k to explain how this parameter affects



(a) #IO (b) CPU Time

Figure 3.5: Efficiency vs. Dimensionality

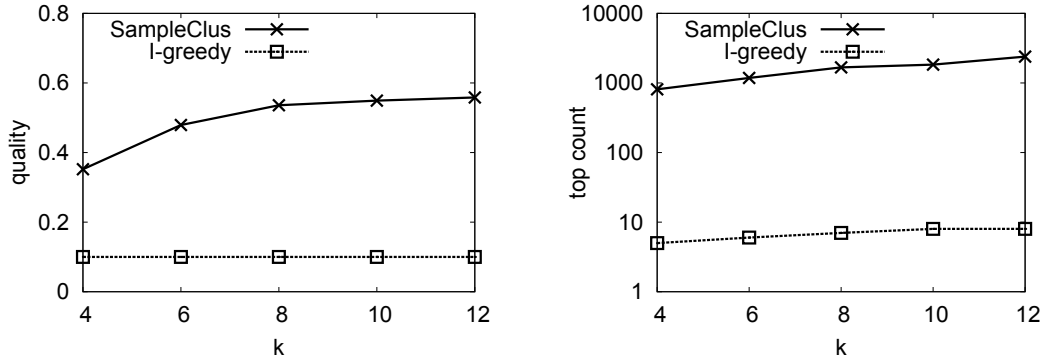
Table 3.4: The Relative Representative Error

Dimensionality	**2**	**3**	**4**	**5**
$Er(K,S)$	0.09	0.39	0.64	0.86
$NormEr(K,S)$	0.06	0.23	0.32	0.38

the effectiveness of our algorithm. The $V(R(K))$ is multiplied by the sampling size for clearer display. As shown in Figure 3.6, the $Quality(K,S)$ and $V(R(K))$ of *I-greedy* almost remain constant as the number of representatives increases. For *SampleClus*, the $Quality(K,S)$ and $V(R(K))$ are always greater than those of *I-greedy*, and increase as more representatives are returned.

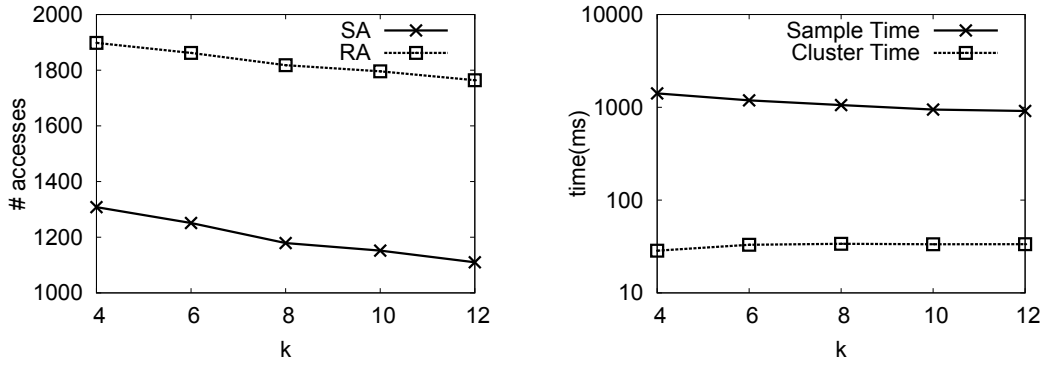
In Figure 3.7, we present the effect of k on the efficiency measurements. As m equals to $|S|/k$, when the skyline set is fixed, m decreases along with the increase of k. Therefore, both of the random access times and sorted access times decrease accordingly. Similarly, we need to discover smaller top-m skyline list for each sample, so the sampling time reduces since the sampling size keeps invariable. On the other hand, the search space enlarges with respect to k, leading to the growing of the clustering time.

The last set of experiments focuses on the scalability of our algorithm as the function of cardinality. Although the cardinality of the dataset increases, the related skyline sizes are 773, 808, 936, 1101, 990 for cardinality 200K to 1M. Figure 3.8 presents the result. The performance does not show any significant changes since the major



(a) Quality(K,S) (b) V(R(K))· Sample Size

Figure 3.6: Effectiveness vs. *k*



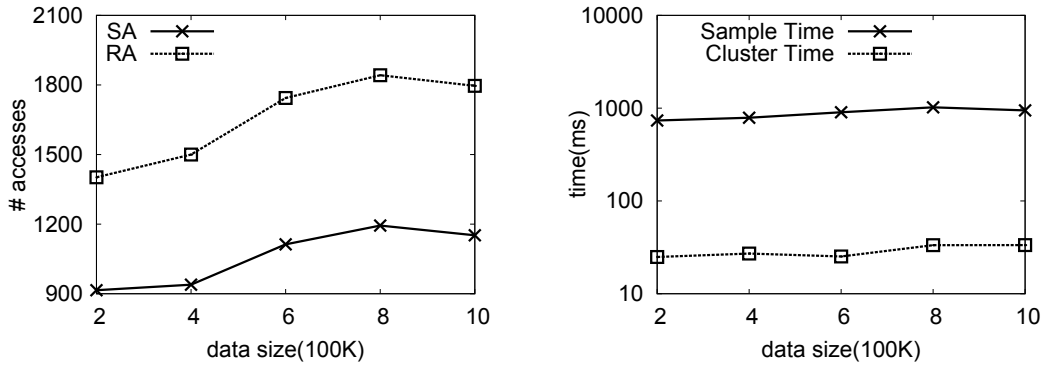
(a) #IO (b) CPU Time

Figure 3.7: Efficiency vs. *k*

factor is the skyline size, but not the dataset cardinality. The trend of the curve is proportional to the number of skyline objects.

3.5.2 Real Data

In this section, we report results of experiments performed on the *NBA* dataset. *NBA* includes 16399 nine-dimensional objects. We denote each object as $p(A_1, A_2, \ldots, A_9)$, representing the regular season performance of a player from 1973-2008 on nine attributes: points per game (pts), rebounds per game (reb), assists per game (ast), steals



(a) #IO (b) CPU Time

Figure 3.8: Efficiency vs. Cardinality

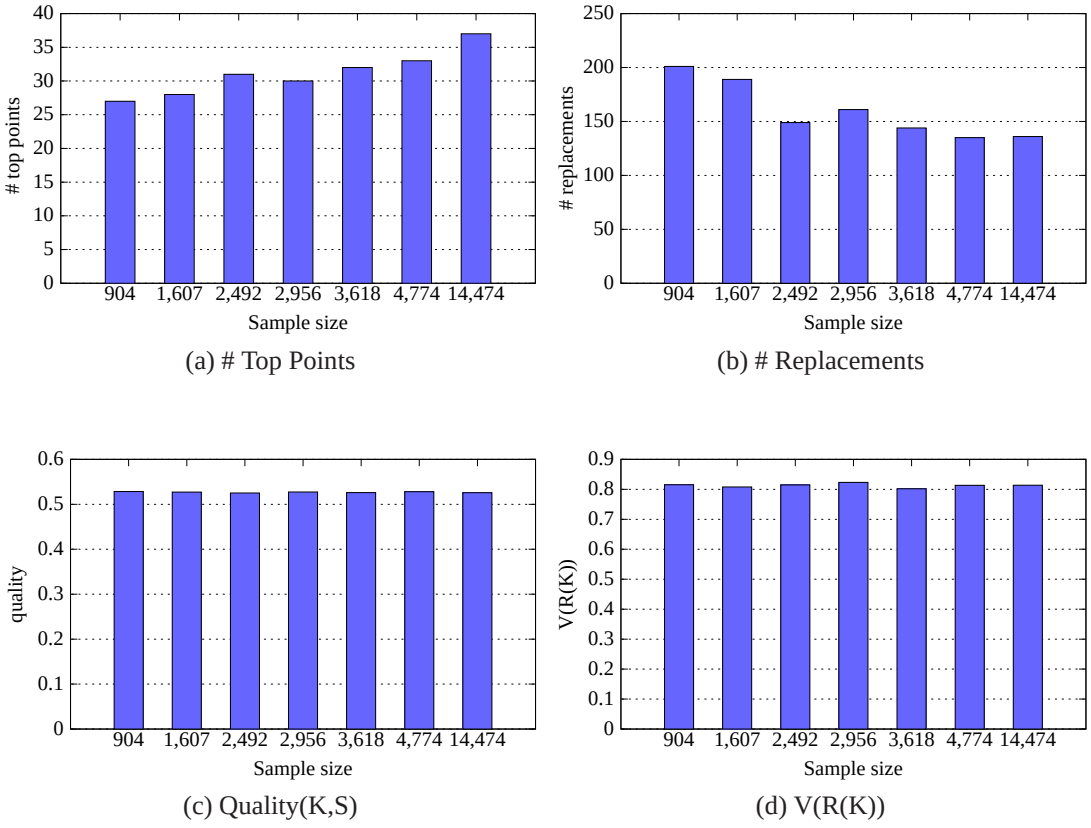
per game (stl), blocks per game (blk), assists to turnovers (a/t), field goal percentage (fgp), free throw percentage (ftp) and three points percentage (tpp). The skyline set of *NBA* consists of 1024 players. Since the dataset's properties are fixed, we adjust γ, δ and k to measure the performance.

First, we show the quality of the results as a function of γ and δ in Figure 3.9. Following the same setting of γ and δ, the derived sampling size is the same as that of the synthetic data. The values of # top objects, # replacements, $Quality(K, S)$ and $V(R(K))$ with respect to sampling size are shown in Figure 3.9. Although the robustness properties are similar, there exist several distinctions due to the correlations between *NBA* attributes. As such, the # top points is fewer and the # replacements becomes larger. Furthermore, the region sizes between different skyline objects are skew, resulting in better $Quality(K, S)$ and larger $V(R(K))$ when compared to these measurements for the synthetic data.

Table 3.5: Sampling Time vs. γ and δ

γ	0.1	0.2	0.3	0.4
Sampling time(ms)	2,731	2,025	1,653	1,409
δ	0.01	0.02	0.03	0.04
Sampling time(ms)	8,363	2,025	898	515

Figure 3.10 displays the relationship between k and the effectiveness of the representatives. The $V(R(K))$ is multiplied by the sampling size for clearer display. The *I-greedy* algorithm exerts no explicit relationship with the change of k. On the other



(a) # Top Points (b) # Replacements

(c) Quality(K,S) (d) V(R(K))

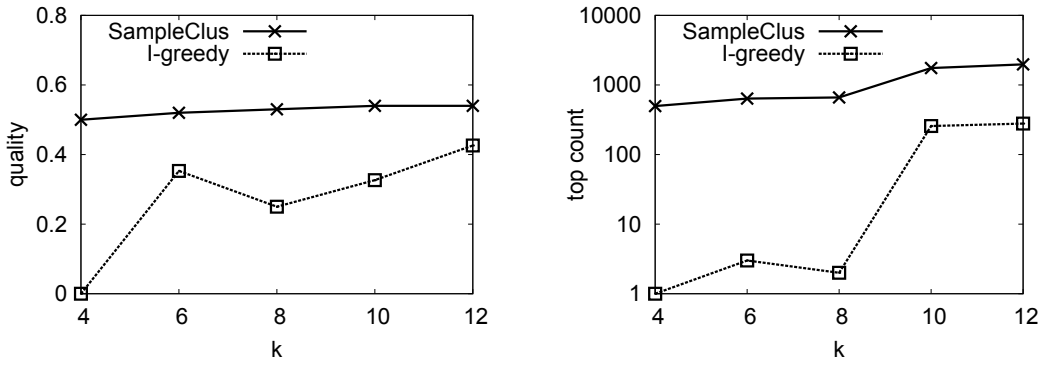
Figure 3.9: Robustness vs. Sampling Size

hand, the order-based representative skylines present better $Quality(K, S)$ as well as $V(R(K))$ in comparison to distance-based representative skylines. Since the *NBA* dataset has correlated character, the gain of the two measures in *SampleClus* are not so significant by adding more representatives.

Figure 3.11 shows the relationship between k and the efficiency of the representatives. When k varies from 4 to 12, the values of m are 256, 171, 128, 102, 85 respectively. Consequently, except for clustering time in proportion to k, the random access times, sorted access times and sampling time decrease as k increases.

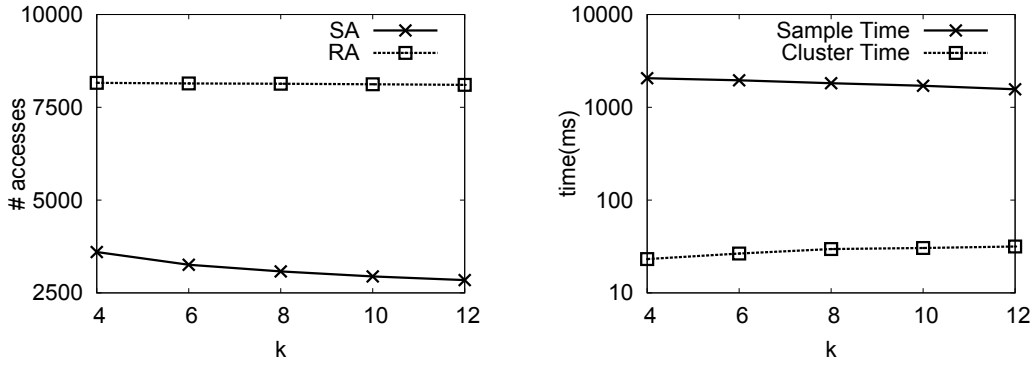
3.5.3 Case Study of Preference Elicitation

In this section, we further investigate the effect of preference function and the process of hierarchical browsing. Both these two factors exert an important influence on



(a) Quality(K,S) (b) V(R(K))· Sample Size

Figure 3.10: Effectiveness vs. k



(a) #IO (b) CPU Time

Figure 3.11: Efficiency vs. k

the outcome of preference elicitation. The experiments are conducted on the *NBA* dataset.

To begin with, we test the algorithm on different monotonic functions. Unlike distance-based representative skylines, the order-based representative skylines could vary on the same skyline set to reflect the underlying interest of different users.

We illustrate three different monotonic functions in Table 3.6 to show the distinct perspectives on the *NBA* dataset. For the function $\overrightarrow{f_1(\cdot)}$, the user favors players who are comparable in attributes ast, stl and a/t, while $\overrightarrow{f_2(\cdot)}$ could be a good choice if the user prefers players with better reb and blk. Comparing between Table 3.7 and 3.8,

Table 3.6: The Preference Functions

	Attributes
$\overrightarrow{f_1(\cdot)}$	$(pts, \sqrt{reb}, ast^2, stl^2, \sqrt{blk}, a/t^2, fgp, ftp, tpp)$
$\overrightarrow{f_2(\cdot)}$	$(pts, reb^2, ast, stl, blk^2, a/t, fgp, \sqrt{ftp}, \sqrt{tpp})$
$\overrightarrow{f_3(\cdot)}$	$(pts, reb, ast, stl, blk, a/t, fgp, ftp, tpp)$

Table 3.7: The $\overrightarrow{f_1(\cdot)}$ Representatives

Player ID	**pts**	**reb**	**ast**	**stl**	**blk**	**a/t**	**fgp**	**ftp**	**tpp**
2006_Nash	18	4	**12**	0.8	0.1	3	.53	.90	.45
1975_Jabbar	28	17	**5**	1.5	4.1	0.0	.53	.70	.00
1987_Bird	30	9	**6**	1.6	0.8	2.2	.53	.92	.41
1987_Jordan	35	5	**6**	3.2	1.6	1.9	.54	.84	.13
1991_Stockton	16	3	**14**	3.0	0.3	3.9	.48	.84	.41

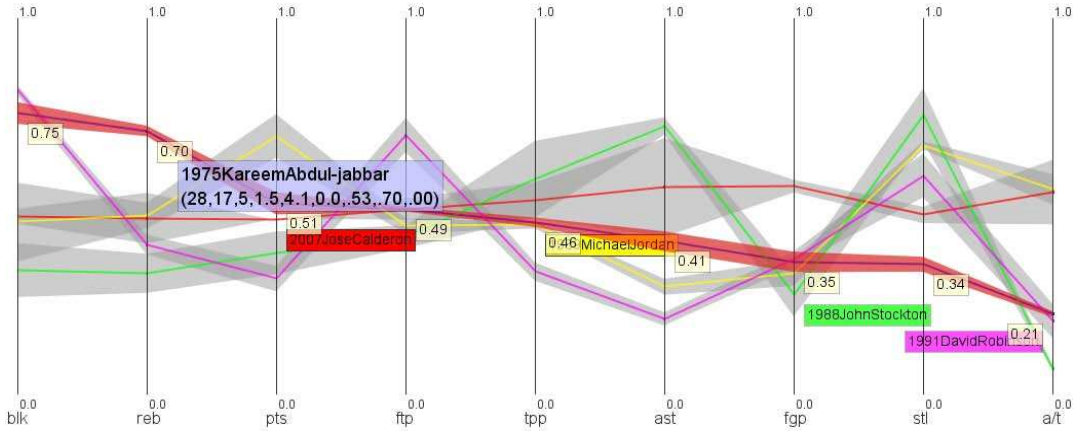
the five order-based representative skylines of $\overrightarrow{f_1(\cdot)}$ and $\overrightarrow{f_2(\cdot)}$ are of noticeable distinction. The former contains good assisters such as Nash and Stockton, while the latter includes outstanding defenders: Gilmore, Ewing and Macdoo. Moreover, taking ast for instance, the average ast of representatives in Table 3.7 are much higher than that in Table 3.8. Note that the output changes according to monotonic function is totally different from ranking based on specific function. The order-based representative skylines achieve a tradeoff between accuracy and heterogeneity, so the all-round players have the high probability to be selected as the representatives, such as Jordan and Jabbar. Besides comprehending the overall situation of the skyline set, users are likely to find desired objects as well. However, the results of distance-based representative skylines, as shown in Table 3.9, are less satisfactory. Although close to other skyline objects in Euclidean distance, most of the representatives themselves are not quite important. Furthermore, the result is fixed and unable to express the difference between the preference functions.

Table 3.8: The $\overrightarrow{f_2(\cdot)}$ Representatives

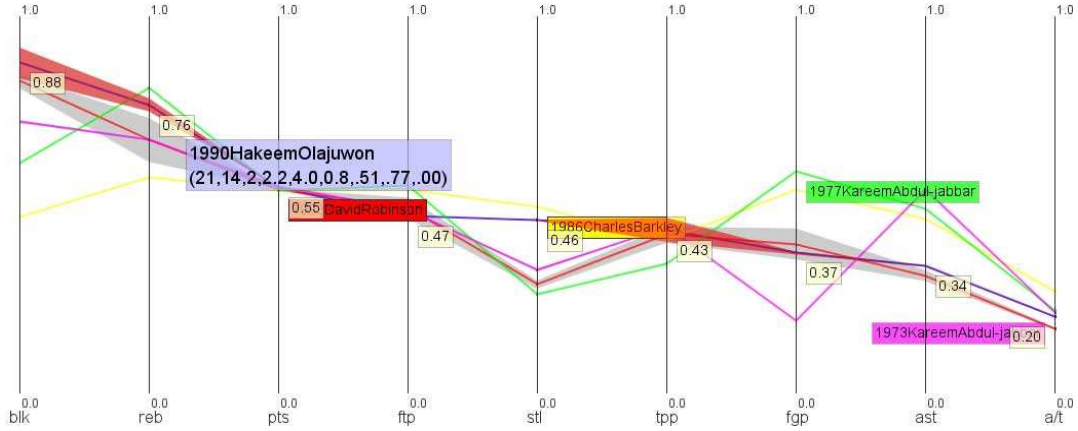
Player ID	**pts**	**reb**	**ast**	**stl**	**blk**	**a/t**	**fgp**	**ftp**	**tpp**
1980_Gilmore	18	10	**2**	0.6	2.4	0.7	.67	.70	.00
1975_Jabbar	28	17	**5**	1.5	4.1	0.0	.53	.70	.00
1989_Ewing	29	11	**2**	1.0	4.0	0.7	.55	.77	.00
1986_Jordan	37	5	**5**	2.9	1.5	1.4	.48	.86	.18
1974_Mcadoo	35	14	**2**	1.1	2.1	0.0	.51	.81	.00

Table 3.9: The Distance-based Representatives

Player ID	pts	reb	ast	stl	blk	a/t	fgp	ftp	tpp
1989_Bogues	11	3	**9**	1.3	0.0	5.1	.48	.89	.19
1997_Rodman	5	15	**3**	0.6	0.2	1.6	.43	.55	.17
2003_Wallace	17	7	**3**	0.8	1.6	1.3	.44	.74	.34
2008_ Diener	4	2	**2**	0.5	0.1	5.8	.41	.80	.39
1986_Jordan	37	5	**5**	2.9	1.5	1.4	.48	.86	.18



(a) First Level Visualization



(b) Second Level Visualization

Figure 3.12: Example of Hierarchical Browsing

As displayed above, the tabular view of result is not intuitive especially for high dimensional case. We thus visualize the process of hierarchical browsing of $\overrightarrow{f_3(\cdot)}$ using the approach presented in Section 3.4. Since we adopt the linear function, the five representative players are averagely excellent. In Figure 3.12(a), the axes are ordered according to highlighted representative Jabbar, whose strengths are blk and

reb. Also, the set of representatives are well separated and covering large area on the weight space. For example, the representative Stockton dominates distinct region comparing with Jabbar, which is reasonable because they are totally different kinds of players. Following the highlight representative, the re-sampling is performed and five new representatives are shown in Figure 3.12(b). These representatives are all excellent defenders as the Jabbar in higher level, nicely following the interest of the user. Note that one object represents one regular season record of certain player, so Jabbar appears twice in the new representative skylines with the records in 1973 and 1977 respectively. Furthermore, the ordering of attributes in the second figure is very close to that in the first one, suggesting that Olajuwon has the similar strength as Jabbar. In summary, the hierarchical browsing approach enables users to drill down to the preferred region effectively, especially with the help of our visualization tool.

3.6 Summary

In this chapter, we have introduced the order-based representative skylines, a novel concept that integrates the discovery of representatives with order preference. Unlike previous work, we brought the preference function back into the picture when determining representative skylines in order to elicit the preference. Moreover, a hierarchical sampling-clustering framework was developed based on the new notion. To further consolidate this interesting framework, we provided visualized view to guide the user's refinement of the result. The outcomes from an experimental study demonstrated that our order-based representative skylines can provide more informative views of data.

Chapter 4

Diversified Keyword Search in Databases

4.1 Overview

In this chapter, we propose to develop a novel keyword search system to support diversified keyword search and browsing over databases. To make this possible, three new challenges must be overcome:

(1) Diversity Measurement: Intuitively, result diversification is a trade-off between having more relevant results of the "correct" intent and having diverse results in the top positions for a given query [52]. As such, aside from considering the relevance of answers, we also need to take into account the pairwise difference between them. Therefore, our first and the most important challenge is to define a meaningful measure between substructures tailored for keyword search in databases. Various efforts have been made to measure the dissimilarity of keyword search results [109, 35, 51]. While we will discuss these papers in detail subsequently, it suffices to point out here that none of them capture both textual and structural information when trying to diversify keyword search answers.

(2) Query Answering: Due to the NP hardness of result diversification [52], it is thus necessary to develop an efficient scheme to produce diversified results. Although finding representatives in clustering problem is a candidate solution, it is imperative to notice that clustering method also has high computational cost. More importantly, the diversity quality of the clustering method is shown low compared with heuristic approaches [37]. Although we try to divide results into k groups, our objective is to make the distinction between k answers as large as possible.

(3) Result Representation: Our ultimate goal is to facilitate search experience and database usability. Since the original structural answers are complex and not easy to understand, we need to simplify them in order to let users quickly perceive the underlying difference between answers. To achieve this goal, the challenge is to effectively summarize distinct features from rich structures and contents in diversified results.

To overcome these challenges, we develop a novel system for browsing and diversified keyword searching in databases, i.e. BROAD (BROAD is an acronym for BROwsing And Diversified keyword searching). Our contributions towards diversified keyword search in databases are as follows:

- We have devised an effective kernel distance to measure the diversity of keyword search results. This metric integrates both the textual difference and the structural distinction in the answer trees.
- We have developed an efficient algorithm to find k diverse keyword query answers based on cover tree index structure. Unlike the post-processing approach, our solution seamlessly combines both relevant result discovery and diverse result set selection, allowing us to dynamically update the search results.
- We have provided a hierarchical browsing interface to further enhance our system. By coupling our solution with summarization techniques, we enable users to efficiently locate desired results by drilling down to relevant answers incrementally.

- We have conducted extensive experiments on two real datasets to show that our framework is both effective and efficient.

The rest of the chapter is organized as follows. Section 4.2 defines the problem handled throughout this work and proposes our new diversity measure. Section 4.3 introduces the BROAD system architecture. Section 4.4 presents the efficient index based solution. The browsing interface of diversified result is described in Section 4.5 and followed by a demonstration in Section 4.6. Our extensive experimental study is reported in Section 4.7. Section 4.8 concludes the chapter.

4.2 Problem Definition

In this section, we introduce the keyword search modeling and describe the diversity problem studied in this work. Furthermore, we propose a novel diversity measure to capture both content and structure information.

4.2.1 Keyword Search Modeling

We model a database as a graph since it is the widely used modeling suitable for unstructured, semi-structured and structured data [79]. Database schema is a directed graph G_S called **schema graph**, in which nodes represent tables and edges represent foreign key references. Edge $R \rightarrow S$ between tables R and S indicates that the foreign key on S refers to the primary key on R. Note that there may exist multiple edges between tables to represent multiple foreign key references. Given the schema graph G_S, the **data graph** G_D consists of nodes representing tuples and directed edges representing the foreign key references between tuples. Consider an l-keyword query q $\{c_1, c_2, \ldots, c_l\}$. Typically, the result of q on G_D is represented as follows.

Definition 4.2.1 *(Answer tree)*
An answer tree T to the keyword query q is a rooted subtree of the data graph, satisfying: T contains all the keywords, and any subtree of T is not a valid answer tree. Denote the root of T as $n_r(T)$ and the node set of T as $N(T)$.

Note that we assume the result has a single root in this work. Generally, without restriction on the size of an answer tree, w e will find a large number of meaningless trees due to long paths between nodes. Instead, we restrict the results to those trees that have a radius less than or equal to r. Note that the radius indicates the largest path length between the root node and leaf nodes, which varies with respect to the dataset. This is a common approach for keyword search in databases [69, 61].

Definition 4.2.2 *(Res(q, r))*
Given keyword query q and radius r, an answer tree T is in the result set $Res(q, r)$ iff the path lengths between $n_r(T)$ and all the keyword nodes are less than or equal to r.

4.2.2 Diversity Problem Definition

We first assume that the dissimilarity between two answer trees can be measured by a distance function $dist(T_a, T_b)$ (with larger distance being more dissimilar), which will be discussed later in this section. There are typically two ways to define diversity. One is the rank aware diversity; another is based on an objective function. The former defines diversity by re-ranking the result taking diversity into consideration. However, since different users have different criteria, it does not always make sense to present a universal ranking. Instead, we discover a set of answers based on an objective function as follows and let users discover which one is his/her intention.

Problem 2 *(Keyword Search Diversification)*
Given keyword query q and radius r, find a set of k answer trees $\mathcal{S} \in Res(q, r)$ which maximize $\sum\{dist(T_a, T_b)\}$ where $T_a, T_b \in \mathcal{S}$.

Max-sum objective is a widely used diverse definition [52, 109, 37]. Nevertheless, our solution can be easily adopted to other popular definitions, such as the threshold based measure [125] and the max-min measure [52]. Without loss of generality, we use Problem 2 to illustrate our idea throughout this work.

4.2.3 Kernel Based Diversity Measure

The core of diversity problem is the need to measure the pairwise dissimilarity between answer trees, i.e. $dist(T_a, T_b)$. Here, we choose a kernel based method for this purpose and will explain our choice subsequently.

Answer Tree Kernel

Formally, a kernel function [105] is a function measuring the similarity of any pair of objects $\{x, x'\}$ in the input domain $\mathcal{X}$. It is written as $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$, in which ϕ is a mapping from $\mathcal{X}$ to a feature space $\mathcal{F}$. Given a set of examples $\{x_1, x_2, \ldots, x_m\}$, the Gram matrix is defined as the $m \times m$ matrix G^κ whose entries are $G^\kappa_{i,j} = \kappa(x_i, x_j)$. A kernel function is valid if and only if it is symmetric positive semidefinite, i.e. if any of its Gram matrices is symmetric positive semidefinite. Readers are referred to the book [105] for a comprehensive introduction on kernel methods.

To ensure efficient computation of the kernel, we utilize the subtree kernel [114] as the starting point since it is a linear complexity kernel for tree structural data. This kernel is extended from the state-of-the-art convolution kernel [60]. The basic idea is to express a kernel on a discrete object by a sum of kernels of its constituent parts. The features of the subtree kernel are proper subtrees of the input tree T. A proper subtree f_i comprises node n_i along with all of its descendants. Two proper subtrees are isomorphic if and only if they have the same tree structure. Considering T_1 and T_7 in Example 1, all of their proper subtrees are shown in Figure 4.1. Both answer trees contain four different proper subtrees, and they share three of them, namely, f_1, f_2, f_3. The definition of subtree kernel is as follows.

Definition 4.2.3 *(Subtree Kernel)*
Given two trees T_a and T_b, the Subtree Kernel is:

$$\kappa_S(T_a, T_b) = \sum_{n_a \in N(T_a)} \sum_{n_b \in N(T_b)} \Delta(n_a, n_b)$$

where $\Delta(n_a, n_b) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_a) I_i(n_b)$, and where $I_i(n)$ is an indicator function which determines whether the proper subtree f_i is rooted at node n.

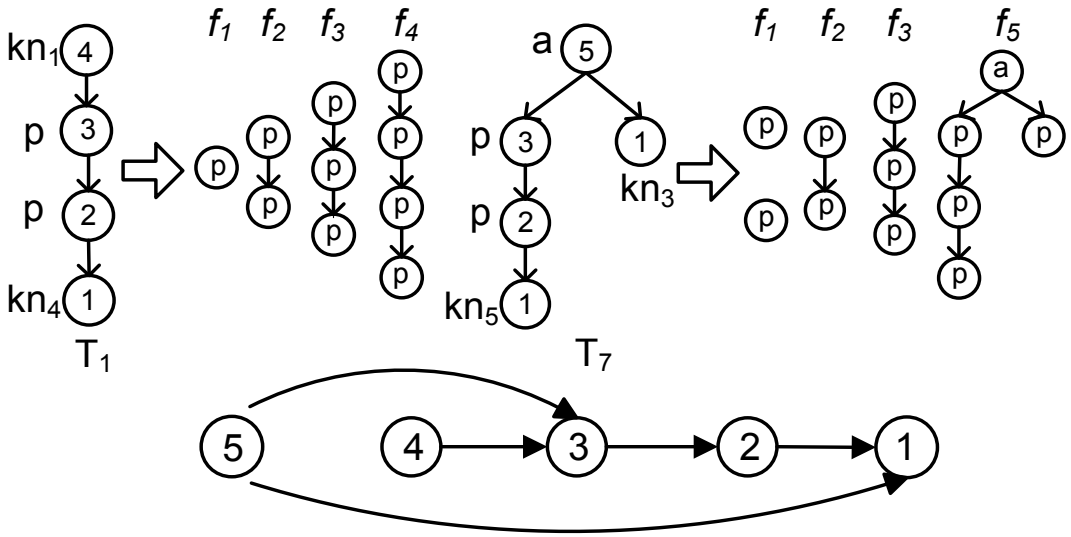


Figure 4.1: Kernel Example

Originally, the subtree kernel is designed to compare only tree structures without taking node contents into consideration. For example, the kernel score $\kappa_S(T_1, T_7) = 1\times2+1\times1+1\times1 = 4$ only because they share the substructures f_1, f_2, f_3. In our case, the comparison between answer trees needs to consider node contents as well. Although Bloehdorn et al. [16] has integrated textual information into the convolution kernel, their approach is designed for parsing tree in grammar analysis. Our paper is the first attempt to design a kernel for structural keyword search answers. We devise a new tree kernel which takes the keyword semantic differences as well as answer tree structural differences into consideration, and can be computed in linear time. The differences between the kernel in [16] and our kernel are two-folds. First, the text kernel in [16] is based on subset structures, which include internal fragments, while our kernel is based on subtree structures, since we focus on the connections between keyword nodes. Second, the partial match in [16] only considers the terminal term differences according to the parsing tree structures, while we also need to take the internal textual difference into consideration.

The idea of answer tree kernel is to take $\Delta(n_1, n_2)$ as a fuzzy match between proper subtrees. Since answer trees contain textual information, we could compare the content similarity of two proper subtrees from two answer trees that have the same structure. Let f_i^a be a proper subtree in T_a and f_i^b be a proper subtree in T_b that share the proper subtree f_i. We merge the textual content in the nodes of f_i^a and f_i^b into d_i^a and d_i^b and refer to them as documents. Next, we represent each document

as $v = (w_1, w_2, \ldots, w_t)$ with each dimension corresponding to a separate term. If a term occurs in the document, its value in the vector is non-zero. Applying one of the best known schemes, i.e. TF-IDF weighting, we obtain $\kappa_D(d_i^a, d_i^b) = \langle v_i^a, v_i^b \rangle$ where v_i^a and v_i^b are the weighted term vectors of d_i^a and d_i^b respectively. Furthermore, the keyword query q provides another source of semantic information. Intuitively, f_i^a and f_i^b contribute more to the overall kernel if they share more keywords. Thus, we introduce a weight setting $w_{ab} = \sqrt{s/l}$ where s indicates the number of shared keywords and l represents the total number of input keywords, yielding:

Definition 4.2.4 *(Answer Tree Kernel)*
Given two trees T_a and T_b, the Answer Tree Kernel is:

$$\kappa_A(T_a, T_b) = \sum_{n_a \in N(T_a)} \sum_{n_b \in N(T_b)} w_{ab} \Delta'(n_a, n_b)$$

where $\Delta'(n_a, n_b) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_a) I_i(n_b) \kappa_D(d_i^a, d_i^b)$, and where $I_i(n)$ is an indicator function which determines whether the proper subtree f_i is rooted at node n.

In order to define a metric distance, we proof that the answer tree kernel is valid in Lemma 4.2.1.

Lemma 4.2.1 *Answer tree kernel is a valid kernel.*

Proof 4.2.1 *For a convolution kernel, if the kernels on the subparts are positive semidefinite, the overall kernel is also positive semidefinite [60]. As the answer tree kernel accords with the convolution kernel format, we need to prove that $\kappa_D(d_i^a, d_i^b)$ is valid. This can be shown by the kernel definition because $\kappa_D(d_i^a, d_i^b)$ is computed explicitly in terms of a dot product. Therefore, the answer tree kernel $\kappa_A(T_a, T_b)$ is a valid kernel and we map the answer tree to a doc product space.*

Answer tree kernel serves as an effective method to map original answer trees to a kernel space. However, in the original answer tree kernel, larger trees have higher

chances to share many common features with any small tree. To overcome this drawback, we compute a normalized kernel, i.e.

$$\kappa(T_a, T_b) = \kappa_A(T_a, T_b) / \sqrt{\kappa_A(T_a, T_a) \cdot \kappa_A(T_b, T_b)} \tag{4.1}$$

Finally, we define a norm $\|T\| = \langle T, T \rangle = \kappa(T, T)$, and then obtain the metric distance via [105]:

$$\begin{aligned} dist(T_a, T_b) &= \|T_a - T_b\| \\ &= \sqrt{\langle T_a, T_a \rangle + \langle T_b, T_b \rangle - 2\langle T_a, T_b \rangle} \\ &= \sqrt{\kappa(T_a, T_a) + \kappa(T_b, T_b) - 2\kappa(T_a, T_b)} \\ &= \sqrt{2(1 - \kappa(T_a, T_b))} \end{aligned}$$

The above deduction relies on $\kappa(T_a, T_a) = \kappa(T_b, T_b) = 1$ by substituting Equation 4.1.

Alternative Methods

There exist several different ways to define the similarity between answer trees. We could extract a finite-length feature vector for each answer tree, and then map it to a feature space to calculate the similarity via dot product. However, explicitly defining an effective feature space needs domain expert knowledge. Another way is to adopt tree edit distance [34]. This metric is defined as the minimal number of edit operations to transform one tree to another. However, computing tree edit distance for trees T_a and T_b suffers an expensive computational complexity $O((|T_a| + |T_b|)^3)$ [34]. Compared to these methods, the kernel based approach can be computed in linear time and capture both structural and textual similarity without the need for domain knowledge.

Besides, we can decompose answer trees into a set of nodes and utilize Jaccard's distance to measure the difference. This method is efficient but sacrifices the result quality. First, it only considers the exact match of nodes, but ignores the textual similarity between them. Second, it fails to measure the structural connections due to a decomposition. Two recent work [35, 109] apply the Jaccard's distance by separating answer trees into a set of nodes. We compare them with our method in

the experimental section to show the kernel distance can achieve better precision and recall.

4.3 System Architecture

We next present the BROAD system architecture as in Figure 4.2. We try to use a pipelined framework to overcome the challenges we discussed earlier. When a user inputs one l-keyword query in the browsing interface, it will be sent to keyword search engine generating candidate answer tree set $\mathcal{T}$. Here we rely on the standard keyword search engine in graph databases, which discovers answer trees from the data graph building on top of relational databases [3, 69]. Note that this component can be easily replaced with the relational keyword search engine [64]. Our BROAD system builds the connection between user interface and keyword search engine. It mainly consists of three components: Cover Tree Indexer, Diverse Result Generator and Hierarchical Browsing Operator. The results from the search engine can be progressively inserted into cover tree index in an online fashion. Based on this index structure, we will discover diverse result set and interact with users in a hierarchical browsing manner. For better illustration, we briefly explain the functionality of these components in BROAD system as follows.

- Cover Tree Indexer: This module is the core of our system and will be discussed in details in Section 4.4. It dynamically manages the answer trees that are returned by search engine. The kernel calculator serves as a subcomponent that computes the distance between answer trees based on the schema graph, so that the cover tree can index results effectively.

- Diverse Result Generator: The generator relies on the Cover Tree Indexer to discover k diverse results. This can not only directly show results to users, but also provides them with the Hierarchical Browsing Operator for further improvement.

- Hierarchical Browsing Operator: This component allows users to browse answer trees in a hierarchical fashion and will be discussed in Section 4.5. The hierarchy is constructed by partitioning answer trees into k groups based on

their similarity to the k diverse results, and then recursively applying partitioning to each group. By summarizing the answer trees in each of the k groups, we provide a way for users to quickly locate the desired results.

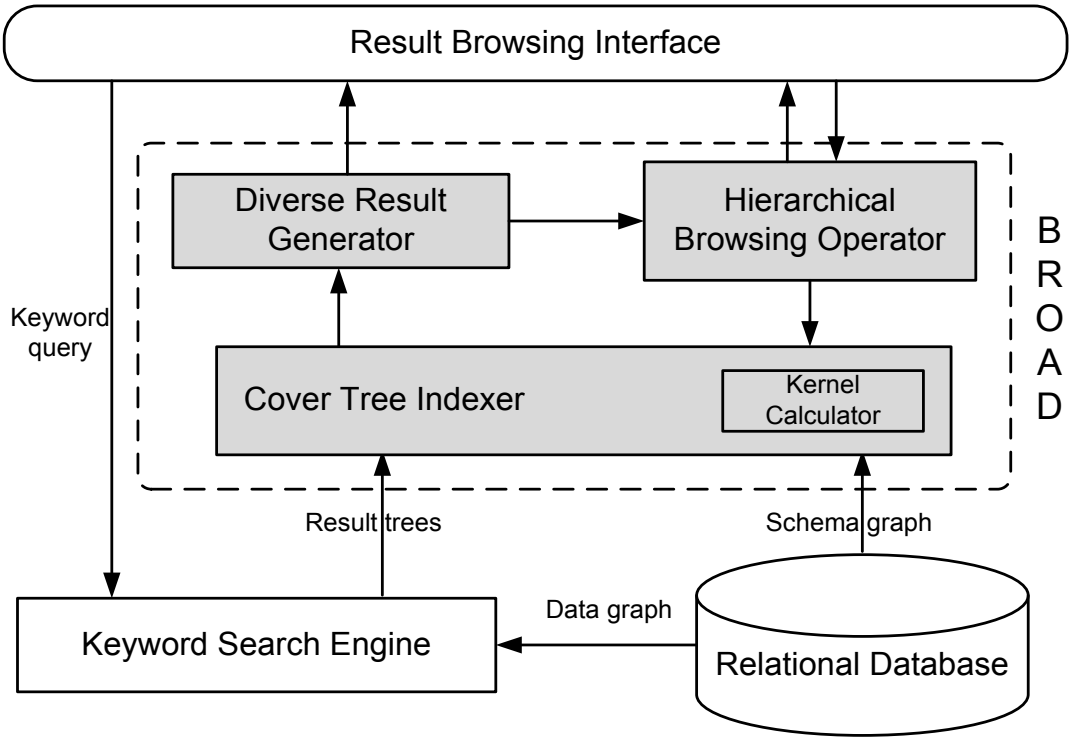


Figure 4.2: BROAD System Architecture

4.4 Methodology

In this section, we propose an efficient algorithm computing the tree kernel distance. Based on this, we develop a cover tree based algorithm to solve Problem 2. Alternative approaches are listed in Section 4.4.3.

4.4.1 Kernel Distance Computation

To compute the tree kernel distance, a naïve calculation follows naturally from the idea in Definition 4.2.4. Intuitively, this method checks all the possible combinations

between nodes of two answer trees and sums up the shared parts to obtain the final score. It is straightforward but suffers from $O(|T_a||T_b|)$ computational complexity. Here we consider this problem from another aspect. The number of proper subtrees in a tree equals to the size of the tree. Let us consider Figure 4.1 again. T_1 has four proper subtrees $\{f_1, f_2, f_3, f_4\}$, and T_7 has five $\{f_1, f_1, f_2, f_3, f_5\}$. Therefore, we could directly enumerate all proper subtrees instead of checking every possible node combination.

Based on this intuition, we design a novel bottom-up algorithm to merge answer trees into a directed acyclic graph. The graph at the bottom of Figure 4.1 is generated from answer trees T_1 and T_7. The number inside each node represents the correspondence between a tree node and a graph node. For instance, the nodes with label 3 in two answer trees can be merged into the graph node with label 3. It is because they have the same structure f_3, in which all the nodes come from the "paper" table. Due to the bottom-up traversal, the children of newly accessed node must be mapped to certain graph node before it. Thus, by checking the child correspondences, we could easily determine whether this node should be mapped to an existing node or we need to create a new graph node. At last, each graph node represents one kind of proper subtree, because we create a new graph node if and only if we discover a new proper subtree. In this example, the two answer trees are merged into the graph with five nodes, indicating that they contain five different substructures in total. Following Definition 4.2.4, we calculate and sum up kernel scores of all the substructures to derive the final kernel score.

The improved algorithm contains two major subcomponents as in Algorithm 4. Function **buildDAG** merges two answer trees into one directed acyclic graph G. Following the bottom-up order, we add nodes in T_a into the *leftset* of nodes in G and nodes in T_b into the *rightset* of nodes in G. We then utilize G in the **kernel** function. This component computes semantic scores based on the *rightset* and the *leftset* of each graph node, and adds them up to obtain kernel score. In the main algorithm, we need to derive the self kernels for T_a and T_b and the cross kernel between T_a and T_b. Finally, we can calculate the kernel distance $dist(T_a, T_b)$ in line 4. Concerning the computational cost, the merging part needs single bottom-up traverse of two answer trees, and the computing part has $O(|G|)$ complexity with $|G| \leq |T_a| + |T_b|$. Obviously, the total complexity of Algorithm 4 is linear to the answer tree size.

Algorithm 4: **KernelDistance**

Input: Answer trees T_a and T_b
Output: The kernel distance $dist(T_a, T_b)$

DAG $G_{ab} \longleftarrow$ buildDAG(T_a, T_b)
DAG $G_{aa} \longleftarrow$ buildDAG(T_a, T_a)
DAG $G_{bb} \longleftarrow$ buildDAG(T_b, T_b)
$dist(T_a, T_b)$=
$\sqrt{2(1 - kernel(G_{ab})/\sqrt{kernel(G_{aa})kernel(G_{bb})})}$

buildDAG(Answer tree T_a, Answer tree T_b)

enqueue T_a's and T_b's leaf nodes into queue Q
create empty DAG G
while *Q is not empty* **do**
    dequeue node w from Q ; $found \longleftarrow false$
    foreach *node $v \in G$ in bottom up order* **do**
        break if v and w have different heights, outdegrees, or provenances
        if *v and w have the same children* **then**
            if $w \in T_a$ **then** add w to $v.leftset$
            else if $w \in T_b$ **then** add w to $v.rightset$
            $found \longleftarrow true$; break
    if *found = false* **then**
        add a new node v to G
        if $w \in T_a$ **then** add w to $v.leftset$
        else if $w \in T_b$ **then** add w to $v.rightset$
        add arcs in G from v to all children of w
    if *$w \neq$ Root **and** parent(w)' children are processed* **then** enqueue node parent(w) into Q

kernel(DAG G)

$\kappa_A \longleftarrow 0$
foreach *node $v \in G$ in bottom up order* **do**
    $d_a \longleftarrow \bigcup$ text content of v.leftset
    $d_b \longleftarrow \bigcup$ text content of v.rightset
    $w_{ab} \longleftarrow \sqrt{s/l}$; $\Delta'(v) = \kappa_D(d_a, d_b)$
    $\kappa_A += w_{ab}\Delta'(v)$

4.4.2 Cover Tree Based Diversification

Cover Tree Overview

The cover tree [14] is a metric tree to index data and perform nearest neighbor search in metric spaces. It is a leveled tree where each level is a "cover" for the level beneath it. Each level is indexed by an integer scale i which starts from zero (root node) and increases as we descend the tree. For instance, a cover tree in Figure 4.3 indexes fifteen results of Example 1. Every answer tree repeats in the lower level after it first appears, so the lowest level contains all the answer trees.

Assume that we use the cover tree CT to index our answer set $\mathcal{T}$ based on answer tree distances, and C_i to indicate answer trees in $\mathcal{T}$ associated with the nodes at level i. Cover tree obeys three important properties for all levels $i \geq 0$:

- *Nesting*: $C_i \subseteq C_{i+1}$
- *Covering*: For every tree $T_a \in C_{i+1}$, there is a tree T_b such that $dist(T_a, T_b) \leq 1/2^i$ and exactly one such T_b is a parent of T_a.
- *Separation*: For all trees $T_a, T_b \in C_i$, the distance from T_a to T_b is greater than $1/2^i$.

Note that the cover tree definition in our case is different from the original definition in [14]. In contrast to the Euclidean distance without upper bound, the kernel distance between answer trees ranges from 0 to 1, so we assign the root of the cover tree as level 0 with the maximal distance coverage 1, and descend the coverage through the tree level by level.

In order to better illustrate the diversification on top of cover tree structure, next we summarize the procedure of the cover tree construction. The intuitive idea is to iteratively insert answer trees into the cover tree and also keep the three properties stated above. Each answer tree T is recursively inserted starting from level 0 until the highest possible level i such that T has the distance greater than $1/2^i$ to all the answer trees in level i, and is covered by the answer tree in level $i-1$ within distance

$1/2^{(i-1)}$. Take T_{10} in Figure 4.3 as an example. In level 0, it is covered by T_8 within distance 1, so that it drops to level 1. Similarly, it is further covered by T_4 and T_7 until it is inserted into level 3. The authors in [14] proved the correctness of this insertion. Besides, they also provide a batch construction which is empirically superior to a sequence of single point insertions. Readers are referred to the cover tree paper [14] for a comprehensive explanation.

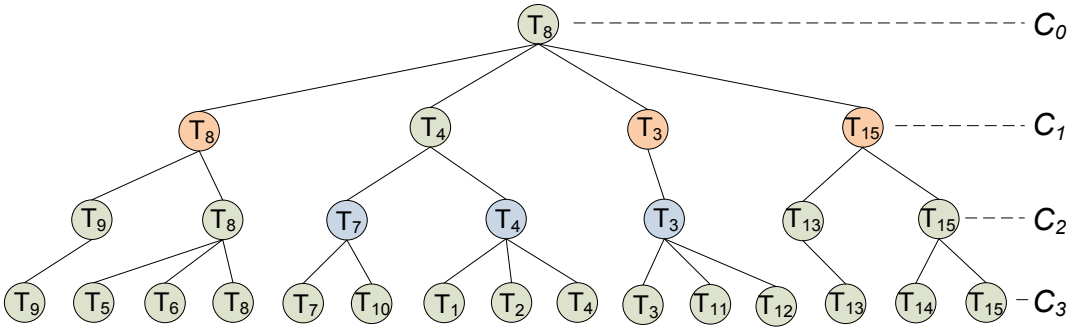


Figure 4.3: Cover Tree Example

Diversification on Cover Tree

We next describe a cover tree solution to find k diversity answers out of N answer trees. We assume $N > k$ throughout the paper, since it is trivial to return all the candidates as diverse results when $N \leq k$. The separation property of cover tree suggests that nodes at higher level are more diverse. Therefore, instead of discovering k diverse results from the whole answer set, we could make use of the cover tree to efficiently find good candidates for the result diversification problem. Unlike the nearest neighbor search on cover tree, we propose a greedy algorithm to meet our need. Intuitively, the idea is to discover diverse results in the highest possible level on the cover tree. As illustrated in Algorithm 5, we access cover tree one level at a time, and stop at the first level including at least k nodes, which is denoted as the working level C_i. If the size of C_i equals k, all the answer trees in this level are returned as diverse results. Otherwise, C_{i-1} is selected as the partial results, for that they are more separate in general according to the separation property. Next, we heuristically expand the farthest node in the working level until $|\mathcal{S}| = k$. When k is set to 3 for the cover tree in Figure 4.3, the algorithm proceeds as follows. At level 0, it only contains the root node. Then the algorithm continues to the next level with

four nodes. The number of nodes is larger than k, so this level becomes the working level and T_8 from the above level is selected as the partial result. Next, T_3 and T_{15} are further included by means of farthest expanding. Finally, we discover $\{T_8, T_3, T_{15}\}$ as diverse results for this running example.

In this algorithm, we first construct the cover tree index in $O(c^6 N \ln N)$ time for the expansion constant c [14]. The basic operation later in this algorithm is $setdist(T, \mathcal{S})$, i.e. $\sum_{T_a \in C_i} dist(T, T_a)$, which requires $|\mathcal{S}|$ distance computations. Since $|\mathcal{S}| \leq k$, we obtains its complexity as $O(k)$. This operation is performed $O(k|C_i|)$ times in the while loop. In the worst case, $|C_i|$ equals to $O(N)$. Combining the above two parts, the final complexity of this algorithm is $O(c^6 N \ln N + k^2 N)$ in terms of distance computations.

Furthermore, we propose an update method to support updating the k diverse results when the candidate set is progressively generated. The underlying idea is to check whether this newly added answer tree T_{new} affects the working level, and then adjust the k diverse results by means of swapping between T_{new} and T_{old} in original results. The $swapcost(T_{new}, T_{old})$ indicates the sum distance change when we replace T_{old} with T_{new} in $\mathcal{S}$, i.e. $setdist(T_{new}, \mathcal{S}) - setdist(T_{old}, \mathcal{S})$. The complexity of Algorithm 6 consists of two components. The first is the beginning insertion with a complexity of $O(\ln N)$ [14]. The following part has $O(k^2)$ complexity since the *swapcost* operation is performed $O(k)$ times. Thus, the total complexity of the update algorithm is $O(\ln N + k^2)$ in terms of distance computations.

The complexity of Algorithm 6 consists of two components. The first is the insertion at the beginning, which has a complexity of $O(\ln N)$ according to [14]. The basic operation of the following part is $swapcost(T_{new}, T_{old})$, which has the same complexity $O(k)$ as the *setdist* operation. Thus, this part has complexity $O(k^2)$ since the *swapcost* operation is performed $O(k)$ times. In summary, the total complexity of the second algorithm is $O(\ln N + k^2)$ in terms of distance computations.

To sum up, the cover tree based approach has several advantages. First, instead of diversifying results in the whole answer set, we utilizes the separation property to reduce the number of distance computations. Furthermore, cover tree supports progressive insertions with minor efforts. Finally, the tree-like structure makes it a great tool for hierarchical browsing, which will be further explained in Section 4.5.

Algorithm 5: **CoverTreeDiversification**

Input: Answer tree set $\mathcal{T}$, k
Output: The k diverse result set $\mathcal{S}$

build cover tree CT from answer tree set $\mathcal{T}$
// find the working level
$C_{i-1} \longleftarrow NULL$
$C_i \longleftarrow C_0$
while $|C_i| < k$ **do**
    $C_{i-1} = C_i$
    $C_i = C_{i+1}$
// discover k diverse results
if $|C_i| = k$ **then**
$\mathcal{S} \longleftarrow \bigcup$ all the answer trees in C_i
else
    $\mathcal{S} \longleftarrow \bigcup$ all the answer trees in C_{i-1}
    while $|\mathcal{S}| < k$ **do**
        find answer tree $T \in C_i \setminus \mathcal{S}$, s.t.
        $setdist(T, \mathcal{S}) = \max\{setdist(T, \mathcal{S}) : T \in C_i \setminus \mathcal{S}\}$
        $\mathcal{S} \longleftarrow \mathcal{S} \cup \{T\}$

Algorithm 6: **Update**

Input: Cover tree CT, Answer tree T_{new}, Result set $\mathcal{S}$
Output: The refined k diverse result set $\mathcal{S}'$

insert T_{new} into CT
if *working level* $C_i \in CT$ *is not changed* **then**
    if $|C_{i-1}| = k$ **then**
        $\mathcal{S}' \longleftarrow \bigcup$ all the answer trees in C_{i-1}
        set the working level to be C_{i-1}
    else $\mathcal{S}' \longleftarrow \mathcal{S}$
else
    $maxcost \longleftarrow 0$; $swaptree \longleftarrow NULL$
    foreach *answer tree* $T_{old} \in \mathcal{S}$ **do**
        if $swapcost(T_{new}, T_{old}) > maxcost$ **then**
            $maxcost \longleftarrow swapcost(T_{new}, T_{old})$
            $swaptree \longleftarrow T_{old}$
    if $maxcost > 0$ **then**
        replace $swaptree$ in $\mathcal{S}$ with T_{new}
        $\mathcal{S}' \longleftarrow \mathcal{S}$

4.4.3 Alternative Solutions

We propose two state-of-the-art alternative approaches to solve the diversification problem. One solution is adapted from the farthest expansion algorithm [37, 52]. It maintains two sets of trees: the answer tree set $\mathcal{T}$ and diverse result set $\mathcal{S}$. Initially, the size of $\mathcal{T}$ is N and the size of $\mathcal{S}$ is zero. The farthest answer trees are iteratively moved from $\mathcal{T}$ to $\mathcal{S}$ until $|\mathcal{T}| = N - k$ and $|\mathcal{S}| = k$, as shown in Algorithm 7. $setdist(T, \mathcal{S})$ in line 4 is the sum distance between answer tree T and all answer trees in $\mathcal{S}$, i.e. $\sum_{T_a \in \mathcal{S}} dist(T, T_a)$. Although guarantees a 2-approximation to Problem 2's optimal solution [52], this algorithm has complexity $O(N^2)$ in terms of distance computations, which is relatively high when the number of answer trees is large. One possible relaxation of the quadratic complexity is to randomly select the first result and expand to rest $k-1$ results. However, this method needs to select diverse results from the whole answer set and is sensitive to the first result, which needs multiple restarts to obtain a stable performance.

Another approach, the k-medoids clustering, is derived from CLARANS [92]. The idea is to cluster candidates into k groups and select medoids as k diverse results. The number of distance computations is $O(Ik(N-k)^2)$, where I is the number of iterations. This method suffers high computational cost. Furthermore, it also requires starting from multiple initial medoids to approach global optimal results. The detailed comparison among these algorithms will be shown in the experimental section.

Algorithm 7: **FarthestExpanding**

Input: Answer tree set $\mathcal{T}$, k
Output: The k diverse result set $\mathcal{S}$

1 find T_a, T_b, s.t. $dist(T_a, T_b) = \max\{dist(T_a, T_b) : T_a, T_b \in \mathcal{T}, T_a \neq T_b\}$
2 $\mathcal{S} \longleftarrow \{T_a, T_b\}$
3 **while** $|\mathcal{S}| < k$ **do**
4 find answer tree $T \in \mathcal{T} \setminus \mathcal{S}$, s.t. $setdist(T, \mathcal{S}) = \max\{setdist(T, \mathcal{S}) : T \in \mathcal{T} \setminus \mathcal{S}\}$
5 $\mathcal{S} \longleftarrow \mathcal{S} \cup \{T\}$

4.5 Result Representation

To improve the usability of the BROAD framework, we implemented a demo system [134] with an interactive visual interface, so that the user can explore the query results by means of hierarchical browsing.

4.5.1 Hierarchical Browsing

Hierarchical browsing is an effective approach to interact with users and can be elegantly supported by the cover tree structure. We proceed as follows. First, we separate answer trees in the working level into k answer tree groups $\mathcal{G}$ based on their kernel similarities to the k diverse results. A user then selects a subset $\mathcal{H}$ of interest. Second, we fetch all nodes in the next level covered by $\mathcal{H}$, and treat them as nodes in a new working level. Thus, we can perform Algorithm 5 again to obtain a new set of diverse results. This procedure iteratively proceeds until we obtain the intended answer tree/s. For instance, T_8, T_3 and T_{15} in Figure 4.3 are diverse results found previously. We first assign T_4 to T_3 due to the kernel similarity and these four answer trees form three groups. Assume that users are interested in the group $\{T_3, T_4\}$, so we drill down to the next level with answer tree set $\{T_3, T_4, T_7\}$. They are directly selected as new diverse answers because the size of this level equals to three.

4.5.2 Visual Interface

To support hierarchical browsing, we develop a circular view to summarize both structures and contents of a group of answer trees. As such, users can quickly browse and select preferred answer trees from the whole answer set. The basic idea is derived from the Circos project [73] and we adapt it for the answer trees' summarization. In the following, we take answer tree T_7 to show the process of mapping one answer tree into a circle. Figure 4.4a depicts T_7 and it is transformed to the red part in Figure 4.4b and 4.4c. The root node and keyword nodes are mapped to segments, and pathes between nodes are mapped to ribbons. Answer tree contents, which will be discussed later, are selected as representative words around the circle. We also support the focused view when a user chooses certain answer tree. It is displayed with

color and path structures, while other answer trees become transparent. To illustrate, T_7 in Figure 4.4c is shown in red and highlighted with the structure "author→paper" between the root node and the "skyline" keyword node. For a group of answer trees, the shared nodes among answer trees are presented just once to save space. For instance, Figure 4.4b only contains two "skyline" nodes and three "rank" nodes for eight answer trees. As a result, the circular view for a group of answer trees salvages large spaces compared to the original layout. Furthermore, we utilize different colors to distinguish answer trees so that users can quickly capture how many of them are covered in a group. In general, this view is suitable for keyword search, because k is usually much less than one hundred in real keyword search use cases.

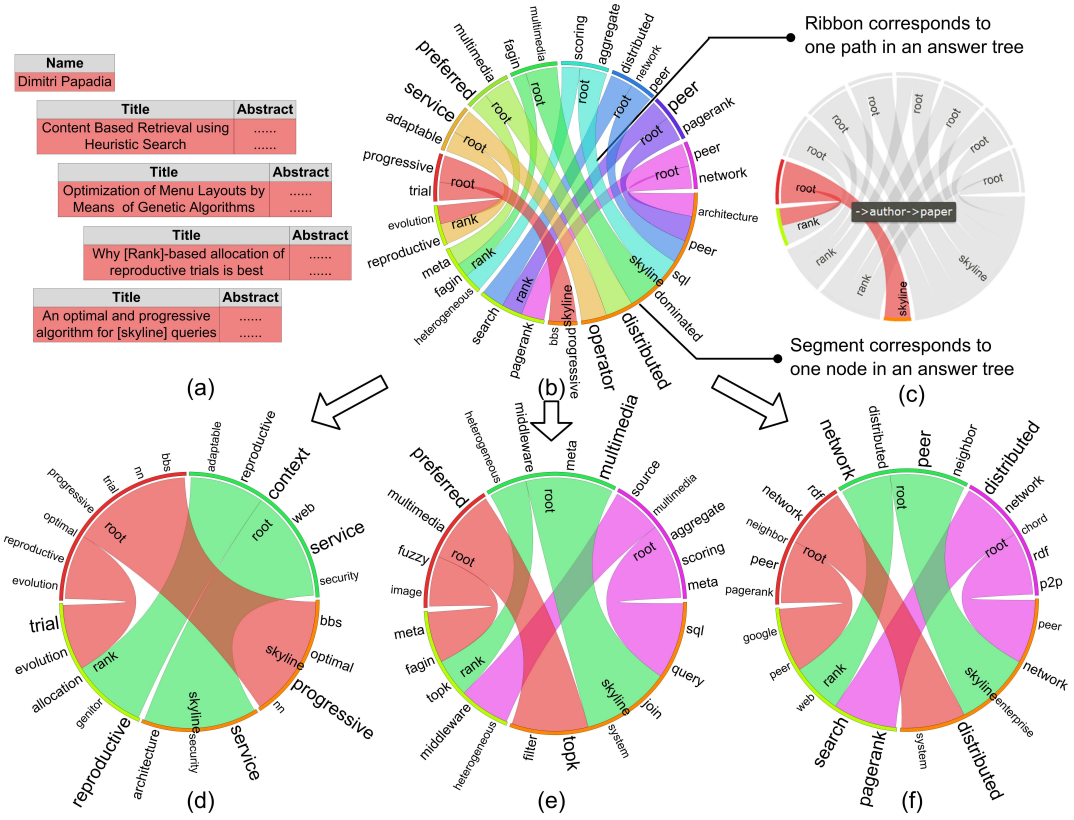


Figure 4.4: Result Representation

Aside from structural summarization, representative words $\mathcal{W}_r$ are attached to the related segments in order to distinguish the circles that are on the same level of the hierarchy. Given any segment s, let the node it represents be n. The ribbons that connect s to other segments in the circle represent paths in the answer trees that connect n to other nodes in the answer trees. For each segment, candidate words

Algorithm 8: **Word Selection**

Input: Segment s, Answer tree groups $\mathcal{G}$
Output: Representative words $\mathcal{W}_r$

```
1  obtain candidate words W_c of segment s
2  obtain group g that segment s belongs to
3  foreach candidate word w ∈ W_c do
4      T' ⟵ all the answer trees in g containing w
5      T ⟵ all the answer trees in g
6      w.CoverRatio ⟵ |T'|/|T|
7      G' ⟵ all the groups in G containing w
8      w.Frequency ⟵ |G'|/|G|
9      w.Score ⟵ w.CoverRatio × log(1/w.Frequency)
10 sort W_c with decreasing scores
11 select W_r proportional to the width of segment s
```

$\mathcal{W}_c$ are selected from these pathes. Candidates for the highlighted root segment in Figure 4.4c are all the words from nodes in T_7. We then obtain $\mathcal{W}_r$ from these candidates as in Algorithm 8. In short, we compute a TF-IDF like score for candidate words, and select top candidates as representative words. As such, we sketch out the distinct contents of answer trees. The number of representative words selected depends on the width of the segment. For the green root segment in Figure 4.4f, the words "network", "distributed", "peer" and "neighbor" are selected as representative words, since they have highest scores. To further emphasize the word distinctions within a segment, we present the selected words in different font sizes, according to their term frequencies in one segment. The words "network" and "peer" are highlighted with the biggest font size since their term frequencies are the largest in the segment.

The circular representation provides a summarized view both for structural and textual information about an answer tree group, which enhances the process of hierarchical browsing. In Figures 4.4d, 4.4e and 4.4f, we show three circles representing three groups of answer trees on the lower level of the browsing hierarchy. The left circle consists of two answer trees with one "rank" node and two "skyline" nodes. The content is mainly about the web services. The middle circle contains three answer trees. The major topic is the relationship between top-k query, skyline query and preference discovery. The right circle with three answer trees emphasizes the connection between skyline algorithm and distributed environment. In summary, cir-

cles can show distinct and summarized information about groups, which help users to browse and select desired answer trees. Note that circle view is a complement but not a substitution of presenting trees. Thus, we show both the circle view and the tree view in the demo. Users may quickly obtain the summary for a group of answer trees in the circle view. They can further know the detailed information in the answer tree view.

4.6 Demonstration

In demonstration, we develop a web based browsing interface[1] to support interactive diversified keyword search. As shown in Figure 4.5, the interface consists of a search input area and a result display area. Search input area on the top of the interface contains keyword input field, zoom in/out buttons and setting fields for user-specified parameters (k and n). Therefore, we enable user to search by keyword query as well as perform hierarchical browsing using zoom in/out buttons.

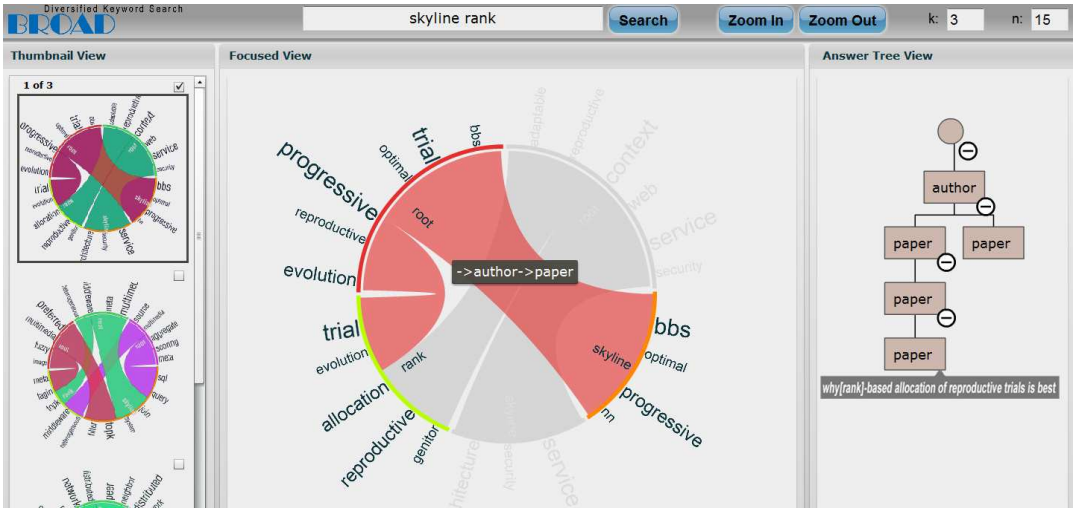


Figure 4.5: BROAD Interface

Result display area on the bottom is composed of three views from left to right: thumbnail view, focused view and answer tree view. The thumbnail view displays k circles to summarize k answer tree groups as a list of thumbnail images. Consequently, users can click the desired circle and enlarge it in the focused view, which

[1] http://db128gb-b.ddns.comp.nus.edu.sg:8080/broad/

allows users to focus on certain segments or ribbons. The chosen element is highlighted with color, while other elements become transparent. We also make use of a tooltip to describe the structure of selected path. Furthermore, the corresponding answer tree will be represented in the answer tree view. This view utilizes a tree layout to depict the answer tree structure in node labels, and depict the answer tree content in tooltips.

Take the keyword query {skyline,rank} as an example in Figure 4.5. A user sets $k = 3$ and $n = 15$ to discover three diverse answers from fifteen candidates. The thumbnail view shows a preview of three circles, so the user can browse the overview through a scroll bar and select the desired one to display in the focused view. For example, the top circle in the thumbnail view is selected with the detailed information in the center. Besides, T_5 is highlighted with red color because the user clicks on the root segment of T_5. Moreover, when mouse hovers over on the ribbon between the root node and the "skyline" keyword node, a tooltip "author→paper" shows the structure information. Correspondingly, the tree layout in the answer tree view visualizes the structure and the content of T_5. If the user ticks the top checkbox in the thumbnail view and then presses the zoom in button, the system can drill down to next level and present a set of new circles.

In summary, our BROAD system provides a user friendly interface that helps users search and explore diversified keyword search results. To the best of our knowledge, our work is the first attempt to support interactive hierarchical browsing on keyword search in databases.

4.7 Experiments

We present experimental studies to evaluate the BROAD system in this section. Without loss of generality, we implemented the state-of-the-art graph based keyword search algorithm [69] to discover candidate answer trees. It returns N candidates and we then discover k diverse answers out of it. Table 4.1 explains the parameters used throughout this section. It also shows the range and the default values (in bold) of the parameters. In each experiment, we adjust one parameter while keeping the other one at its default value.

4.7.1 Datasets and Queries

We use two real datasets to assess our system. One is CiteSeerX, a collection of scientific and academic papers focusing on computer and information science. We choose this dataset for two reasons: i) It maintains a large amount of paper abstracts as well as citations between papers; ii) It is a dataset for an online search engine associated with a query log. Another is Yago [110], a huge semantic knowledge base derived from Wikipedia and WordNet. Originally, Yago dataset is stored as a set of triples(subject,property,object). It contains several millon of entities and 88 property types between them. According to the entities' type attributes, we transform Yago to a traditional database storage by separating entities to different tables and connecting them by foreign key references. For instance, people entities become tuples in people table and may connect to tuples in location table by bornIn references. Statistics about the graphs generated from the datasets is shown in Table 4.2. As in Definition 4.2.2, we tune radius r with respect to different datasets to generate meaningful candidate set respectively.

Table 4.1: Parameter Settings

Parameter	Description	Range
N	answer tree set size $\|\mathcal{T}\|$	25, **50**, 75, 100
k	diverse result size $\|\mathcal{S}\|$	2, 4, **6**, 8, 10

Table 4.2: Dataset Statistics

Property	CiteSeerX	Yago
Node count	1,127,838	9,960,479
Edge count	3,414,540	16,666,533
Radius r	6	3

To obtain a reasonable query set, we adopt a two-stage procedure. In the first stage, we extract meaningful query terms for each dataset. For CiteSeerX, there is a query log which is dominated by short queries with no more than 2 keywords (> 94%). As such, we derive query terms from the log instead of directly using it. This is done by extracting terms with term frequencies larger than 10. For Yago, we extract ambiguous terms from wikipedia disambiguation pages [2]. Ambiguous terms refer to more than one topic. For example, "Healer" may refer to a film or a music album. We

[2] http://en.wikipedia.org/wiki/Wikipedia:Disambiguation

collect and use them as query terms. The second step is to generate keyword queries by randomly combining query terms. For query size l from 2 to 5, we produce 1000 initial queries for each value of l. In order to guarantee correctness, we test the queries using the keyword search engine, and filter queries that cannot produce enough answer trees. Then we rank the remaining queries according to the number of different keyword nodes in a descending order. Finally, we select the top-10 queries for each l, i.e. 40 queries per dataset.

4.7.2 Evaluation Metrics

In IR community, evaluating the accuracy of diverse query results is well studied and several evaluation metrics are established, such as S-recall and S-precision [127], α-NDCG [31], NDCG-IA [5] and so forth. The metrics extended from NDCG are not suitable for our problem, for these metrics rely on the result ranking. Therefore, we will evaluate our system based on S-Recall and S-Precision. However, we need to carefully adapt them for keyword search in databases. In general, most of these evaluation metrics are based on subtopics or nuggets, which indicate semantics covered by answers. Differently, in the context of database keyword search, we are required to capture both semantic information and structural information. Therefore, we consider substructures as a complement to subtopics.

We first generate subtopics for two datasets respectively. For CiteSeerX dataset, each paper is associated with a conference or a journal. We thus derived the topic information based on the research area of the conference or the journal. Note that author nodes may be related to multiple topics because they published papers to different research areas. Since entities in Yago dataset have type attributes derived from the wikipedia categories, we then utilized them to assign nodes with different subtopics. As for substructures, if the result set contains more different pathes from the root node to keyword nodes, it intuitively covers more diverse structural information. So we decompose each answer tree to l pathes from the root node to all the keyword nodes to evaluate the structural diversity. As a result, answer trees are reliably mapped to subtopics and substructures. Let subtopics_q and substructs_q for query q be the subtopics and substructures in N candidates, and subtopics(T) and substructs(T) be the relevant subtopics and substructures in answer tree T. We formally define S-recall in database keyword search as follows:

Definition 4.7.1 *(S-recall)*
Given k results for keyword query q,

$$\text{S-recall} = \alpha \cdot \frac{|\bigcup_{a=1}^{k} subtopics(T_a)|}{|subtopics_q|} + (1-\alpha) \cdot \frac{|\bigcup_{a=1}^{k} substructs(T_a)|}{|substructs_q|}$$

where $\alpha \in (0,1)$ is a parameter to balance semantic information and structure information. The above metric refers to the percentage of subtopics and substructures covered by one of the k results. However, it is trivial to achieve recall of 100% by returning all candidates in response to any query. Therefore we define S-precision as a complement to S-recall. The subtopics_k and substructs_k refer to the ideal size of subtopics and substructures in k results, assuming that all the keyword nodes contain distinct topics and l different pathes.

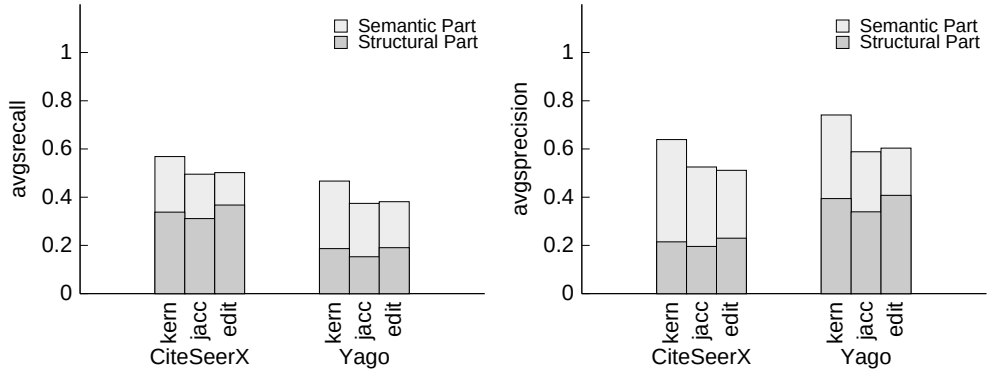
Definition 4.7.2 *(S-precision)*
Given k results for keyword query q,

$$\text{S-precision} = \alpha \cdot \frac{|\bigcup_{a=1}^{k} subtopics(T_a)|}{|subtopics_k|} + (1-\alpha) \cdot \frac{|\bigcup_{a=1}^{k} substructs(T_a)|}{|substructs_k|}$$

where $\alpha \in (0,1)$ is a balance parameter same as that in the definition of S-recall. In the following experiments, we set $\alpha = 0.5$ to treat semantic difference and structural difference equally. Besides taking substructures into consideration, our S-precision still differs from the S-precision in paper [127]. They defined S-precision based on S-recall. Given S-recall s_r, S-precision equals to $minRes(\mathcal{S}_{opt}, s_r)/minRes(\mathcal{S}, s_r)$. $minRes(\mathcal{S}, s_r)$ indicates the minimal size of results having S-recall s_r. This definition is not straightforward in the first place, since it is derived from S-recall s_r instead of result size k. Moreover, the computation is impractical due to the hardness of generating the optimal solution. These two reasons drive us to alter the definition of S-precision. Nevertheless, our definitions of S-recall and S-precision are natural analogy of the standard recall and precision measures.

4.7.3 Kernel Distance v.s. Other Distance Functions

In order to verify the effectiveness of the kernel distance, we compare it with two state-of-the-art distance functions: tree edit distance and Jaccard distance. The following figure shows the diverse result for three distances with the default parameter setting($N = 50$ and $k = 6$).



(a) S-recall (b) S-precision

Figure 4.6: Comparison of Distance Functions

This figure compares three distance functions on S-recall and S-precision for two datasets. We present the detailed components of S-recall and S-precision, i.e. the semantic part and structural part. As can be seen, Jaccard distance has better score on the semantic part while tree edit distance has better score on the structural part. Since kernel distance captures both semantic difference and structural difference, it shows much higher overall score than other two distance functions. Therefore, we utilized kernel distance to compare algorithms in the following.

4.7.4 Cover Tree Algorithm v.s. Other Algorithms

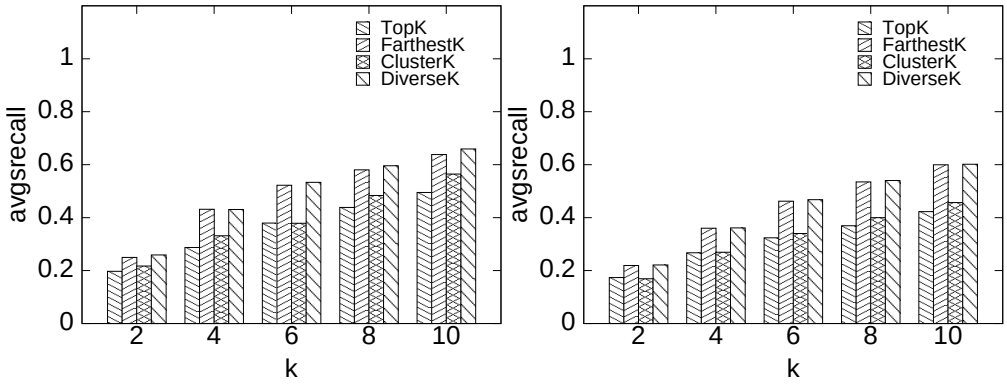
To assess cover tree diversification (*DiverseK*), we compare it with top-k candidate answers (*TopK*), Farthest Expanding algorithm (*FarthestK*) and Clustering algorithm (*ClusterK*).

Effectiveness Evaluation

We first vary k to study how it affects the effectiveness of four approaches. To begin with, we present the average S-recall in Figure 4.7. It is clear that this metric has an ascending trend with the increase of k for all schemes. Since $subtopics_q$ and $substructs_q$ remain the same for all schemes, S-recall is only dependant on the numerator parts of its definition. Their values increase as k increases, bringing about the ascent of S-recall. Nevertheless, we can easily notice their distinction in performance from the bar graph. $TopK$ performs worst because its ranking only relies on the relevance of the keyword query. Although *ClusterK* groups answer trees according to the similarity, it is not optimized to select pairwise different medoids, so it also produces low quality results. For $FarthestK$ and *DiverseK*, it can be seen that they acquire the best results since they both apply the greedy strategy to discover results with respect to Problem 2's objective. Comparing the two datasets, the S-recall for Yago in Figure 4.7b is lower than that of CiteSeerX in Figure 4.7a, since the number of $\mathcal{T}$'s subtopics and substructures in Yago is larger than that in CiteSeerX.

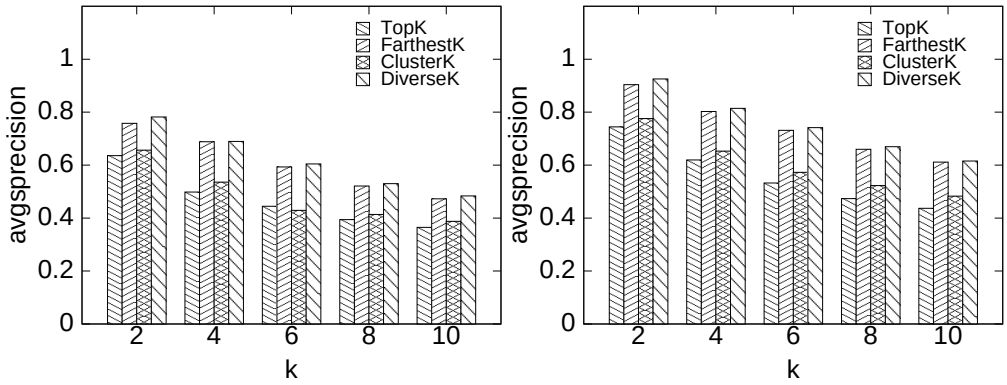
Figure 4.8 depicts the effect of k on the average S-precision for both datasets. The relative performances among all approaches are similar to the analysis for S-recall but the trend is negatively proportional to k. Our solution together with $FarthestK$ discovers the most diverse results among these four algorithms, since higher S-precision indicates smaller subtopic and substructure overlap between answer trees. Comparing Figure 4.8a and Figure 4.8b, we observe that the S-precision for Yago is higher than that of CiteSeerX. This again is due to the rich number of subtopics and substructures in $\mathcal{T}$, which results in Yago having a lower chance of obtaining overlapped result trees.

Next, we show the effectiveness measures by varying N. The comparisons with respect to average S-recall and average S-precision are illustrated in Figure 4.9,4.10 respectively. Like the case of varying k, our solution as well as *FarthestK* gives the best quality answers and outperforms *ClusterK* and *TopK* by 20% to 40%. This result shows the effectiveness of our solution. In Figure 4.9, S-recall decreases as N increases. This is because $|subtopics_q|$ and $|substructs_q|$ increase with N. In Figure 4.10, when $subtopics_k$ and $substructs_k$ remain the same with invariant k, S-precision is proportional to the numerator parts of the S-precision definition, i.e. the



(a) CiteSeerX (b) Yago

Figure 4.7: avg S-recall w.r.t. k



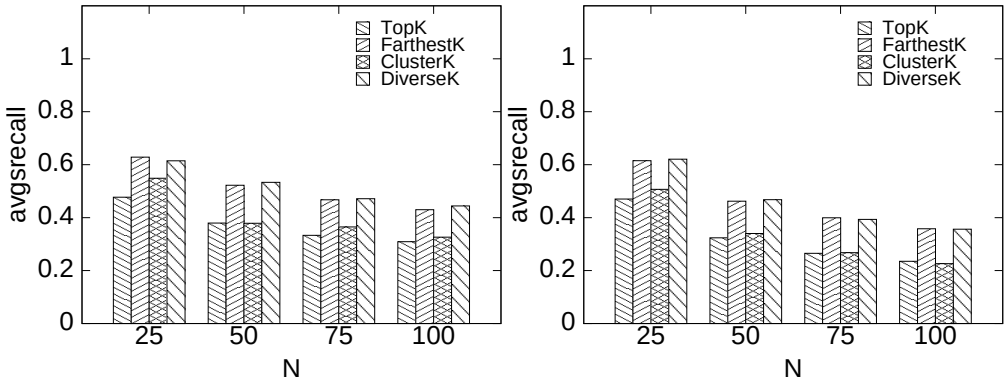
(a) CiteSeerX (b) Yago

Figure 4.8: avg S-precision w.r.t. k

coverage of the k results. The results of *TopK* have the same coverage. The coverage of *clusterK* has a small fluctuation, because the k medoids of clustering method are affected by the randomly selected initial medoids. For the other two algorithms, the quality increases with N. This observation is accordant with the intuition that we have more chances to discover better results as N increases.

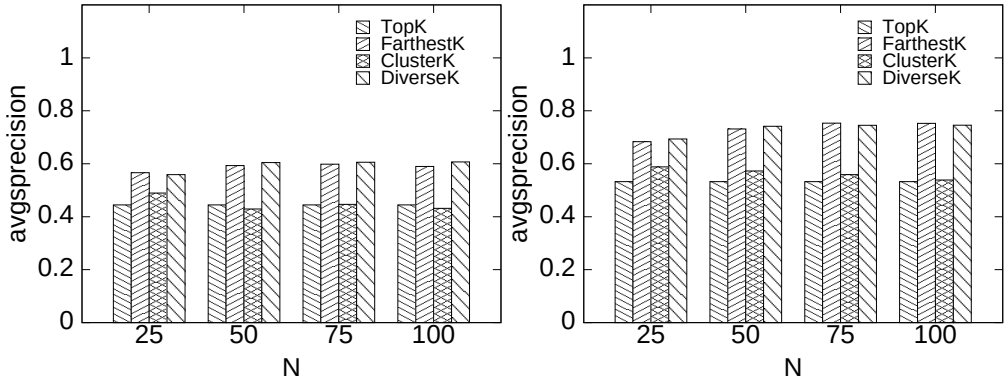
Efficiency Evaluation

We then report the efficiency results considering the response time for result diversification excluding the time to discover candidate set. This is because we used identical keyword search engine without affecting the comparison among algorithms. We



(a) CiteSeerX (b) Yago

Figure 4.9: avg S-recall w.r.t. N



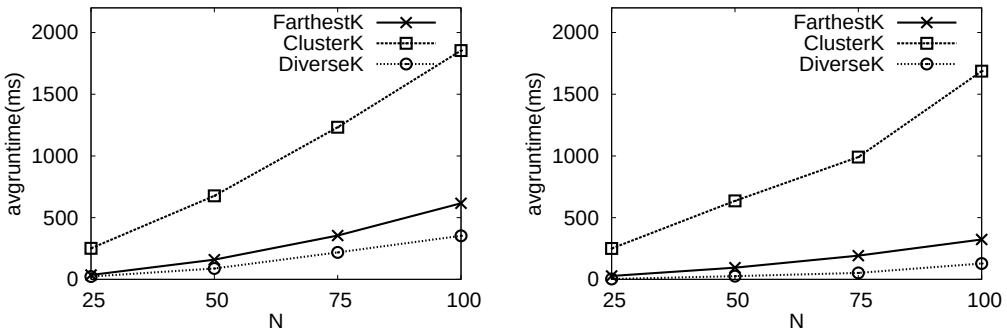
(a) CiteSeerX (b) Yago

Figure 4.10: avg S-precision w.r.t. N

also show the update cost for cover tree based solution to test its flexibility. In summary, by evaluating above metrics, we expect to investigate the overheads and gains of our framework. One important observation is the repeated distance computations waste a lot of time for all three algorithms. In our implementation, we cached the computed kernel distances in the main memory and reused them in case they needed again. Because each distinct distance is calculated only once, the running time can be improved. Note that this trade-off between time and space is meaningful because the answer tree set size N is remarkably small compared to the cardinality of the dataset.

Figure 4.11 displays the relationship between the running time and N. Although the response time for three algorithms increase at a super-linear trend, the cover

tree based algorithm is still at least twice faster than the *FarthestK*. *ClusterK* is the slowest one since it needs multiple iterations and each iteration takes quadratic time. Respecting to the running time for two datasets, we perceive that Yago queries are faster than CiteSeerX queries on average. The underlying reason is the average answer trees size of Yago is smaller than that of CiteSeerX. As a result, the individual distance computation takes less time for Yago dataset. Besides, the cover tree based solution supports dynamical updates as shown in Algorithm 6. This operation averagely takes 27ms for CiteSeerX and 15ms for Yago on the default parameter settings, which just incurs a small overhead compared to discovering from scratch. Readers are referred to our technical report [135] for a comprehensive efficiency comparison.



(a) CiteSeerX (b) Yago

Figure 4.11: avg Runtime w.r.t. N

In summary, we show that the BROAD system is both effective and efficient from the above experiments. The proposed kernel distance nicely captures the diversity of answers. Aiming at the same objective, the cover tree based solution is comparable to the *FurthestK* algorithm, which is much better than the clustering method and the original top-*k* answers. Also, it has the best response time and can dynamically update *k* diverse results instantly.

4.8 Summary

In this chapter, we have introduced BROAD, a novel system that integrates the discovery of diverse results with the current keyword search engine in databases. Unlike previous works, we proposed a new kernel distance metric between answer trees

which captures both structural and semantic information. Moreover, a cover tree based approach was developed in order to quickly and progressively return diverse results. To further consolidate this interesting framework, we provided a hierarchical browsing interface that helps navigate users in refining and browsing keyword search results. The outcomes from an experimental study demonstrated that the BROAD system can provide broad views of the answers that are returned by keyword search engine.

Chapter 5

Social Network Visual Analytics

5.1 Overview

In this chapter, we propose an analytic system which allows users to perform intuitive, visual browsing on a large scale social networks. There are two major challenges must be overcome:

First of all, how to improve the scalability is one potential challenge of cohesive subgraph discovery for social network analysis. Most of the existing approaches [116, 117, 128] mainly focus on the dense region recognition for moderate size graphs. However, many practical social network applications need to store the large scale graph in disks or databases. Like Facebook, over 800 million active actors use its service per month all over the world [11], which is impossible to fit in memory. Therefore, besides providing memory based solutions, we focus on developing a solution to handling a large scale social graphs stored in a graph database, which is more scalable for graph operations than a relational database. Like Twitter, recently it migrated its social graph to FlockDB [62], a distributed, fault-tolerant graph database for managing data at webscale. By leveraging graph databases, we extend memory based algorithms to I/O efficient solutions for large scale social networks.

Additionally, exploring and analyzing social network can be time consuming and not user-friendly. Visual representation of social networks is important for understanding the network data and conveying the result of the analysis. However, it is a

challenge to summarize the structural patterns as well as the content information to help users analyze the social network. One previous work [116] proposes a novel linear plot for graph structure, which sketches out the distribution of dense regions and is suitable for static dense pattern discovery. Unlike this work, our system insulates users from the complexities of social analysis by visualizing cohesive subgraphs and the contents in an interactive fashion. For graph structure, we propose an orbital layout to decompose the graph into a hierarchy with respect to the cohesive value, in which more important social actors are located in the center. Figure 1.4b shows an orbital layout for the graph in Figure 1.4a. Briefly speaking, this layout consists of four orbits with four different colors, in which the more cohesive vertices are located closer to the center. Like the 5-clique (a, b, c, d, f), all five vertices are in the innermost orbit. As for vertices size setting, ordering and edge filtering, we will explain them in details later. For the contents, we make use of tag cloud technique to summarize the major semantics for a group of social actors. Generally speaking, our visualization is flexible and can be easily applied to other cohesive graph concepts.

In this work, we develop a novel social network visual analytic framework for large scale cohesive subgraphs discovery. Our contributions are summarized as follows:

- We have introduced a novel cohesive subgraph concept to capture the intrinsic feature of social network analysis nicely.
- By leveraging graph databases, we have devised an offline algorithm to compute global cohesive subgraphs efficiently. Moreover, we have developed an online algorithm to further refine local cohesive subgraphs based on the results of offline computations.
- We have developed an orbital layout to decompose the cohesive subgraph into a set of orbits, and coupled with tag cloud summarization, which allows users to locate important actors and their interactions inside subgraphs clearly.
- We have conducted extensive experiments, and the results show that our approach is both effective and efficient.

The rest of the paper is organized as follows. Section 5.2 defines the cohesive subgraph discovery problem handled throughout this work. Section 5.3 presents the

offline computations in the graph database. The online visual analytic system is described in Section 5.4 and followed by a demonstration in Section 5.5. Our extensive experimental study is reported in Section 5.6. Section 5.7 concludes the paper.

5.2 Problem Definition

In this section, we first introduce the preliminary knowledge, then define the maximal k-mutual-friend finding problem, and show several important properties about this concept. Furthermore, we compare it with clique, k-core, DN-Graph as well as truss decomposition in depth.

5.2.1 Preliminaries

As stated in Section 5.1, we model a social network as an undirected, simple **social graph** $G(V, E)$ in which vertices represent social actors and edges represent interactions between actors. The k-mutual-friend subgraph proposed in this work is derived from a clique and k-core [102]. Clique is a fully connected subgraph, in which every pair of vertices is connected by an edge. If the size of a clique is c, we call the clique a c-clique. k-core is one successful degree relaxation of clique concept defined as follows.

Definition 5.2.1 *(k-core Subgraph)*
A k-core is a connected subgraph g such that each vertex v has degree $d(v) \geq k$ within the subgraph g.

The k-core is motivated by the property that every vertex has degree $d(v) = c - 1$ in a c-clique. k-core also needs to satisfy the degree condition, but the restriction on subgraph size is not required. As such, k-core can be efficiently computed in $O(|E|)$ time complexity [102]. Differently, based on the observation in Section 5.1, we propose the k-mutual-friend subgraph to emphasize on tie strength. One important property about edges in a clique is that every edge is supported by $Tr(e) = k - 2$

triangles in a k-clique. Analogous to the k-core definition, the k-mutual-friend sets a lower bound for every edge's triangle count. Next we will formally define the k-mutual-friend and show its relationships to other cohesive structures.

5.2.2 The k-mutual-friend Subgraph

Definition 5.2.2 *(k-mutual-friend Subgraph)*
A k-mutual-friend is a connected subgraph $g \in G$ such that each edge $e(u, v)$ is supported by at least k other vertices which connect to both vertex u and vertex v within g. The k-mutual-friend number of this subgraph, denoted as $\mathcal{M}(g)$, equals k.

Note that we need to exclude the trivial situation to consider a single vertex as a mutual-friend. Given the parameter k, we may discover many k-mutual-friend subgraphs that overlap with each other. In the worst case, the number of k-mutual-friend subgraphs can be exponential to the graph size. Therefore, we further define the maximal k-mutual-friend subgraph to avoid redundancy.

Definition 5.2.3 *(Maximal k-mutual-friend Subgraph)*
A maximal k-mutual-friend subgraph is a k-mutual-friend subgraph that is not a proper subgraph of any other k-mutual-friend subgraph.

To compare with clique and core, we present two interesting properties about the k-mutual-friend subgraph.

Property 5.2.1 *Every $(k + 2)$-clique of G is contained in a k-mutual-friend of G.*

Proof 5.2.1 *Since a $(k + 2)$-clique is a fully connected subgraph with order $k + 2$, each edge is supported by k triangles. Therefore, it is contained in a k-mutual-friend subgraph by Definition 5.2.2.*

Property 5.2.2 *Every k-mutual-friend of G is a subgraph of a $(k + 1)$-core of G.*

Proof 5.2.2 *For each vertex v in g_k, it connects to at least k triangles. Every triangle adds one neighbor vertex to v except the first adding two neighbors, so that v has $(k+1)$ neighbors, i.e. $d(v) \geq (k+1)$. Therefore, g_k qualifies as a $(k+1)$-core of G.*

The above two properties suggest one important observation: $(k+2)$-clique $\subseteq$ k-mutual-friend $\subseteq$ $(k+1)$-core, showing that the mutual-friend is a kind of cohesive subgraph between the clique and the core. Note that the reverse of the above two properties are not true. Again in Figure 1.4, the 4-clique (m, n, p, q) is a subgraph of the 2-mutual-friend (m, n, p, q, t, u), while 2-mutual-friend (a, b, c, d, e, f) and (m, n, p, q, t, u), both of them are contained in the 3-core $(a, b, c, d, e, f, m, n, p, q, t, u)$. Finally, we define the main problem we investigate in this work as follows.

Problem 3 *(Maximal k-mutual-friend Subgraph Finding)*
Given a social graph $G(V, E)$ and the parameter k, find all the maximal k-mutual-friend subgraphs.

Comparison to *DN*-Graph

Before we illustrate the solution to Problem 3, we further state an interesting connection between the mutual-friend concept and the *DN*-Graph concept proposed by Wang et al. [117] recently. A *DN*-Graph, denoted by $G'(V', E', \lambda)$, is a connected subgraph $G'(V', E')$ of graph $G(V, E)$ that satisfies the following two conditions: (1) Every connected pair of vertices in G' shares at least λ common neighbors. (2) For any $v \in V \backslash V'$, $\lambda(V' \bigcup \{v\}) < \lambda$; and for any $v \in V'$, $\lambda(V' - \{v\}) \leq \lambda$.

At the first glance, *DN*-graph is similar to the maximal k-mutual-friend subgraph. However, these two concepts are distinct due to the second condition in *DN*-Graph definition. Intuitively, the *DN*-graph defines a strict condition that the maximal subgraphs need to reach the local maximum even for adding or deleting only one vertex. On the other hand, the maximal k-mutual-friend defines the local maximal subgraph that is not a proper subgraph of any other k-mutual-friend subgraph. As demonstrated in Figure 1.4a, (m, n, p, q), (p, q, t, u) and (m, n, p, q, t, u) are all *DN*-Graphs with $\lambda = 2$, since the λ value can only decrease if adding or removing any vertices. However, only (m, n, p, q, t, u) is the maximal 2-mutual-friend since other two are

its subgraphs. This example shows that the *DN*-Graph finding may generate many redundant subgraphs. Furthermore, due to the hardness of satisfying the second condition, solving the *DN*-Graph problem is NP-Complete as proven by the authors. To solve it they iteratively refine the upper bound for each edge to approach the real value, but it still has high complexity and is not suitable for large scale graph. Actually, the mutual friend finding is inspired by the *DN*-Graph concept and we improve it by providing efficient solution in polynomial time subsequently.

Comparison to Truss Decomposition

Truss decomposition is a process to compute the k-truss of a graph G for all $2 \leq k \leq k_{max}$, in which k-truss is a cohesive subgraph ensures that all the edges in it are supported by at least $(k-2)$ triangles [115]. The truss definition is similar to but proposed independently with the mutual friend defined in this work except the meaning for k. Besides, the authors for truss decomposition realize that memory solution can not handle large scale social networks. They develop two I/O efficient algorithms. One is a bottom-up approach that employs an effective pruning strategy by removing a large portion of edges before the computation of each k-truss. The second one takes a top down approach, which is tailor for applications that prefer the k-trusses of larger values of k. Differently, we store the social graph in graph database that is scalable for graph traversal based algorithms.

5.3 Offline Computations

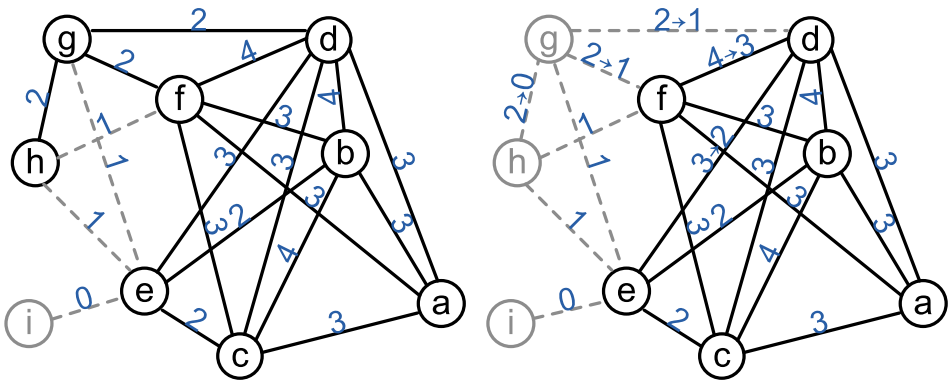
In this section, we first propose memory based solutions to solve Problem 3 in polynomial time, and then leverage the graph database to extend the solution for large scale social network analysis.

5.3.1 Memory Based Solution

Given a social graph G and the parameter k, the intuitive idea of discovering the maximal k-mutual-friend is to remove all the unsatisfied vertices and edges from G.

Based on the Definition 5.2.2, we iteratively remove edges that are not contained in k triangles until all of them satisfy the condition $Tr(e) \geq k$. The procedure is illustrated in Example 2.

Example 2 *Considering a maximal k-mutual-friend finding with* $k = 2$ *over the graph in Figure 5.1a, the left part of Figure 1.4a. First, edges* $\{(e,i),(e,h),(e,g),(f,h)\}$ *are removed since their triangle counts are less than* 2*. Next,* $\{(d,g),(f,g),(g,h)\}$ *are further removed since their triangle counts become less than* 2*, while* $e(d,e)$ *is still part of the* 2*-mutual-friend due to* $Tr(e(d,e)) = 2$*. In the third loop,* $Tr(e(d,f))$ *reduces to* 3 *but still satisfies the condition. Because all the remaining edges with triangle counts larger than or equal to* 2*, the graph remains unchanged and the loop terminates. Lastly, we delete all the isolated vertices and obtain* 2*-mutual-friend* (a,b,c,d,e,f) *as in Figure 5.1b.*



(a) Step one (b) Step two

Figure 5.1: Example of in Memory Algorithm

Although this is a straight forward solution, the computational complexity is relatively high because it has lots of unnecessary triangle computations. In the worst case it removes one edge at a time and needs $|E|$ times loops to remove all the edges from G. As such, the total complexity is $|E|\times\sum_{e(u,v)\in G}(d(u)+d(v))$, in which $d(u)+d(v)$ is the complexity to compute the triangle count for one edge. This expression can be further simplified to the order of $|E| \times \sum_{v\in G} d(v)^2$, because we need to get the v's neighbors $d(v)$ times in one loop. For practical case, we seldom encounter this extreme situation, but a large number of iterations is still a bottleneck of this solution.

As such, we propose an improved algorithm based on the following observation. When an edge is deleted, it only decreases the triangle counts of the edges which are forming triangles with that edge. Thus we can obtain edges affected by the deleted edge and only decrease triangle counts for them. This intuition is reflected in Algorithm 9, which can be divided into three steps. First, one necessary condition for $Tr(e(u,v)) \geq k$ is $d(u) \geq k+1$ and $d(v) \geq k+1$ as in the proof of Property 5.2.2. This is a lightweight method of deleting many vertices and their adjacent edges before removing unsatisfied edges with insufficient triangles. The remaining graph is then processed by the second step, which costs most of the workload to remove edges not supported by at least k triangles. From line 6 to 9, we first check all the edges' triangle counts. The Q is implemented as a hash set to record non-redundant removed edge elements. Next, instead of computing the triangle on all the edges to check the stability of the graph, we iteratively retrieve the affected edges from Q until Q is empty. This is the indicator that the graph becomes unchanged. Finally, the removal of inadequate edges likely results in isolated vertices, which are removed in the end. We show the procedure in the running example as follows.

Example 3 *We consider a maximal* 2*-mutual-friend finding in Figure 5.1a again based on Algorithm 9. According to the degree condition, we first remove vertex* i *and the edge* (e,i) *since the degree of* i *is less than* 3*. We then check the edge's triangle counts and delete* $\{(e,g),(e,h),(f,h)\}$*. Moreover, we record these edges in* Q *for affected edges. Edges* $\{(d,g),(f,g),(g,h)\}$ *are further removed until* Q *is empty. Finally, we delete all the isolated vertices and generate the same result as in Example 2.*

We next prove the correctness of Algorithm 9 in two aspects. On one hand, the remaining vertices and edges are part of the maximal-k-mutual-friend subgraphs. This aspect is true according to the definition of k-mutual-friend subgraph. On the other hand, the removed vertices and edges are not part of the maximal-k-mutual-friend subgraphs. Because the only modification on G is the removal of edges, bringing about the decrease of triangle counts, the edges supported by less than k triangles can be safely deleted since they cannot be part of a k-mutual-friend subgraph any more.

As for complexity analysis, the improved algorithm outperforms the naive one remarkably because it avoids a great deal of unnecessary triangle computations. The

Algorithm 9: **Improved k-mutual-friend**

```
   Input: Social graph G(V, E) and parameter k
   Output: k-mutual-friend subgraphs
   // filter by degree of vertices
 1 foreach v ∈ V do
 2     if d(v) < k + 1 then
 3         remove v and related e from G
   // delete edges with insufficient triangles
 4 initialize a queue Q to record removed edges
 5 initialize a hash table Tr to record triangle counts
 6 foreach e = (u, v) ∈ E do
 7     compute Tr(e) based on N(u), N(v)
 8     if Tr(e) < k then
 9         enqueue e to Q
10 while H ≠ ∅ do
11     dequeue e from Q
12     find out edges E′ forming triangles with e
13     remove e from G
14     foreach e′ ∈ E′ do
15         Tr(e′) − −
16         if Tr(e′) < k then
17             enqueue e′ to Q
   // delete isolated vertices
18 foreach v ∈ G do
19     if d(v) == 0 then remove v from G
20 return G
```

first step takes $O(|V|)$ complexity to check vertices' degree. The second step dominates the whole procedure. The initial triangle counting has time complexity $\sum_{v \in G} d(v)^2$. From line 10 to 17, finding all the edges forming triangles with the current edge $e(u, v)$ takes $d(u) + d(v)$ work. In the worst case, all the edges are removed from Q. Since Q only stores each edge one time, the total cost is $\sum_{e(u,v) \in G}(d(u) + d(v))$, equal to $\sum_{v \in G} d(v)^2$. The last step also takes $O(|V|)$ complexity to delete isolated vertices. As a whole, the total time complexity is $O(\sum_{v \in G} d(v)^2)$. It not only avoids the unnecessary iterations, but also reduces the graph size with relative small effort in the first step. Although the above algorithm is efficient, but is not suitable for large scale graph processing stored in disk. Retrospect the algorithm, it needs $O(|E|)$ space

Figure 5.2: Graph Database Storage Layout

complexity, which is too large to store in memory. So we extend it to the disk based solution in the following section.

5.3.2 Solution in Graph Database

In this section, we first introduce the concept of graph database, and then present a streaming solution in graph database and improve it by means of partitioning.

The graph database

A graph database [97] represents vertices and edges as a graph structure instead of storing data in separated tables. It is designed specifically for graph operations. To this end, a graph database provides index-free adjacency that every vertex and edge has a direct reference to its adjacent vertices or edges. More explicitly, there are two fundamental storage primitives: vertex store and edge store, which layouts are shown in Figure 5.2. Both of them are fixed size records so that we could use offset as a "mini" index to locate the adjacency in the file. Vertex store represents each vertex with one integer that is the offset of the first relationship this node participates in. Edge store represents each edge with six integers. The first two integers are the offset of the first vertex and the offset of the second vertex. The next four integers are in order: The offset of the previous edge of the first vertex, the offset of the next edge of the first vertex, the offset of the previous edge of the second vertex and finally the offset of the next edge of the second vertex. As such, edges form a doubly linked list on disk, so that this model possesses a significant advantage: there is a near constant time cost for visiting adjacent elements in a graph in some algorithmic fashion. This is actually a primitive operation in graph-like queries or algorithms, naturally

suitable for shortest path finding, maximal connected subgraph problem and graph's diameter computations and so on. Furthermore, it can scale more naturally to large data sets as they do not typically require expensive join operations.

Instead, the typical way to store graph data in relational database is to create edge table with index on vertices:

```
CREATE TABLE Edge (
    1stNode int NOT NULL,
    2ndNode int NOT NULL
)
CREATE INDEX IndexOne ON Edge (1stNode)
CREATE INDEX IndexTwo ON Edge (2ndNode)
```

Based on the above schema, we need to use index to support graph traversal since we cannot directly obtain the adjacent elements from the table. Example 4 shows a comparison between graph database and relational database.

Example 4 *Consider the process of the triangle counting. Given* $e(u, v)$*, we need to fetch* $N(u)$ *and* $N(v)$*. In relational database, we can utilize vertices to query the edge table index with* $O(\log |V|)$ *I/O cost, and then compute the shared neighbors as the triangle count. This procedure can be largely improved in graph database. According to the edge store, we can retrieve* $N(u)$ *and* $N(v)$ *as the traversal in the double linked list.* prevEdge *and* nextEdge *in Figure 5.2 provide reference to all the neighbors of vertices u and v, so that we can finish this step with* $O(d(v))$ *I/O cost, which is invariant to the graph size.*

Later in this section, we make use of the traversal operator extending the in memory algorithm to I/O-efficient algorithms in a graph database. We define the traversal operator as $traverse(elem, step)$ for better demonstration, which means that the length of shortest paths from graph element *elem* to the satisfied results cannot be larger than $step$. For example, $traverse(u, 1)$ retrieves all the vertices that are directly connected to u and the edges among them. For implementation, we utilize the Neo4j[1]

[1] http://neo4j.org

graph database, which is build on the graph storage layout in Figure 5.2. Note that we could easily migrate our algorithms to other popular graph databases as long as they are optimized for graph traversal, such as DEX[2], OrientDB[3] and so forth.

Streaming based solution

The streaming based solution is modified from Algorithm 9 and implemented in the graph database. The major changes are two-fold. On one hand, we use graph traversal to access vertices and edges (line 1 and 3), as well as compute triangle counts (line 5 and 6). On the other hand, we build index on edge attributes to mark edges as deleted (line 7, 9 and 15) and record edges' triangle counts (line 8, 13 and 14). Note that the edge attributes are in the order of $O(|E|)$, so they still need to be maintained out of core for large graph datasets. In this way, we make full use of the graph database, and keep all the advantages in the improved memory algorithm.

We next analyze the I/O cost in this algorithm. Filtering by degree and deleting isolated vertices need $O(|E|)$ I/O. The most costly part is removing edges with insufficient triangles. For edge (u,v), finding triangle count takes $O(d(u)+d(v))$ I/O work. Similar to the analysis for memory based algorithm, each edge can only be marked as deleted once. We conclude that this step needs $O(\sum_{v\in G} d(v)^2)$ I/O cost, which is also the total order of I/O consumptions. Besides, the traversal on vertices and edges is dominated by sequential I/O, which further reduces the I/O cost.

Partition based solution

Since all the triangle computations are directly operated in graph database, the streaming algorithm fails to make full use of the memory. Therefore, we proposed an improved approach based on the graph partitioning, and load partitions into memory to perform in memory triangle computations to save I/O cost and improve efficiency. To begin with, we derive a greedy based partitioning method in Algorithm 11 from the heuristics in paper [108]. The basic idea is to streamingly process the graph and

[2]http://www.sparsity-technologies.com/dex
[3]http://www.orientechnologies.com

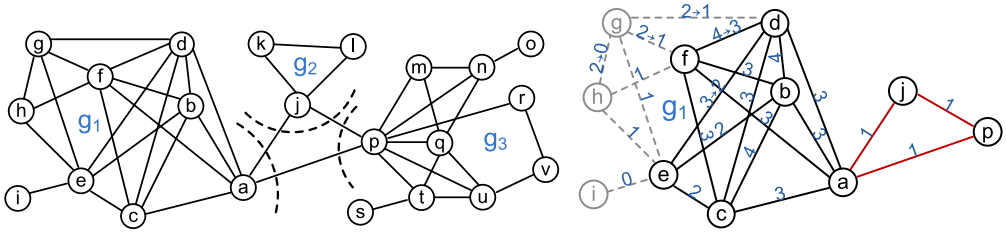
Algorithm 10: **Streaming based Algorithm**

Input: Social graph $G(V, E)$ and parameter k
Output: k-mutual-friend subgraphs
`// filter by degree of vertices`
traverse the vertices of G
remove v and related edges if $d(v) < k + 1$
`// delete edges with insufficient triangles`
traverse the edges E of G
foreach $e = (u, v) \in E$ **do**
    $N(u) \longleftarrow traverse(u, 1)$;$N(v) \longleftarrow traverse(v, 1)$
    compute $tr(e)$ according to $N(u), N(v)$
    if $Tr(e) < k$ **then** mark e as *deleted*
    else set e's mutual number attribute as $Tr(e)$
while *exist edges* $e(u, v)$ *marked as deleted* **do**
    $E' \longleftarrow$ edges form triangles with e in traverse(e,1)
    remove e from G
    foreach $e' \in E'$ **do**
        $Tr(e') - -$
        if $Tr(e') < k$ **then**
            mark e' as *deleted*
delete isolated vertices from G
return G

then assign every vertex to the partition where it has the largest number of edges connecting to. As in line 11 in Algorithm 11, *localPartitionNum* records the number of edges in each partition, $(1 - |g_i| \times p/|G|)$ suggests that partitions with larger size have smaller weight, and the product of the above two factors decides which partition the current vertex belongs to. This algorithm, requiring one breadth first graph traversal, is efficient with linear I/O complexity. However, the resulting partitions cannot be directly used because this algorithm is a vertex partitioning. Typically, it only extends partitions by including all the vertices connecting to the vertices inside the partition, which may result in the loss of triangles. As in Figure 5.3a, the running example is partitioned into three parts $\{g_1, g_2, g_3\}$. In this case, the triangle (a, j, p) is missing since its vertices are separated into three partitions. In order to keep all the triangles, we define an induced subgraph as in Definition 5.3.1.

Definition 5.3.1 *(Induced Subgraph)*
Denote $g_i+ = (V_i+, E_i+)$ *as an induced subgraph of a partition* $g_i(V_i, E_i)$ *of* G. *The*

extended vertex set is defined as $V_i+ = V_i \bigcup \{v : u \in V_i, v \in V \backslash V_i, (u,v) \in E\}$. *The extended edge set is defined as* $E_i+ = \{(u,v) : (u,v) \in E, u \in V_i\} \bigcup \Delta E_i$. *where* ΔE_i *are edges satisfying* $\{(v,w) : u \in V_i$,
$(u,v),(u,w) \in E, v.partition \neq w.partition, u.id < v.id$,
$u.id < w.id\}$.



(a) Partition into $\{g_1, g_2, g_3\}$ (b) Computation on g_1

Figure 5.3: Example of Partition based Algorithm

Based on the induced subgraph, the triangle (a, j, p) in Figure 5.3a is allocated in g_1 as shown in Figure 5.3b, because id a is smaller than j, p in this triangle. Next we formally prove the correctness of the partitioning method in Lemma 5.3.1.

Lemma 5.3.1 *Induced subgraphs* $\{g_1, \ldots, g_p\}$ *derived from* p *partitions of* G *have the same set of triangles as* G.

Proof 5.3.1 *The lemma is equivalent to the statement that every triangle* (u,v,w) *in* G *appears once and only once in all partitions. The proof can be divided into three cases. If three vertices belong to* V_i *of partition* i, *the triangle can only be inside the same partition. If any two of three vertices belong to* V_i *of partition* i, *without loss of generality, we assume that* $u, v \in V_i$ *and* $w \in V_j$. *The triangle is in partition* i *but not in partition* j, *since* (u,v) *can only be assigned to partition* i. *If three vertices are located in different partitions, we assign the triangle to the vertex with smallest id as defined in* ΔE_i, *so this triangle only appears once in induced subgraphs.*

Finally, we provide a partition based solution in Algorithm 12. First we partition the graph into p partitions, and for each partition, we do the in memory edge removal. Note that we only consider inside edges, which only affect triangles satisfying $\{(u,v,w), u,v,w \in V_i\}$. As such, we make use of the memory to reduce the

Algorithm 11: **Graph Partitioning**

Input: Social graph $G(V, E)$, partition number p
Output: $\{g_1, \ldots, g_p\}$ partitions

```
1  foreach v ∈ G in BFS order do
2      if d(v) < k + 1 then
3          remove v and related edges; continue
4      initialize the array localPartitionNum with size p
5      N(v) ⟵ traverse(u, 1); foreach u ∈ N(v) do
6          ind ⟵ u's partition index
7          if ind > 0 then localPartitionNum[ind]++
8      maxWeight ⟵ 0; curWeight ⟵ 0
9      pIndex ⟵ −1
10     for i from 1 to p do
11         curWeight ⟵ localPartitionNum[i] × (1 − |g_i| × p/|G|)
12         if curWeight > maxWeight then
13             maxWeight ⟵ curWeight
14             pIndex ⟵ i
15     set v's partition index as pIndex
16 return G
```

graph size as well as keeping the correctness of the solution. After this, we write the induced subgraphs back to graph database and use Algorithm 10 to do post processing. We take the induced subgraph g_1 in Figure 5.3b to find 2-mutual-friend subgraph. Note that edges $\{(a, j), (a, p), (j, p)\}$ are outside edges, while others are inside edges. For inside edges, we directly apply in memory algorithm and remove edges in dotted lines with triangle counts less than 2. But for outside edges, we cannot delete them since they may affect triangle counts in other partitions. After we deal with all the partitions, we post process the refined graph using Algorithm 10 to obtain the final result. In the worst case, this algorithm has the same I/O complexity as Algorithm 10. But in practice, it loads and processes the induced subgraphs to memory and avoids many disk triangle computations. The detailed comparison between this two disk-based solutions will be presented in the experimental section.

Algorithm 12: **Partition based Algorithm**

Input: Social graph $G(V, E)$, parameter k, and partition number p
Output: k-mutual-friend subgraphs

```
1  partition the graph based on Algorithm 11
2  for i from 1 to p do
3      load induced subgraph g_i+ into memory from the partition i
       // Do in memory edge removal
4      queue Q ⟵ ∅
5      hash table Tr ⟵ ∅
6      foreach e = (u, v) ∈ E_i+ ∧ e is inside do
7          compute Tr(e) based on N(u), N(v)
8          if Tr(e) < k then
9              enqueue e to Q
10     repeatly remove inside edges until Q is empty
11     write g_i+ back to the graph database
12 use Algorithm 10 to do post processing
13 return G
```

5.4 Online Visual Analysis

Based on the algorithms proposed in the previous section, we develop a client-server architecture to support online interactive social visual analysis. As in Figure 5.4, the offline computations are the base for the online visual analysis. For online analysis, we retrieve a local subgraph g close to the user selected vertex on top of offline computing result, online compute the exact $\mathcal{M}$ values for graph elements inside g, and generate the orbital layout for visualization. Moreover, we select representative tags to summarize the textual information in the local graph. In the client side, users can search and browse the visualized subgraph.

5.4.1 Online Algorithm

Based on the offline computations, we retrieve a local subgraph associated with the input keywords from graph database and compute exact $\mathcal{M}$ values for every edge and vertex inside the subgraph. This is a fundamental step to support graph layout later in this section. User can select a focused vertex v from a list of vertices containing

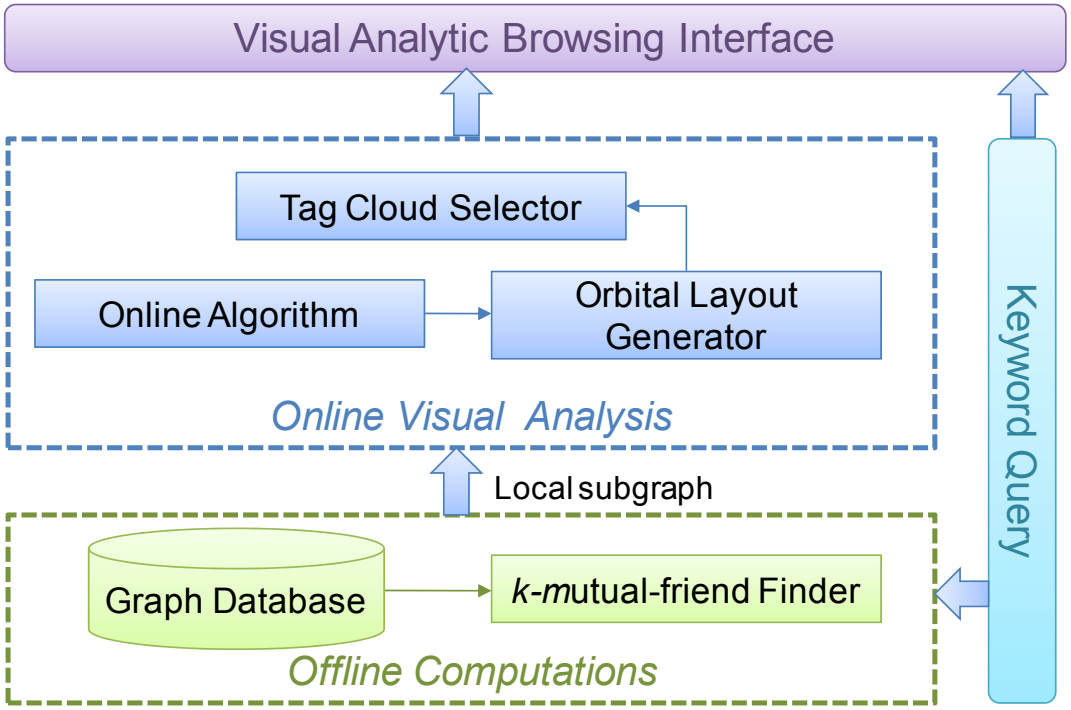


Figure 5.4: Social Network Visual Analytic System

the keywords, and our system will return a local subgraph including all the vertices within the distance τ from v and the edges among these vertices, i.e. $traverse(v, \tau)$. For efficient online computation, we show one important stability property of the k-mutual-friend subgraph as follows.

Property 5.4.1 *The k-mutual-friend is stable with respect to the parameter k, i.e.* $g_{k+1} \subseteq g_k$.

For every edge e in subgraph g_{k+1}, $Tr(e) \geq k+1 > k$ suggests that this subgraph is also a g_k. Therefore, based on the stability property, if one wants to compute the exact $\mathcal{M}$ values for graph elements, we can make use of the offline result as input, with much less work than computing from scratch. Furthermore, the offline computations provide a useful upper bound for online computations.

Lemma 5.4.1 *Given* $G(V, E)$ *after offline computation, the edges from the online local subgraph* $g \subseteqq G$ *satisfy* $\{\mathcal{M}_g(e) \leq Tr_g(e) \leq Tr_G(e), e \in g\}$.

Proof 5.4.1 *Since g is a subset of G, for every edge $e \in g$, its local triangle count should be smaller or equal to the global triangle count, i.e. $Tr_g(e) \leq Tr_G(e)$. Based on the definition of k-mutual-friend subgraph, the local triangle count bounds the $\mathcal{M}_g$ value. All in all, we obtain the relationship $\mathcal{M}_g(e) \leq Tr_g(e) \leq Tr_G(e)$.*

We implement Algorithm 13 based on the above observations. The first step is to retrieve the local subgraph within the distance τ to v. Then, we iteratively compute the exact g_m from $m = \mathcal{M}_{min}$ to $m = \mathcal{M}_{max}$. Finally, we merge all the g_m to obtain the local subgraphs with exact $\mathcal{M}$ values. To illustrate, we retrieve a local subgraph by $traverse(a, 2)$ from the graph in Figure 1.1, and the result local graph is shown in Figure 5.5a. The number shows the triangle counts computed by the offline algorithm, which are the upper bound for the exact $\mathcal{M}$ values. Vertices $\{k, l, j\}$ and edges in dotted lines are immediately removed since their triangle counts are smaller than 2. In the first loop, we remove vertex g and edges $e(d, g), e(f, g)$ because their $\mathcal{M}$ values become one in the local graph. The rest of the graph is the 2-mutual-friend. In Figure 5.5b, we use the similar procedure to find 3-mutual-friend from the 2-mutual-friend, which includes vertices $\{a, b, c, d, f\}$ and edges connecting them. The algorithm terminates since the $\mathcal{M}_{max}$ is updated to the current largest triangle count equal to three.

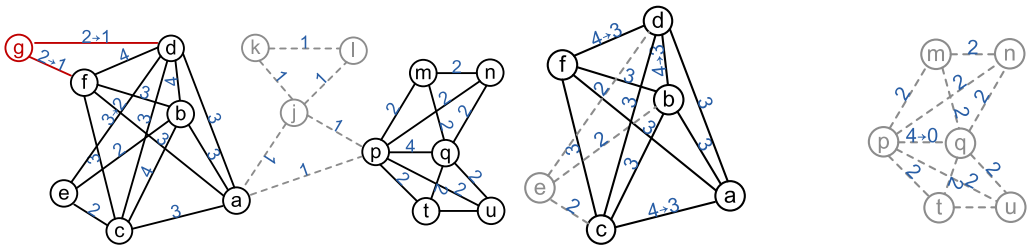
Algorithm 13: **Online Algorithm**

Input: $G(V, E)$, k, vertex v, and distance threshold τ
Output: Local subgraphs with exact $\mathcal{M}$ values

```
1 g ⟵ traverse(v, τ)
2 M_max ⟵ max{Tr_G(e) : e ∈ g}
3 M_min ⟵ k
4 for m from M_min to M_max do
5     compute m-mutual-friend and update g by Algorithm 9
6     g_m ⟵ {e : e ∈ g, Tr(e) = m}
7     M_max ⟵ max{Tr_g(e) : e ∈ g}
8 return g_{M_min} ⋃ ... ⋃ g_{M_max}
```

5.4.2 Visualizing *k*-mutual-friend Subgraph

Based on the online algorithm results, we next visualize the local subgraph reflecting the characteristics of the k-mutual-friend in social network. To begin with, we



(a) *traverse*$(a, 2)$ to 2-mutual-friend (b) 2-mutual-friend to 3-mutual-friend

Figure 5.5: Example of Online Computation

propose an orbital layout to decompose the network into hierarchy. Subsequently, we describe the implementation details of this layout in our visual system.

Orbital layout

As claimed in the introduction, the k-mutual-friend definition is proposed to capture the tie strength property in social network. Intuitively, vertices with larger $\mathcal{M}$ values are more important since they are closely connected with each other in the social network with many mutual friends. Therefore, a good layout for k-mutual-friend needs to emphasize elements with larger $\mathcal{M}$ values since they compose more cohesive subgraphs. With this observation we propose a layout with a set of concentric orbits. Vertices with larger $\mathcal{M}$ values are located close to the center, while vertices with smaller $\mathcal{M}$ values are placed on orbits further away from the center. Since the layout is analogous to the planetary orbits, it is called orbital layout as depicted in Figure 1.4b. The most connected part of the network is also the most central, such as the 5-clique (a, b, c, d, f) in the innermost orbit.

Furthermore, since we organize vertices with different $\mathcal{M}$ values into separated circles, the orbital layout forms a hierarchical structure. As such, users can filter out outer orbits and focus on the most central vertices, especially useful when the graph size is too large to clearly view. More importantly, the orbital layout is stable in the sense that the central part has the similar topological properties as the original graph. Figure 5.6 shows the cumulative degree distribution for the Epinions social network introduced in Table 5.2. Yet interestingly, the shape of the distributions is not affected by the parameter k. Note that the degree is normalized by the corresponding

average degree in each k-mutual-friend, since it tends to have higher average degree for larger k. The y-axis shows $P_>(d)$, i.e. the probability that the vertex degree in this k-mutual-friend subgraph is larger than d. Based on this nice property, the filtering operation on the hierarchy is reasonable without losing much structural information.

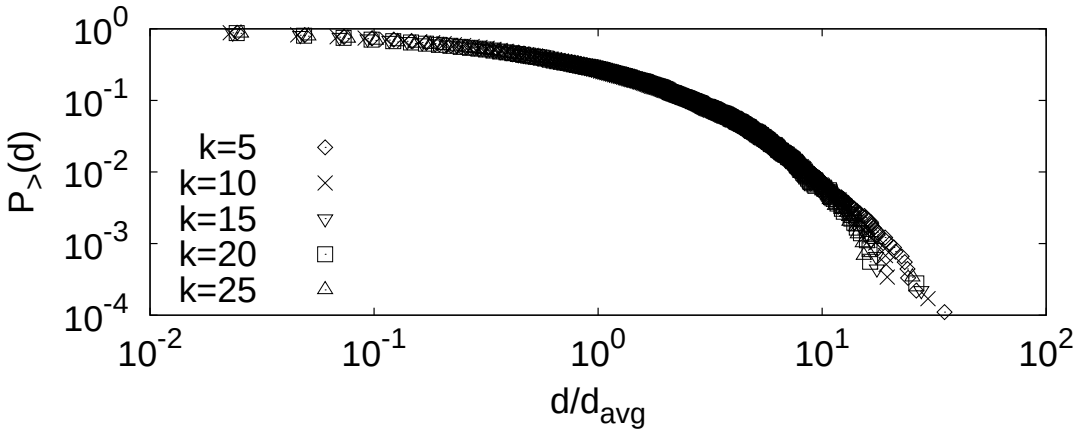


Figure 5.6: Stability Test on Epinions Social Network

Note that users can perceive more insights using orbital layout comparing with other popular layout algorithms, such as the radial layout [20] and the force directed layout [48]. Although radial layout is a hierarchical structure, it is sensitive to the focused vertex in the center and the layout may totally change with a different center. Force directed layout represents the topology well but is not a hierarchical structure to highlight social actors with many mutual friends. Also, it is not scalable due to $O(|V|^3)$ complexity. The qualitative comparison among these layouts is summarized in Table 5.1.

Table 5.1: Layout Comparison

	Hierarchy	Stability	Cost
Orbital layout	Yes	Yes	Median
Radial layout	Yes	No	Low
Force directed layout	No	Yes	High

Implementations

To improve the visual effect, we need to overcome the visual complexity of orbital layout, because it is a challenge to clearly present the cohesive subgraph with a large number of vertices. First, we set different colors to distinguish vertices in different orbits. Retrospect the motivating example in Figure 1.4b, it consists of four orbits in different colors representing vertices with four $\mathcal{M}$ values from 3 inside to 0 outside. In order to distinguish vertices within one orbit, the size of vertices is proportional to vertex degree to reflect the importance. For instance, vertex p has the largest degree so that it has the biggest size.

Next, we consider how to visualize edges to further reduce the visual complexity. Since vertices within one orbit may form several connected k-mutual-friend subgraphs, so we carefully order vertices such that vertices belongs to one subgraph are located successively on the orbit. As such, we can hide edges within one orbit without losing much connection information. As the Figure 1.4b shows, vertices g and h are near in the orbit and vertices j, k and l are near in the orbit. Furthermore, inspired by the radial layout, we put a vertex close to connected vertices in the inner orbit to minimize crossing edges. For example, vertices g and h are located in the top left since they are close to the inner neighbor vertex e.

5.4.3 Representative Tag Cloud Selection

Besides structure visualization, another dimension of social network analysis is to understand the interactions among social actors, which come from, for instance, the newfeeds from Facebook or tweets from Twitter. Since users may select a group of social actors with a great number of textual contents, we incorporate the tag cloud approach to summarizing various topics inside it. A potential challenge is how to select the most important tags to capture the major interests of these actors. Moreover, for distinct topics, the challenge might be how to discover a set of tags so that they could be comprehensive enough to cover different interests inside the same group.

To tackle these challenges, we compute a score for each tag by multiplying two factors, the significance and diversity. On the one hand the significance measure guarantees the truly popular tags can be selected, and on other hand the diversity measure

captures various rather than only similar topics. In our implementation, we adopt the TF-IDF approach for significance and the semantic distance in WordNet [21] for diversity. In representative tag selection, we first generate top N frequent words to form a candidate set, and filter out infrequent words to improve the efficiency. Then, we utilize a greedy strategy that iteratively moves tags with the largest score from the candidate set to the representative set until the number of selected tags reaches $n, n < N$, a user adjustable parameter. As such, we discover representative tags summarizing the interactions inside the local subgraph. Users can quickly select and browse preferred subgroup of actors to explore what activities they are involved in, or what topics they are taking about, etc.

5.5 Demonstration

To support online visual analysis, we implement a visual interactive system accessible on the Web[4], and provide a use case on Twitter dataset in Figure 5.7 to illustrate our idea.

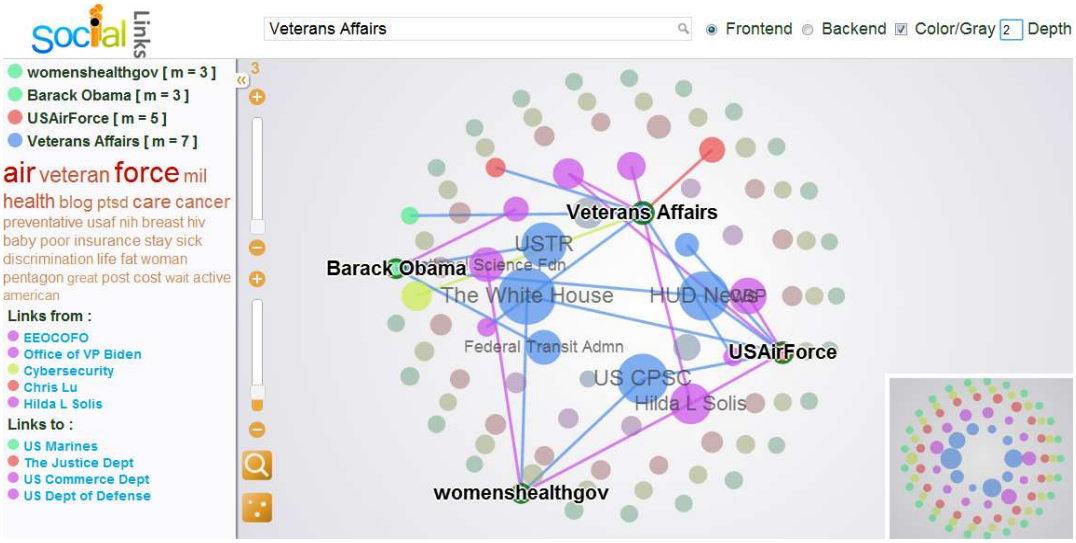


Figure 5.7: Visual Analysis Interface

Based on the real use case on Twitter social graph, we illustrate the functionalities and the advantages of our visual analytic browsing interface in Figure 5.7, which

[4]http://db128gb-b.ddns.comp.nus.edu.sg:8080/vis/demo

consists of three parts, i.e. search input area on the top, information summarization in the left column, and subgraph visualization in the main frame. After users input keywords in search box and select a focused vertex matching the keywords, our system visualizes the local subgraph in the main frame, so that users can select vertices they are interested in with the summarization in the left column. Without loss of generality, this example shows the 3-mutual-friend graph for the keyword "white house", in which vertices represent twitter actors and edges represent the "following" relationships. The depth, equivalent to the distance threshold, is set to 2.

With the help of online algorithm and layout generation, we dramatically reduce the visual complexity in the main frame. The visible subgraph only contains 89 vertices and 527 edges, which is much smaller the initial local subgraph with 2006 vertices and 2838 edges. As a result, we could quickly perceive that the networking of "The White House" is dominated by various US departments and government officials, which is unlikely to obtain from thousands of vertices with messy information. Furthermore, users can highlight several vertices and their neighbors while other vertices and edges become transparent. Considering in some cases subgraphs are quite large, users can use frontend search to locate preferred vertices within the current subgraph, or adjust the $\mathcal{M}$ value lower bound to filter out unsatisfied graph elements using the slide bar at the top left corner. Moreover, we support zoom in/out function to focus on part of the graph and users can view the sketch of the whole subgraph with a thumbnail at the bottom right corner.

The left column displays the $\mathcal{M}$ values of the highlighted vertices, the corresponding tag cloud as well as the link information for the vertex representing officials of "Veterans Affairs". The tag cloud is a helpful tool that summarizes the most significant and diverse topics in their tweets. In this example, we select 30 representative tags out of 100 candidates, where "Veterans Affairs" may show great concern about the PTSD (Post Traumatic Stress Disorder) and discrimination problems while "womenshealthgov" mainly focuses on topics like health, breast cancer and baby. In order to know the source of these tags, hovering over specific tag in the tag cloud will trigger the source vertices being highlighted. If we point to the "insurance" tag, the Twitter actor "Barack Obama" will be highlighted indicating that he pays close attention to the insurance issue.

5.6 Experiments

We present experimental studies to evaluate our social network visual analysis system in this section. For simplification, we refer to the intuitive algorithm in Section 5.3.1 as *mNaive*, Algorithm 9 as *mImproved*, while refer to Algorithm 10 as *dStream*, Algorithm 12 as *dPartition*. The *mOnline* is short for the online algorithm. We implement these algorithm in Java language and evaluate on the Windows operating system with Quad-Core AMD Opteron(tm) processor 8356 and 128GB RAM.

We compare our solutions on a great deal of real social network datasets described in Table 5.2, most of which are collected from the Stanford Network Analysis Project's website[5]. The datasets are sorted in increasing order of edge number. We utilize moderate size datasets (the first three) to compare in memory algorithms, while use large size datasets (the last three) to compare algorithms in graph database. Moreover, Twitter and DBLP datasets are selected for online visual analysis since they contain rich textual information.

Table 5.2: Dataset Statistics

Dataset	**Vertex**	**Edges**	**Description**
Epinions	$75k$	$405k$	Who-trusts-whom graph
Twitter	$452k$	$813k$	Who-follows-whom graph
DBLP	$916k$	$3,063k$	Who-cites-whom graph
Flickr	$1,715k$	$22,613k$	Flickr contact graph
FriendFeed	$653k$	$27,811k$	Friendship graph
Facebook	$72,661k$	$160,975k$	Friendship graph

5.6.1 Offline Computations Evaluation

Memory based Algorithms

We compare *mNaive* and *mImproved* algorithms on three datasets and results are summarized in Figure 5.8. This figure depicts the effect of k on the response time of three datasets. For Epinions and DBLP datasets, *mImproved* outperforms *mNaive* evidently, while their performances on Twitter dataset are in the same level. This is

[5] http://snap.stanford.edu/

because Twitter dataset having average degree less than 2 is much more sparse than the other two datasets. Therefore, even the naive algorithm can reach the stable state very fast without incurring a great deal of unnecessary triangle computations. For other two datasets, *mImproved* is about one order faster than *mNaive* averagely.

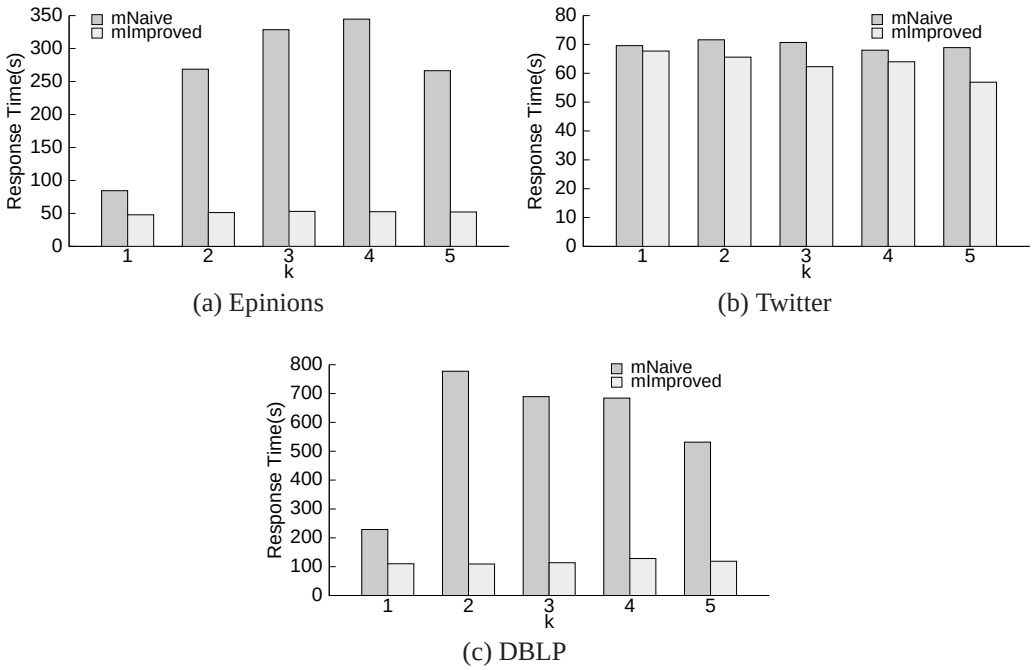


Figure 5.8: Comparison of Memory Algorithms

One interesting observation is that the response time is not quite related to k, but mainly determined by the triangle computing times in each algorithm, i.e. how many times the algorithm calls the triangle counting operator. As in the first two rows in Table 5.3, the triangle computing times for Epinions dataset in *mNaive* is about ten times of that in *mImproved*, which is close to the ratio of response time. Thus, the result again justifies our conclusion in Section 5.3.1 that *mImproved* outperforms *mNaive* mainly because it largely reduces the amount of triangle computations. More specifically, when $k = 1$, because we only remove edges not in any triangles without affecting other edges, *mNaive* can finish in two iterations (make sure that the graph is unchanged in the second iteration), and *mImproved* only needs one iteration. The response time for *mNaive* decreases when k equals to 5 since the number of triangle computations drops to $2,439k$, smaller than the number when k equals to 3 and 4. The triangle computing times for DBLP dataset in the last two rows in Table 5.3 have

the similar pattern. For Twitter dataset, both algorithms need the number of triangle computations in the same level, which determines that their response time also close to each other. To sum up, *mImproved* is much faster than *mNaive* mainly because it reduces the number of triangle computations, especially when the graph is dense.

Table 5.3: Triangle Computing Times

	1	**2**	**3**	**4**	**5**
mNaive	717k	2,219k	2,840k	3,088k	2,439k
mImproved	130k	202k	249k	284k	311k
mNaive	1,097k	1,261k	1,324k	1,364k	1,391k
mImproved	873k	867k	836k	819k	817k
mNaive	5,950k	24,767k	22,950k	25,166k	21,085k
mImproved	288k	1,028k	1,921k	2,671k	3,240k

Disk based Algorithms

Next we evaluate the disk based algorithms with three large scale datasets. For partition based algorithm, we control the usage of memory by only allowing to store a subgraph with at most $1GB$ size. As such, we can estimate the number of partitions p for each dataset according to the graph size in graph database as in Table 5.4. Since the response time is not determined by k, we set k as 3 to compare the performance of two disk based algorithms. The results in Figure 5.9 depicts the response time for the three datasets with two parts: I/O time and CPU time. All in all, the partition based algorithm is about five times faster than the streaming based algorithm, and the response times for both of them are increasing with respect to the increase of graph size. In particular, *dStream* algorithm is dominated by the I/O time, while *dPartition* is dominated by the CPU time, in accord with our analysis in Section 5.3.

In essence, the major difference between *dStream* and *dPartition* is the cost for triangle computations. As shown in Table 5.5, the average cost for triangle computations in *dPartition* is only one tenth of that in *dStream*, because most of the triangle computations in the former approach are in memory while all the triangle computations in the later one are in graph database. Comparing three datasets, the average triangle computing time for Facebook is the fastest for both algorithms due to the smallest average degree of Facebook. As a result, although the number of edges in Facebook is much larger than that in FriendFeed, the response time of Facebook is slightly

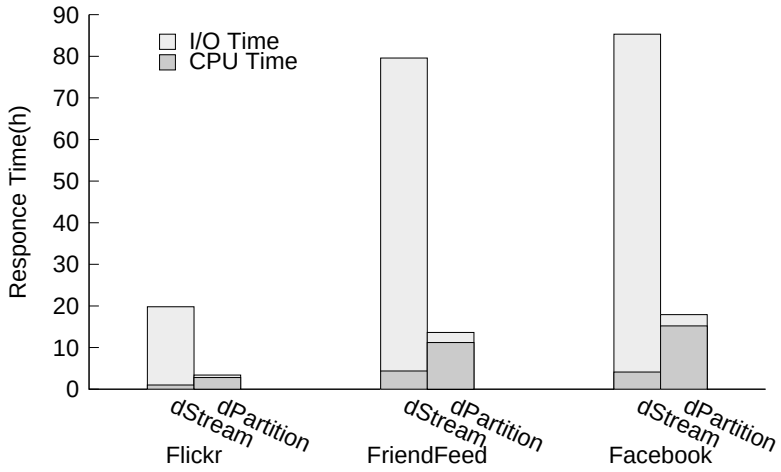


Figure 5.9: Comparison of Disk Algorithms

larger than that of FriendFeed. Moreover, Table 5.6 summarizes the percentages of the partitioning part and the computing part for *dPartition* algorithm. Because the partitioning algorithm reads the input graph only once and writes the partitions back to graph database, the partitioning part costs small amount of time comparing to the computing part.

Table 5.4: Number of Partitions in Algorithm 12

	Flickr	**FriendFeed**	**Facebook**
Size(GB)	1.57	1.92	11.6
p	2	2	12

In conclusion, *dPartition* trades off a lightweight graph partitioning for fast triangle computing in memory. The result verifies our claim in Section 5.3 that the partition based algorithm is I/O-efficient in practice.

Table 5.5: 10k Times Triangle Computing Cost

Dataset	*dStream*	*dPartition*
Flickr	122.1s	11.3s
FriendFeed	349.6s	33.5s
Facebook	12.9s	1.3s

Table 5.6: Percentages of Response Time

	Flickr	**FriendFeed**	**Facebook**
Partitioning part	9.1%	10.5%	13.2%
Computing part	90.9%	89.5%	86.8%

5.6.2 Online Analysis Evaluation

By randomly selecting 10 focused vertices on Twitter and DBLP datasets respectively, we obtain the average performance of online analysis with three components: *mOnline* algorithm, orbital layout generation and tag cloud selection. All the experiments are based on the 3-mutual-friend graph calculated by the offline solution. For tag cloud selection, we obtain 20 representative tags out of 100 candidates from the text in focused vertices. The major objective is to test whether our system can well support online analysis.

Table 5.7 shows the efficiency measures by varying the distance threshold τ from 1 to 3. It is clear that the total response time has an ascending trend with the increase of τ for both datasets. Taken separately, the costs of online algorithm and the layout generation are largely increasing with respect to τ. The major reason is that the response time for the first two components is proportional to the number of edges, which increases obviously with respect to τ, as in the bottom row of Table 5.7. However, the speed of tag cloud selection remains stable since it is only affected by the textual content in the focused vertex. Comparing the difference between two datasets, the tag cloud selection for Twitter is much slower because the number of words in tweets is large than that in paper title.

Table 5.7: Average Response Time(in ms)

	distance threshold τ					
	Twitter			DBLP		
Component	**1**	**2**	**3**	**1**	**2**	**3**
OnlineAlgo	1	32	563	2	16	498
Layout	2	6	138	2	5	108
TagCloud	1986	1726	1829	164	176	189
Avg edge num	2	368	9856	22	348	7727

Moreover, the average edge number suggests that distance threshold $\tau = 2$ is a practical setting for online analysis, generating local subgraph with reasonable size. Note

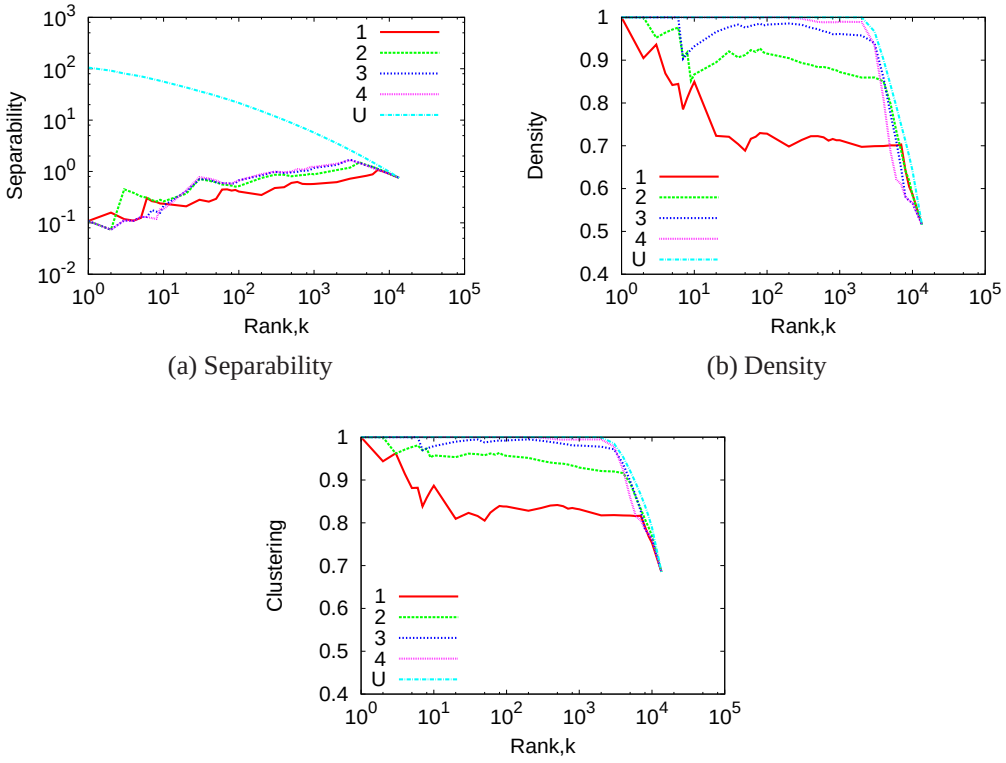
that we don't consider network transmission time since it is unstable and highly affected by the network condition, which is not the focus of this evaluation. In summary, the whole analytical procedure can be finished less than three second so that it is acceptable for online interactive applications.

5.6.3 Evaluation based on the ground-truth communities

According to the methodology proposed by Jaewon et al. [123], we further evaluate the effectiveness of the k-mutual-friend definition in identifying the ground-truth communities. In [123], the authors compared the performance of six representative community scoring functions with respect to a set of goodness metrics. In order to do experiments comparing with these scoring functions, first we need to define a scoring function based on the k-mutual-friend subgraph. Inspired by the triangle participation ratio, which is the fraction of nodes in community S that belong to a triad, we propose the mutual-friend participation ratio, since triangle is the special case of k-mutual-friend subgraph with k equals to one. Specifically, given the parameter k, the mutual-friend participation ratio is defined as $\frac{|\{v: v \in S \wedge v \in g_k \wedge g_k \subseteq S\}|}{n_S}$, in which n_S is the number of nodes in S. In particular, it is a generalized triangle participation ratio, which is exactly the same as triangle participation ratio when $k = 1$.

Next, we briefly review three goodness metrics defined in [123], i.e. separability, density and clustering coefficient. The goodness metrics $g(S)$ are defined for one community S. Separability measures the ratio between the internal and the external number of edges of S: $g(S) = \frac{m_S}{c_S}$, in which m_S is the number of edges in S and c_S is the number of edges on the boundary of S. Density builds on intuition that good communities are well connected. One way to capture this is to characterize the fraction of the edges (out of all possible edges) that appear between the nodes in S, $g(S) = \frac{m_S}{n_S(n_S-1)/2}$. Clustering coefficient is based on the premise that network communities are manifestations of locally inhomogeneous distributions of edges. It is the number of closed triplets (or 3 times number of triangles) over the total number of triplets (both open and closed). To sum up, the above goodness metrics quantifies different desirable properties of a community.

We test on the real-world networks with ground-truth communities downloaded from the SNAP website [107], including DBLP dataset, Amazon dataset and LiveJournal



(a) Separability

(b) Density

(c) Clustering Coefficient

Figure 5.10: Cumulative Average of Goodness Metrics

dataset. The experiments are formulated as follows. For each social network dataset, we have a set of ground-truth communities S. For each community scoring function $f(S)$, we rank the ground-truth communities by the decreasing score $f(S)$. We measure the cumulative running average value of the goodness metric $g(S)$ of the top-k ground-truth communities. If the scoring function ranks the communities in the decreasing order of the goodness metric, the cumulative running average value would decrease monotonically with k. In this way, we could know whether the scoring function can capture the characteristic of the goodness metric.

We found qualitatively similar results on all our datasets. Here we only present results for the DBLP dataset in Figure 5.10 to show our findings. We vary the setting of parameter k from 1 to 4 to compare the performance difference with respect to k. First of all, Figure 5.10a shows the results of separability for DBLP ground-truth communities ranked by four mutual-friend participation ratio with different k. Moreover, we use a curve "U" to present upper bound, i.e., the cumulative running

average of separability when ground-truth communities are ordered by decreasing separability. It can be easily observed that all of them can not well represent the separability. This is because the mutual-friend participation ratio prefer densely linked ground-truth communities, which tend to connect to many other vertices outside the ground-truth communities.

Similarly, Figures 5.10b,5.10c show the cumulative running average of density metrics and clustering coefficient respectively. We observe that all the mutual-friend participation ratios have the similar trend with respect to the upper bound curve, because all of them tend to rank denser and more clustered ground-truth communities higher. More specifically, with larger k value, the curve is more closer to the upper bound curve in general, since we tend to discover denser communities with larger k value. Based on the above analysis, we conclude that the k-mutual-friend subgraph definition is meaningful for identifying cohesive communities in real life networks.

5.7 Summary

In this chapter, we have introduced a novel framework that integrates the cohesive subgraphs discovery with the visual social network analysis. Unlike previous works, we proposed a new cohesive subgraph definition called k-mutual-friend to take the tie strength into consideration. Moreover, a memory based solution is proposed and extended to the scalable solution in the graph database. To further consolidate this interesting framework, we provided a visual analytic browsing interface that helps navigate users in searching and browsing the graph structure as well as semantics. The outcomes from an experimental study demonstrated that our solution is both efficient and effective. As for future research, we expect to extend our framework for other graph based analytic applications, such as protein-protein interaction analysis, RDF graph analysis etc. Another challenging direction is to maintain the cohesive subgraphs with frequently updates. As such, we shall provide a real time analytic toolkit to monitor everyone's evolving social network.

Chapter 6

Conclusions

In this thesis, we claim that making database applications accessible to ordinary users is as important as improving database capability. As such, we have conducted an intensive study to convert data into intelligence by means of data analytics and data visualization, in order to make database usable. Particularly, we identified new data analyzing problems and efficiently solved them in three key aspects, i.e. preference mining, keyword search in databases as well as social network analysis. Extensive experiments were conducted and the results validated the feasibility and the efficiency of these approaches. Furthermore, we provided prototype systems for users to test, and found that they were indeed helpful because users were able to interact with the visualized interfaces and drilled down to desired results by understanding the key information from the summarized result view intuitively.

Subsequently, the following states the major contributions of this thesis in interactive data analysis in three key aspects and then present the future directions for this thesis.

6.1 Results and Contributions

For eliciting users' preference, we addressed a user preference query on top of multi-dimensional datasets. We proposed to elicit the preferred ordering of a user by utilizing skyline objects as representatives of possible ordering. With the notion of

order-based representative skylines, representatives were selected by means of sampling based on the orderings that they represented. To further facilitate preference exploration, a hierarchical clustering algorithm was applied to compute a denogram on the skyline objects. By coupling the hierarchical clustering with visualization techniques, this framework allowed users to refine their preference weight settings by browsing the hierarchy. We conducted extensive experiments, and the results showed that our approach was both effective and efficient.

We next applied the hierarchical browsing approach in the application of keyword search in databases. To this end, we implemented a novel system allowing users to perform diverse, hierarchical browsing on keyword search results. It partitioned the answer trees in the keyword search results by selecting k diverse representatives from the answer trees, separating the answer trees into k groups based on their similarity to the representatives and then recursively applying the partitioning for each group. By constructing summarized results for the answer trees in each of the k groups, we provided a visual interface for users to quickly locate the results that they desired. Extensive experiments were conducted, and the results validated the feasibility and the efficiency of our system.

We finally introduced a novel subgraph concept to capture the cohesion in social interactions, and proposed an I/O efficient approach to discover cohesive subgraphs. In addition, we proposed an analytic system which allowed users to perform intuitive, visual browsing on a large scale social network. We hierarchically visualized the subgraph out on orbital layout, in which more important social actors are located in the center. By summarizing textual interactions between social actors as the tag cloud, we provided a way to quickly locate active social communities and their interactions in a unified view. The experiments conducted on various social network datasets validated the effectiveness and the efficiency of our system.

6.2 Future Directions

This thesis only covers three important aspects in the area of interactive data analysis in databases. As for future research, there are many research directions relating to the interactive data analysis in databases. We will discuss some of these directions as described below.

6.2.1 Unified Interactive Data Analytical Platform

Although we presented visualized systems implemented for every key topic we studied in, there is still room for improvement by developing a unified interactive data analytical platform, in order to support solutions for various interactive data analytical problems in database applications. The advantages of this platform are two fold. To begin with, it is more flexible for users since they can handle different types of data analysis transparent to the complex underlying storage. Furthermore, data analysis can be more productive by means of cross analyzing on top of multi-structured data, which means a variety of data formats and types. In this way, users probably obtain more insights about the data than single data analyses.

This unified platform will bring about many challenging research directions. First of all, we need a powerful database system or storage platform to treat both structured and unstructured data as first class citizens natively without the loss of efficiency. As for the visualized interface, the challenge is to support more complex analyses while keeping the intuitiveness and effectiveness. Both of the above directions are promising research topics and are the most important foundations for a unified interactive data analytical platform.

6.2.2 Big Data Analysis

According to research by MGI and McKinsey's Business Technology Office [87], the amount of data in real world applications has been exploding, and analyzing large data sets, so-called big data, will become a key basis of competition, underpinning new waves of productivity growth, innovation, and consumer surplus. Therefore, there exist big opportunities for database researchers to move towards big data analysis. To this end, we need to take advantage of parallel/distributed processing using modern hardware, such as cloud computing, GPU general purpose computing (GPGPU) as well as multi-core processing. There may exist two kinds of challenges. On one hand, data analytical problems usually need sophisticated algorithms to solve, so how to devise efficient parallel algorithm for these problems is challenging. On the other hand, even if some algorithms already have parallel/distributed solutions, it is still a challenge to apply these algorithms to making full use of these

modern hardwares. Future work must be done on these two directions in order to make big data analysis feasible for real life applications.

Bibliography

[1] J. Abello, F. Van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, pages 669–676, 2006. 17

[2] H.J. Ader, G.J. Mellenbergh, and D.J. Hand. *Advising on Research Methods: a consultant's companion*. Johannes van Kessel Publ., 2008. 1

[3] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan. Banks: Browsing and keyword searching in relational databases. In *VLDB*, page 1086, 2002. 6, 26, 67

[4] C.C. Aggarwal and P.S. Yu. Redefining clustering for high-dimensional applications. In *TKDE*, pages 210–225, 2002. 20

[5] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009. 8, 17, 82

[6] Nir Ailon. Aggregation of partial rankings, p-ratings and top-m lists. In *SODA*, pages 415–424, 2007. 24

[7] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 2008. 24, 25

[8] Richard D. Alba. A graph-theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, pages 113–126, 1973. 9, 28

[9] J.I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *Networks and Heterogeneous Media*, page 371, 2008. 30

[10] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD*, pages 49–60, 1999. 16

[11] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic. The role of social networks in information diffusion. In *WWW*, 2012. 29, 90

[12] W.-T Balke, Ulrich Gntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004. 19

[13] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *SoftVis*, pages 165–172, 2005. 17

[14] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006. 71, 72, 73

[15] Ken Black. *Business statistics: for contemporary decision making*. Wiley, 2011. 16

[16] Stephan Bloehdorn and Alessandro Moschitti. Structure and semantics for expressive text kernels. In *CIKM*, pages 861–864, 2007. 64

[17] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001. 5, 18, 47

[18] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *TVCG*, pages 1121–1128, 2009. 18

[19] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *TVCG*, pages 2301–2309, 2011. 18

[20] Ulrik Brandes and Christian Pich. More flexible radial layout. *J. Graph Algorithms Appl.*, pages 107–118, 2011. 109

[21] A. Budanitsky and G. Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources*, 2001. 111

[22] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005. 26

[23] Zhe Cao, Tao Qin, Tie Y. Liu, Ming F. Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136, 2007. 26

[24] Chee-Yong Chan, HV Jagadish, Kian-Lee Tan, Anthony KH Tung, and Zhenjie Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006. 20

[25] C.Y. Chan, HV Jagadish, K.L. Tan, A.K.H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006. 20

[26] Li Chen and Pearl Pu. Survey of preference elicitation methods. Technical report, EPFL, 2004. 22

[27] J. Cheng, Y. Ke, S. Chu, and M.T. Ozsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011. 28

[28] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003. 4

[29] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *ICDE*, pages 717–728, 2003. 5, 18, 19

[30] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *SIGKDD*, pages 672–680, 2011. 28

[31] Charles L.A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, pages 659–666, 2008. 8, 17, 82

[32] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10(1):243–270, 1998. 25

[33] C. Correa, T. Crnovrsanin, and K. Ma. Visual reasoning about social networks using centrality sensitivities. *TVCG*, pages 1–15, 2010. 30

[34] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, pages 1–19, 2009. 66

[35] Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang Nejdl. *DivQ*: diversification for keyword search over structured databases. In *SIGIR*, pages 331–338, 2010. 27, 59, 66

[36] W Edwards Deming. *Sample design in business research*, volume 23. Wiley-Interscience, 1990. 16

[37] M. Drosou and E. Pitoura. Comparing diversity heuristics. Technical report, Technical Report 2009-05. Computer Science Department, Univ. of Ioannina, 2009. 17, 60, 62, 75

[38] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622. ACM, 2001. 23

[39] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996. 16

[40] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *PODS*, pages 47–58, 2004. 24

[41] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM J. Discrete Math.*, pages 628–648, 2006. 24

[42] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003. 24

[43] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, pages 301–312, 2003. 24, 25

[44] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001. 38

[45] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost np-complete. In *FOCS*, pages 2–12, 1991. 9

[46] L.C. Freeman. Visualizing social networks. *Journal of social structure*, 2000. 29

[47] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, pages 933–969, 2003. 25

[48] T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, pages 1129–1164, 1991. 109

[49] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *CHI*, pages 211–220, 2009. 29

[50] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, pages 7821–7826, 2002. 29

[51] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, pages 927–940, 2008. 27, 59

[52] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009. 8, 17, 59, 60, 62, 75

[53] Leo A Goodman. Snowball sampling. *The Annals of Mathematical Statistics*, pages 148–170, 1961. 16

[54] Przemyslaw A. Grabowicz, Jose J. Ramasco, Esteban Moro, Josep M. Pujol, and Vłctor M. Egułluz. Social features of online networks: the strength of weak ties in online social media. *CoRR*, 2011. 29

[55] M.S. Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973. 9, 29

[56] Antomn Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984. 19

[57] Torben Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990. 41

[58] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, pages 10–18, 2009. 16

[59] Robert A Hanneman and Mark Riddle. Concepts and measures for basic network analysis. *The SAGE Handbook of Social Network Analysis*, pages 340–369, 2011. 17

[60] David Haussler. Convolution kernels on discrete structures. Technical report, Univ. of California, Santa Cruz, 1999. 63, 65

[61] H. He, H. Wang, J. Yang, and P.S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, page 316, 2007. 6, 7, 62

[62] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *CSC*, pages 336–341, 2011. 90

[63] J. Heer and M. Bostock. Declarative language design for interactive visualization. *TVCG*, 16(6):1149–1156, 2010. 18

[64] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, page 681, 2002. 6, 26, 67

[65] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003. 26

[66] Alfred Inselberg. *Parallel coordinates: visual multidimensional geometry and its applications*. Springer, 2009. 44

[67] HV Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, pages 13–24, 2007. 1, 10

[68] Bin Jiang, Jian Pei, Xuemin Lin, David W. Cheung, and Jiawei Han. Mining preferences from superior and inferior examples. In *KDD*, pages 390–398, 2008. 4, 21

[69] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, page 505, 2005. 6, 26, 27, 62, 67, 80

[70] Werner Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002. 3, 4, 21

[71] D. Knoke, S. Yang, and J.H. Kuklinski. *Social network analysis*. Sage Publications Los Angeles, CA, 2008. 8

[72] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002. 5, 18, 19

[73] Martin Krzywinski, Jacqueline Schein, İnanç Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, pages 1639–1645, 2009. 76

[74] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. Approaching the skyline in z order. In *VLDB*, pages 279–290, 2007. 5, 18

[75] J. Leskovec, K.J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010. 29

[76] Michael S Lewis-Beck. *Data analysis: An introduction*. Sage, 1995. 1

[77] C. Li, B.C. Ooi, A.K.H. Tung, and S. Wang. Dada: a data cube for dominant relationship analysis. In *SIGMOD*, pages 659–670, 2006. 19

[78] C. Li, A.K.H. Tung, W. Jin, and M. Ester. On dominating your neighborhood profitably. In *VLDB*, pages 818–829, 2007. 19

[79] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008. 27, 61

[80] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007. 5, 20

[81] Fang Liu, Clement T. Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006. 26, 27

[82] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. In *VLDB*, pages 313–324, 2009. 27

[83] R.D. Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, pages 169–190, 1950. 9, 28

[84] R.D. Luce and A.D. Perry. A method of matrix analysis of group structure. *Psychometrika*, pages 95–116, 1949. 28

[85] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, page 14, 1967. 16

[86] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001. 17

[87] MGI and McKinsey's Business Technology Office. Big data: The next frontier for innovation, competition, and productivity. http://www.mckinsey.com/insights/business_technology. 123

[88] E. Minack, G. Demartini, and W. Nejdl. Current Approaches to Search Result Diversification. In *Proc. of 1st Intl. Workshop on Living Web*, 2009. 11, 17

[89] Denis Mindolin and Jan Chomicki. Discovering relative importance of skyline attributes. *PVLDB*, pages 610–621, 2009. 4, 22

[90] NBA. Basketball database. http://www.databasebasketball.com. 46

[91] M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, pages 413–421, 2004. 29

[92] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994. 42, 43, 75

[93] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, pages 245–251, 2010. 17

[94] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003. 5, 18, 19, 20

[95] C.A.R. Pinheiro. *Social network analysis in telecommunications*, volume 37. Wiley, 2010. 8

[96] Lu Qin, Jeffrey Xu Yu, Lijun chang, and Yufei Tao. Querying communities in relational databases. In *ICDE*, 2009. 27

[97] Marko A Rodriguez and Peter Neubauer. The graph traversal pattern. *arXiv preprint arXiv:1004.1001*, 2010. 99

[98] Parke Godfrey Ryan, Ryan Shipley, and Jarek Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005. 5, 18, 19

[99] T.L. Saaty. *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, 1980. 23

[100] Romesh Saigal. *Linear Programming: A Modern Integrated Analysis*. Springer, 1995. 36, 37

[101] Nikos Sarkas, Gautam Das, Nick Koudas, and Anthony K. H. Tung. Categorical skylines for streaming data. In *SIGMOD*, pages 239–250, 2008. 19

[102] S.B. Seidman. Network structure and minimum degree. *Social networks*, pages 269–287, 1983. 9, 28, 92

[103] S.B. Seidman and B.L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, pages 139–154, 1978. 9, 28

[104] Mehdi Sharifzadeh and Cyrus Shahabi. The spatial skyline queries. In *VLDB*, pages 751–762, 2006. 19

[105] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge Univ. Press, 2004. 63, 66

[106] G. Smith, M. Czerwinski, B.R. Meyers, G. Robertson, and DS Tan. FacetMap: A scalable search and browse visualization. *IEEE Transactions on Visualization and Computer Graphics*, pages 797–804, 2006. 17

[107] SNAP. Stanford network analysis project. http://snap.stanford.edu. 118

[108] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *WWW*, 2012. 101

[109] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. PerK: personalized keyword search in relational databases through preferences. In *EDBT*, pages 585–596, 2010. 27, 59, 62, 66

[110] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, 2007. 81

[111] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*, pages 990–998, 2008. 30

[112] Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009. 5, 20, 48

[113] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, pages 228–236, 2008. 27

[114] S. V. N. Vishwanathan and Alex Smola. Fast kernels on strings and trees. In *NIPS*, 2002. 63

[115] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5(9):812–823, 2012. 9, 28, 95

[116] N. Wang, S. Parthasarathy, K.L. Tan, and A.K.H. Tung. Csv: visualizing and mining cohesive subgraphs. In *SIGMOD*, pages 445–458, 2008. 30, 90, 91

[117] N. Wang, J. Zhang, K.L. Tan, and A.K.H. Tung. On triangulation-based dense neighborhood graph discovery. In *VLDB*, pages 58–68, 2010. 9, 28, 90, 94

[118] S. Wang, Q.H. Vu, B.C. Ooi, A.K.H. Tung, and L. Xu. Skyframe: a framework for skyline query processing in peer-to-peer systems. *The VLDB Journal*, pages 345–362, 2009. 19

[119] Shiyuan Wang, Beng C. Ooi, and Anthony K. H. Tung. Efficient skyline query processing on peer-to-peer networks. In *ICDE*, pages 1126–1135, 2007. 19

[120] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*. Cambridge university press, 1994. 17

[121] D.R. White and F. Harary. The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Sociological Methodology*, pages 305–359, 2001. 9

[122] Ping Wu, Caijie Zhang, Ying Feng, Ben Y. Zhao, Divyakant Agrawal, and Amr El Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*, pages 112–130, 2006. 19

[123] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, page 3. ACM, 2012. 118

[124] Daniel Yates, David S Moore, and GP McCabe. *The practice of statistics*. WH Freeman and Company, New York, 1998. 16

[125] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, pages 228–236, 2009. 27, 62

[126] J.X. Yu, M.T. Özsu, L. Chang, and L. Qin. *Keyword Search in Databases*, volume 1. Morgan & Claypool Publishers, 2010. 5

[127] C.X. Zhai, W.W. Cohen, and J. Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *SIGIR*, pages 10–17, 2003. 82, 83

[128] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, 2011. 30, 90

[129] Z. Zhang, R. Cheng, D. Papadias, and A.K.H. Tung. Minimizing the communication cost for continuous skyline maintenance. In *SIGMOD*, pages 495–508, 2009. 19

[130] Z. Zhang, X. Guo, H. Lu, A.K.H. Tung, and N. Wang. Discovering strong skyline points in high dimensional spaces. In *CIKM*, pages 247–248, 2005. 20

[131] Z. Zhang, L.V.S. Lakshmanan, and A.K.H. Tung. On domination game analysis for microeconomic data mining. *TKDD*, pages 1–27, 2009. 19

[132] F. Zhao, G. Das, K.L. Tan, and A.K.H. Tung. Call to order: a hierarchical browsing approach to eliciting users' preference. In *SIGMOD*, pages 27–38, 2010. 13

[133] Feng Zhao and Anthony K.H. Tung. Large Scale Cohesive Subgraphs Discovery for Social Network Visual Analysis. In *VLDB*, 2013. 13

[134] Feng Zhao, Xiaolong Zhang, Anthony K.H. Tung, and Gang Chen. BROAD: Diversified Keyword Search in Databases. In *VLDB*, 2011. 13, 76

[135] Feng Zhao, Xiaolong Zhang, Anthony K.H. Tung, and Gang Chen. BROAD: Diversified Keyword Search in Databases. Technical report, TRD3/11, School of Computing, National Univ. Singapore, 2011. 88