KEYWORD SEARCH IN GRAPHS, RELATIONAL DATABASES AND SOCIAL NETWORKS

by **Mehdi Kargar**

a dissertation submitted to the Faculty of Graduate Studies of York University in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

KEYWORD SEARCH IN GRAPHS, RELATIONAL DATABASES AND SOCIAL NETWORKS

by **Mehdi Kargar**

Abstract

Keyword search, a well known mechanism for retrieving relevant information from a set of documents, has recently been studied for extracting information from structured data (e.g., relational databases and XML documents). It offers an alternative way to query languages (e.g., SQL) to explore databases, which is effective for lay users who may not be familiar with the database schema or the query language. This dissertation addresses some issues in keyword search in structured data. Namely, novel solutions to existing problems in keyword search in graphs or relational databases are proposed. In addition, a problem related to graph keyword search, team formation in social networks, is studied. The dissertation consists of four parts.

The first part addresses keyword search over a graph which finds a substructure of the graph containing all or some of the query keywords. Current methods for keyword search over graphs may produce answers in which some content nodes (i.e., nodes that contain input keywords) are not very close to each other. In addition, current methods explore both content and non-content nodes while searching for the result and are thus both time and memory consuming for large graphs. To address the above problems, we propose algorithms for finding r-cliques in graphs. An r-clique is a group of content nodes that cover all the input keywords and the distance between each pair of nodes is less than or equal to r. Two approximation algorithms that produce r-cliques with a bounded approximation ratio in polynomial delay are proposed.

In the second part, the problem of duplication-free and minimal keyword search in graphs is studied. Current methods for keyword search in graphs may produce duplicate answers that contain the same set of content nodes. In addition, an answer found by these methods may not be minimal in the sense that some of the nodes in the answer may contain query keywords that are all covered by other nodes in the answer. Removing these nodes does not change the coverage of the answer but can make the answer more compact. We define the problem of finding duplication-free and minimal answers, and propose algorithms for finding such answers efficiently.

Meaningful keyword search in relational databases is the subject of the third part of this dissertation. Keyword search over relational databases returns a join tree spanning tuples containing the query keywords. As many answers of varying quality can be found, and the user is often only interested in seeing the top-k answers, how to gauge the relevance of answers to rank them is of paramount importance. This becomes more pertinent for databases with large and complex schemas. We focus on the relevance of join trees as the fundamental means to rank the answers. We devise means to measure relevance of relations and foreign keys in the schema over the information content of the database.

The problem of keyword search over graph data is similar to the problem of team formation in social networks. In this setting, keywords represent skills and the nodes in a graph represent the experts that possess skills. Given an expert network, in which a node represents an expert that has a cost for using the expert service and an edge represents the communication cost between the two corresponding experts, we tackle the problem of finding a team of experts that covers a set of required skills and also minimizes the communication cost as well as the personnel cost of the team. We propose two types of approximation algorithms to solve this bi-criteria problem in the fourth part of this dissertation.

Acknowledgements

First and foremost I want to thank my supervisor, Professor Aijun An, for her guidance and all the useful discussions and brainstorming sessions. Her deep insights were extremely helpful at several stages of my research. I am also deeply grateful for her understanding and support during the times when I was down due to personal problems.

I would like to thank my committee members, Professor Parke Godfrey and Professor Xiaohui Yu. They have been helpful in providing advice many times during my research. I am grateful to Professor Jimmy Hunag, for sharing his knowledge in my qualifying exam. I would also like to take this opportunity to thank Professor Nick Cercone for his great support during my time at York University.

The members of both data mining and database labs have contributed immensely to my personal and professional time at York University. The group has been a source of friendships as well as great collaborators. I am also thankful to the administrative and technical staff of our department for their finest support.

Lastly, I would like to thank my family for all their encouragement and love. Words cannot express the feelings I have for my parents for their constant unconditional support in every aspect of my life.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

Exploring and querying structured data (e.g., relational databases and XML documents) using enterprise systems (e.g., DBMSs) needs knowledge of a query language (e.g., SQL) and the structure of the data (e.g., schema). This is sufficient for users who are familiar with both of the query language and the structure of the data. However, a lay user (i.e., anyone without the knowledge of the query language or the schema) may get lost. Keyword search – a well known method for extracting relevant knowledge from a set of documents in information retrieval – offers an alternative for lay users to explore structured data. Keyword search on structured data is different from finding relevant documents that contain query keywords. The former focuses on finding the interconnected structures, while the latter focuses on the content of the documents.

Structured data are usually modeled as graphs. For example, considering IDREF/ID as links, XML documents can be modeled as graphs. Relational databases can also be modeled using graphs, in which tuples are nodes of the graph and foreign key relationships are edges that connect two nodes (tuples) to each other [24, 51]. Users need a simple system that receives some keywords as input and returns a set of interconnected nodes/tupels that together cover all or part of the input keywords.

There are two approaches to keyword search over relational databases.

1. The entire database is materialized as a graph in which tuples are nodes of the graph and foreign key relationships are edges that connect two nodes (tuples) to each other. Then, the search is performed directly on the graph. In this case, proper indexing techniques are required to improve the efficiency.
2. The search engine uses the RDBMS to perform the search through executing a series of SQL statements. In this case, the search engine is benefitted by using query optimization techniques built in the RDBMS.

The former approach is called schema-free since it does not use the database schema for finding the answers, while the latter approach is called schema-based since it uses the RDBMS to issue SQL queries based on the database schema. In this dissertation, we propose solutions to existing problems in both approaches.

The problem of keyword search over graph data is similar to the problem of team formation in social networks. The social network is modeled as a graph whose nodes represent experts, each with one or more skills, and whose edge between two nodes is weighted by the communication cost between the two corresponding experts. In this setting, keywords are required skills and the nodes are the experts that possess skills. In addition, each expert may specify his/her consulting rate. Given a project whose completion requires a set of skills, we tackle the problem of finding from a social network a team of individuals that not only cover all the required skills but can also communicate and collaborate effectively. In addition, the personnel cost of the project should be minimized as well. In this dissertation, we propose a suite of algorithms for solving this bi-criteria problem.

1.2 Contributions

In this thesis, we propose new solutions to keyword search over structured data and team formation in social networks. Several topics in database, data mining and theoretical computer science have been investigated. The contributions of this dissertation are summarized as follows:

- **Finding Compact Answers:** Keyword search over a graph finds a substructure of the graph containing all or some of the input keywords. Current methods for keyword search on graphs may produce answers in which some content nodes (i.e., nodes that contain input keywords) are not very close to each other. To address the above problem, we propose the problem of finding r-cliques in graphs. An r-clique is a group of content nodes that cover all the input keywords and the distance between each two nodes is less than or equal to r. An exact branch and bound algorithm that produces all r-cliques is proposed. Two approximation algorithms that produce r-cliques with a provable performance bound in polynomial delay are proposed. Extensive experiments show that the proposed algorithms produce more compact answers comparing to previous methods. In addition, the running time of the algorithms is also faster than other methods in this area.

- **Finding Duplication Free and Minimal Answers:** Current graph keyword search methods may produce duplicate or very similar answers that contain the same set of content nodes. In addition, some of the nodes in an answer may contain query keywords that are all covered by other nodes in the answer. Removing these nodes does not change the coverage of the answer but can make the answer more compact. Such *minimal* answers are desirable in some applications. We define the problem of finding duplication-free and minimal answers, and propose a series of algorithms for finding such answers efficiently. Extensive performance studies using two large real data sets confirm the efficiency and effectiveness of the proposed methods.
- **Finding Meaningful Answers:** Keyword search over relational databases returns a join tree spanning tuples containing the query keywords. As many answers of varying quality can be found, and the user is often only interested in seeing the top-k answers, how to gauge the relevance of answers to rank them is of paramount importance. This becomes more pertinent for databases with large and complex schemas. We focus on the relevance of join trees as the fundamental means to rank

them. We devise means to measure relevance of relations and foreign keys in the schema over the information content of the database. We further show that finding minimal trees does not necessarily lead to more meaningful answers. It might seem to be in contrast to our previous contribution about finding minimal answers over graph data. However, minimal answers in the context of graph keyword search is desirable in some applications but not in relational databases. We discuss this issue in detail in Chapter 4. We test performance of our measures against existing techniques to demonstrate a marked improvement, and perform a user study to establish naturalness of the ranking.

- **Finding Affordable and Collaborative Teams of Experts:** As mentioned before, the problem of keyword search over graph data is similar to the problem of team formation in social networks. However, team formation often requires optimizing more than one objective. We tackle the problem of finding a team of experts that covers a set of required skills and also minimizes the communication cost as well as the personnel cost of the team. Since two costs need to be minimized, this is a bi-criteria optimization problem. We propose two types of approximation algorithms to solve the problem. The first type receives a budget on one objective and minimizes the other objective under the budget. In the second approach, an approximation algorithm is proposed to find a set of Pareto-optimal teams, in which each team is not dominated by other feasible teams in terms of the personnel and communication costs. Extensive experiments on real datasets demonstrate the effectiveness and scalability of the proposed algorithms.

1.3 Organization of the Dissertation

This dissertation is organized as follows.

Related work is presented in Chapter 2 in which existing work in keyword search in graphs, keyword search in relational database and team formation in social networks is reviewed.

In Chapter 3, the formal definition of an r-clique as an answer to the keyword search over graph data is presented. The motivations and benefits of finding r-cliques are thoroughly discussed. An exact algorithm is proposed that finds all r-cliques in the input graph. An approximation algorithm that produces r-cliques with 2-approximation in polynomial time is proposed. We further decrease the run time of the approximation algorithm with the cost of increasing the approximation ratio. Extensive performance studies using three large real data sets confirm the efficiency and accuracy of finding r-cliques in graphs.

In Chapter 4, the problem of finding duplication free and minimal answers is studied. We motivate the problem using examples and propose a series of algorithms for finding duplication free and minimal answers efficiently. For finding duplication free answers, a general framework for generating top-k duplication-free answers by wisely dividing the search space is introduced. For finding minimal answers, two general frameworks for confining or dividing the search space to ensure minimality and no duplication in the top-k answers are proposed. One of the algorithms is faster (completely polynomial) but may miss some answers, while the other one is complete (i.e., it allows all the possible answers to be considered), but is a fixed-parameter tractable (FPT) algorithm. Extensive performance studies using two large real data sets confirm the efficiency and effectiveness of the proposed methods.

In Chapter 5, we propose to improve relevance scoring of answers for keyword search over relational databases using the schema-based approach. The proposed ranking method is specifically more useful for larger and more complex database schema. We propose a series of measures and algorithms to compute the relevance scores and we use different approaches to capture the intended semantics of queries based on the importance of the connections involved in the answers. Extensive experiments and a user study on a large and complex schema show that the proposed methods are able to capture well the intended semantics behind queries.

In Chapter 6, we study the problem of finding an affordable and collaborative team from an expert network that minimizes two objectives: the communication cost among team members and the personnel cost of the team. Two functions are used to measure the communication cost of a team and another function is proposed to evaluate the personnel cost of the team. A suite of algorithms classified into two approaches are proposed to solve this bicriteria problem. In the first approach, a budget is given on one objective and the purpose is to minimize the other objective under the budget. The budget could be either on the communication cost or the personnel cost. This type of algorithm is called an (α, β) approximation algorithm. In the second approach, a set of approximate Pareto-optimal solutions are generated in which there exists no other team that dominates the solution in both of the costs. All of the proposed algorithms have provable approximation bounds. We evaluate the proposed algorithms on two real datasets and show that our proposed algorithms are effective and efficient.

In Chapter 7, we review the contributions of the dissertation and summarize the directions for future work.

2 Related Work

Related work is presented in this chapter. We review recent work in keyword search in graphs, keyword search in relational database and team formation in social networks. We further investigate the relation between two associated problems: keyword search in graphs and team formation in social networks.

2.1 Keyword Search in Graphs

Current approaches to keyword search over graph data can be categorized based on the type of answers they produce into tree-based methods and graph-based methods[1]. Tree-based methods can be further divided into: Steiner trees and distinct root trees.

2.1.1 Steiner Tree Methods

Steiner tree based methods produce a subtree of input graph G whose leaves are content nodes that together cover all the input keywords. A Steiner tree is evaluated based on the

[1]A complete survey on keyword search in databases and graphs can be found in [70].

total weight of the edges in the tree. Trees with smaller weights are considered as better answers. Finding a tree with the smallest weight is the well known Steiner tree problem, which is NP-complete [70].

In [8], a backward search algorithm for producing Steiner trees from a directed/undirected graph is presented. For each content node, the algorithm concurrently runs the Dijkstra's single source shortest path algorithm as an iterator, using the content node as the source node. All these iterators traverse the input graph in the reverse direction of edges. When an iterator for a content node meets a node in the graph, it finds a path from that node to the content node. Assume that the backward search algorithm finds a common node that has a path to a set of content nodes that cover all of the query keywords. Then, the common node and this set of content nodes plus the nodes on the shortest path between the common node and each node in the set of content nodes form a tree with the common node being the root. A dynamic programming approach for finding Steiner trees in graphs are presented in [16]. Although the dynamic programming approach has exponential run time complexity in terms of the number of input keywords, it is feasible for input queries with small number of keywords.

2.1.2 Distinct Root Trees

Due to the NP-completeness of the Steiner tree problem, algorithms for producing trees with distinct roots were introduced in recent years. For each node in the graph, such algorithms generate a tree rooted at this node, which covers all the input keywords. Thus, the generated trees have distinct roots. Distinct root trees are evaluated based on the sum of the shortest distance from the root to each content node. The number of distinct roots is n, where n is the number of nodes in graph. Thus, such algorithms generate at most n answers. This is much smaller than the number of possible trees which is $O(2^m)$ where m is the number of edges of the graph. It should be noted that the number of edges of the graph could be as large as $O(n^2)$. Thus, distinct root tree algorithms are much faster than Steiner tree based algorithms. However, distinct root tree methods may miss some answers when top-k or all answers need to be produced because only one tree rooted at a node is considered in the algorithms. If another tree rooted at the same node also has a better weight than the trees rooted at other nodes, this tree is not considered.

Similar to the backward search algorithm, a bidirectional search algorithm for finding trees with distinct roots is proposed in [32]. Bidirectional search improves the backward search algorithm by searching the potential roots in a forward manner. BLINKS improves the bidirectional search algorithm by using an efficient indexing structure [24]. The naive index computes and stores all the distances from the nodes to the keywords and vice versa. Since this naive index might be very large, BLINKS creates a bi-level index by first partitioning the input graph and then building intra and inter block indexes. In addition, the authors use two node-based partitioning methods, i.e., BFS-Based Partitioning and METIS-Based Partitioning [42]. To find distinct root trees in graphs that cannot fit into main memory, the work in [14] creates a smaller super node graph on top of the main graph which can reside in main memory.

2.1.3 Graph Based Methods

There are two methods that find subgraphs rather than trees for keyword search over graphs [51, 60]. The first method finds r-radius Steiner graphs that contain all of the input keywords [51]. Since the algorithm for finding r-radius graphs index them regardless of the input keywords, if some of the highly ranked r-radius Steiner graphs are included in other larger graphs, this approach may miss them. In addition, it may produce duplicate and redundant results [60]. The second method finds multi-centered subgraphs, called communities [60]. In each community, there are some center nodes. There exists at least one path between each center node and each content node such that the distance between them is less than R_{max}. Parameter R_{max} is used to control the size of the community. The authors of [60] propose an algorithm that produces all communities in an arbitrary order and another algorithm that produces ranked communities in polynomial delay. The rank of a community is based on the minimum value among the total edge weights from one of the centers to all of the content nodes. Finding communities as the answer for keyword search over graph data has three problems. While some of the content nodes may be close to each other, the others may not. In addition, for finding each community, the algorithm considers all of the nodes within R_{max} distance from every content node as a candidate for a center node. This leads to poor run-time performance. Finally, while including center and intermediate nodes in the answers can reveal the relationships between the content nodes, these center and intermediate nodes may be irrelevant to the query, which makes some answers hard to interpret. Our proposed model improves the community method by (1) finding r-cliques in which all the content nodes are close to each other, (2) improving the run-time by exploring only the content nodes during search, and (3) reducing the irrelevant nodes by producing a Steiner tree (instead of a graph) to reveal the relationship between the content nodes in an r-clique.

2.1.4 Other Related Topics

Finding r-cliques is closely related to *Multiple Choice Cover* problem, which is introduced in [4] and is also used for finding a team of experts in social networks in [3, 15, 48]. The same approach is also used in [46]. These approaches find a single best answer that has the smallest diameter. In comparison, we find all or top-k r-cliques with polynomial delay. Our problem is apparently more challenging. In addition, we use the sum of the weights between each pair of nodes as the ranking function, which we think is more suitable for keyword search. Previous works use other ranking functions such as the diameter of the graph to evaluate the answers.

Keyword search in graphs is related to the graph pattern matching problem. The concept of bounded graph simulation for finding maximum matches in graphs was recently introduced in [18]. The authors proposed algorithms for finding the maximum match in a graph based on the new definition of matches. The r-clique defined in this chapter can be considered as an input pattern in [18]. Also, the output of our algorithm is different from the one in [18]. Their algorithm finds one maximum match in a graph which contains all the nodes in the graph that match with a node in the query. Our top-k r-clique algorithm finds matches that cover all the input keywords but minimize the sum of distances between each two nodes.

Recently, the BROAD system is proposed to find diversified answers for keyword search on graphs [72]. The system is built on top of a keyword search engine and partitions the answer trees produced by the engine into dissimilar clusters. The dissimilarity between answers is measured based on the structural and semantic information of the given trees. The structural dissimilarity is measured based on the sub-tree kernel introduced in [65]. The semantic information is added to the kernel by merging the textual content of the nodes using the well known TF-IDF weighting scheme. A hierarchical browsing method is further proposed to help users navigate and browse the results. Our effort of finding duplication-free answers can be considered as a special case of finding diversified answers, where each answer must have a different set of content nodes. Our method has cheaper computational cost due to its problem simplicity. We find such "diversified" answers during the search process, while BROAD does it as a post-processing process. BROAD can be applied to the results of our method to further diversify the answers using the BROAD's dissimilarity measures.

2.1.5 Keyword Search vs. Team Formation

The problem of keyword search over graph data is similar to the problem of team formation in social networks. The social network is modeled as a graph whose nodes represent experts, each with one or more skills, and whose edge between two nodes is weighted by the communication cost between the two corresponding experts. In this setting, keywords are required skills and the nodes are the experts that possess skills. Given a project whose completion requires a set of skills (i.e., keywords), we tackle the problem of finding from a social network a team of individuals (i.e., answer or sub-graph) that not only cover all the required skills but can also communicate and collaborate effectively. Therefore, minimizing the communication cost among the experts in a team is equivalent to finding an answer to graph keyword search problem in which the weight of the answer (i.e., sub-graph) is minimized. In [34], we studied the single objective team formation problem that minimizes the communication cost among the experts. However, each expert may specify his/her consulting rate. Bi-objective team formation problem studies the possibility of minimizing the personnel cost of the project as well as the communication cost. Since two objectives are minimized, this problem is more challenging than the single objective graph keyword search problem.

2.2 Keyword Search in Relational Databases

Existing approaches to keyword search over relational databases fall into two classes. The first class of approaches convert the database into a graph, on which the search is then performed [8, 35, 60]. In this case, the foreign key connections correspond to the edges of the graph. One of the challenges of this approach is fitting the graph into the main memory [35]. Although some indexing techniques are proposed in the literature [14], converting a relational database with millions of records into a graph is memory consuming. In addition, how to find meaningful sub-graphs as query answers is still an open direction for future research [35]. A thorough overview of existing works on graph keyword search was presented previously in this section.

The second class of approaches considers the relational schema as a graph. However, the search is directly performed on relational databases through generating and executing SQL queries on the RDBMS [1, 27, 28, 53]. Therefore, it heavily relies on the database schema and query processing techniques in RDBMS. The methods for ranking the query answers are divided into two categories. In the first category, the final answers are simply ranked based on the number of joins [1, 27]. This follows from the intuition that the smaller the number of joins, the easier to interpret the results. The second category ranks the final answers based on the IR score of the tuples that contain the input keywords [28, 53]. In this chapter, we propose a series of methods for ranking the final answers first based on the importance of the nodes/edges of their associated join trees, and then based on an IR score as a secondary ranking measure. We believe this approach leads to find more meaningful and suitable results for the users.

Below, we discuss three recent works in the area of finding meaningful answers for keyword search over relational database, pointing out their differences from our work.

Authors of [7] propose a framework for keyword search in relational databases. For building the index, traditional keyword search methods require access to the actual data stored in the RDBMS. In contrast, the method proposed in [7] uses intensional knowledge. Therefore, it can be used in applications in which building and maintaining specialized indexes is not feasible. Such systems only allow access to the data through predefined queries, wrappers or web forms. For interpreting the role of each keyword in the query, the authors extend the Hungarian algorithm [11] for finding the tuples that are most likely related to the meaning of the keyword. Our work could be considered as the next step of this work in which we rank the set of interconnected tuple sets for given pairs of keyword-entity.

The SODA (Search Over DAta Warehouse) system is presented in [9]. The purpose is to enable end users to explore large data warehouses with complex schemas by applying the keyword search approach. SODA is built based on the idea of applying a graph pattern matching algorithm to generate SQL queries based on the given keywords. The focus of the system is to disambiguate the meaning of words using the joins and inheritance relationships among the matching tables. Our system is different from [9] in that we do not use metadata or other extra information. Our ranking model uses the schema and its instance to find the most meaningful relationships between the roles of query keywords. In addition, our focus is to rank the join trees rather than the roles of the keywords.

CI-RANK is introduced in [71] as a new approach for keyword search in databases. The authors considers the importance of individual nodes in an answer as well as the cohesiveness of the structure of the answer. Random Walk with Message Passing is used to built CI-RANK which captures the relationships between nodes in the answer. A branch and bound algorithm is designed for generating top-k answers. However, the results are only evaluated on DBLP and IMDb datasets which has a simple and non-complex schema.

2.3 Team Formation in Social Networks

Discovering a team of experts in a social network is introduced in [48], in which two communication cost functions are proposed. Authors of [50] generalize this problem by associating each required skill with a specific number of experts, but no approximation ratio is provided for the algorithms. The authors of [34] propose the *sum of distances* communication function and a 2-approximation algorithm for minimizing the *sum of distances*. They also introduce the problem of finding a team of experts with a leader.

The authors of [20] propose another communication cost function based on the density of the induced subgraph on selected nodes. They also reported improvements over [48]. Authors of [2] minimize the maximum load of the experts in the presence of several tasks. They do not consider finding teams with low communication cost. Recently, the problem of online team formation is studied in [3], which creates teams of experts with minimized work load and communication cost. Balancing the work load while minimizing the communication cost is also studied in [15]. The personnel cost of the experts is not considered in [3, 15]. In [39], the authors propose to find a team of experts while minimizing both communication and personnel cost. They merged the two objective functions into one function using an input threshold from the user. In this work, we solve the problem using two fundamentally different approaches, finding the solutions within the given budget and finding the Pareto front.

Bicriteria team formation is considered in [12, 66]. One objective is the communication/collaboration cost and the other is the level of skills of the experts. They do not consider the personnel cost of the team. More importantly, these methods convert the bi-criteria functions into a single one by using a weighted sum of two objective functions. As a result, the methods require the specification of a weight that measures the importance of each objective. If the weight value is not chosen properly, the result may not be reliable. In addition, they generate only one (approximate) solution using a simulated annealing or genetic algorithm and no approximation bound is given. They do not consider Pareto solutions. Finding Pareto solutions in other or general domains has been studied. Most of the proposed solutions are based on evolutionary (such as genetic) algorithms [25]. No performance bound can be found for such algorithms.

The problem of team formation has also been studied in the operation research community. Branch and bound, simulated annealing and genetic algorithms are used for solving the problem [19]. The main difference between our work and the works in operation research is that they do not consider the network connecting the experts. Expert search from disparate contents (e.g. web pages) is a related line of research in information retrieval [23]. The purpose is to find the experts and rank them based on their expertise level. In this work, the set of experts and their expertise are given and our purpose is to form a team from this set.

The authors of [21], consider the effect of different graph structures among the members on the performance of the team. However, they performed their studies in an experimental setting and they do not study the problem from a computational point of view. Dynamics of group formation and its effect on the formation of groups in the network is studied in [5]. A game-theoretic approach to this problem is presented in [29]. Although these studies are not directly related to our setting, they might be considered as a complementary work for our problem.

Another line of research in the database community related to finding Pareto sets is the skyline computation [10, 57]. A skyline of a set of objects (i.e. records) contains all the records that are not dominated by any other record, which is the same as a Pareto set. However, in skyline computation, the set of records from which a skyline is found is given in the database. Assuming n is the number of records, a naive algorithm is able to compute the skyline in $O(n \log n)$ [10]. The main purpose of the skyline algorithms is to reduce this complexity. In contrast, the possible teams in our work is not given and our algorithms have to walk through a search space to find the (approximate) best or Pareto-optimal teams. The number of possible teams is exponential with respect to the number of required skills. Thus, it is not feasible to produce all of the teams and then find the Pareto set from it (i.e. run a skyline algorithm on all of the teams).

3 Keyword Search in Graphs: Finding r-cliques

Keyword search from a graph finds a substructure of the graph containing all or some of the input keywords. Most of the previous methods in this area find connected minimal trees that cover all the query keywords. Recently, it has been shown that finding subgraphs rather than trees can be more useful and informative for the users. However, the current tree or graph based methods may produce answers in which some content nodes (i.e., nodes that contain input keywords) are not very close to each other. In addition, when searching for answers, these methods may explore the whole graph rather than only the content nodes. This may lead to poor performance in execution time. To address the above problems, we propose to find r-cliques in graphs. An r-clique is a group of content nodes that cover all the input keywords and the distance between each two nodes is less than or equal to r. An exact algorithm is proposed that finds all r-cliques in the input graph. An approximation algorithm that produces r-cliques with 2-approximation in polynomial delay is proposed. We further decrease the run time of the approximation algorithm with the cost of increasing the approximation ratio. Extensive performance studies using three large real data sets confirm the efficiency and accuracy of finding r-cliques in graphs.

3.1 Introduction

Keyword search is a well known mechanism for extracting relevant information from a set of documents and web pages. It has been studied for extracting information from structured data in recent years. Structured data such as XML documents and relational databases are usually modeled as graphs. In such models, keyword search plays an important role in finding useful information for users. Users are usually not familiar with the structure of data. In addition, they do not have sufficient knowledge about query languages such as SQL. Therefore, they need a simple system that receives some keywords as input and returns a set of nodes that together cover all or part of the input keywords. A node that contains one or more keywords is called a content node.

Most of the works in keyword search over graphs find minimal connected trees that contain all or part of the input keywords [59, 70]. A tree that covers all the input keywords with the minimum sum of edge weights is called *Steiner tree*. Recently, methods that produce graphs are proposed, which provide more informative answers. However, these tree or graph based methods have the following problems. First, while some of the content nodes in the resulting trees or graphs are close to each other, there might be content nodes in the answer that are far away from each other. It means that weak relationships among content nodes might exist in the trees or graphs. We argue that, assuming all the keywords are equally important, results that contain strong relationships (i.e., short distances) between each pair of content nodes should be preferable over the ones containing weak relationships. Second, current graph or tree based methods explore both content and non-content nodes in the graph while searching for the result. Since there may be thousands or even millions of nodes in an input graph, these methods have high time and memory complexity.

In this chapter, we propose to find r-cliques as a new approach to the keyword search problem. An r-clique is a set of content nodes that cover all the input keywords. In addition, the shortest distance between each pair of nodes is no larger than r. The benefits of finding r-cliques are as follows. First, in an r-clique all pairs of the content nodes are close to each other (i.e, within r distance). Second, there is no need to explore all the nodes in the input graph when finding r-cliques if a proper index is built. This reduces the search space by orders of magnitude. To illustrate the differences between r-clique and other approaches (e.g., Steiner tree [8, 24] and community [60]), an example is given below.

Suppose the nodes in an input graph are web pages of researchers and their organizations. Two nodes are connected by an edge if there is a link from one page to the other. Assume the weight on each edge is 1. Let's assume that the user would like to find a collection of pages that contain *James*, *John* and *Jack*. Given such input keywords, our

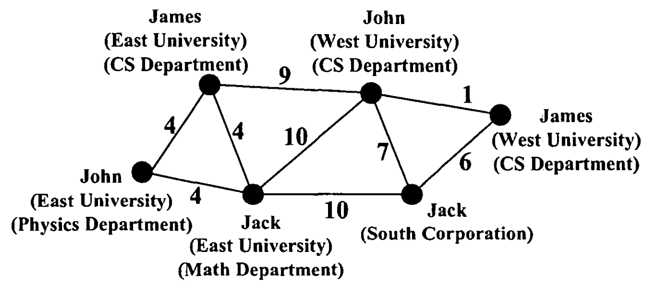


Figure 3.1: A sample graph. The shortest distance between a pair of nodes is shown on their edge.

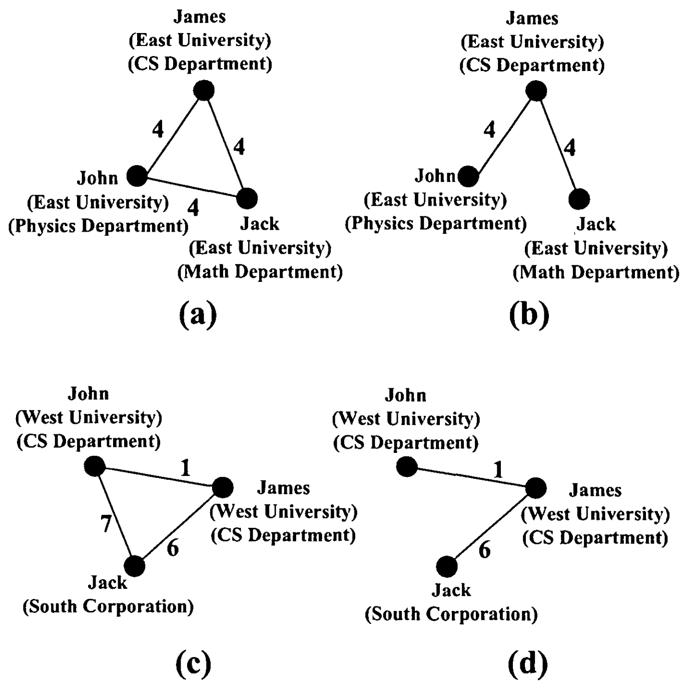


Figure 3.2: Two different answers (a) and (c) and their Steiner trees (b) and (d) over the sample graph

method will reduce the size of the input graph by keeping only the nodes that contain at least one of the input keywords. Assume that there are only 6 nodes containing the above keywords and the reduced graph is shown in Figure 3.1, in which the shortest distance between each pair of nodes is shown in the edge that connects the two nodes. Assuming $r = 10$, edges with distance larger than 10 are ignored. Our method will produce ranked r-cliques, two of which are shown in Figure 3.2 (a) and (c). In Figure 3.2 (a) each node contains one input keyword, all the pages are from the same university, and each pair of the pages are related to each other via 3 other pages (which are not shown). A Steiner tree that covers these nodes is presented in Figure 3.2 (b). The r-clique in Figure 3.2 (c) also contains three pages, two of which from the same university but the other one is from a different organization. A Steiner tree covering these three nodes is shown in Figure 3.2 (d). Our method will rank the answer in (a) ahead of the one in (c) because the sum of the distances between each pair of nodes in (a) is 12, while the one in (c) is 14. On the other hand, a method that produces Steiner trees would rank the result in (d) ahead of the one in (b) because the total distance on the tree paths in (d) is 7, while the one in (b) is 8. Since the web pages in (a) and (b) are from the same organization and all close to each other, the ranking produced by our method is reasonably better.

The closest work in the literature to our work is [60]. The authors proposed to find communities as the results of keyword search over graphs. In each community, there are some center nodes which are close to the content nodes. The answers are ranked based on the sum of the distances of content nodes to the centers. In the above example, node *James* is considered as the center node in both answers (a) and (c) because it has the least sum of the distances to other content nodes. The community method will rank the answer in (c) ahead of the one in (a) because the sum of the distances from the center node to the content nodes in (c) is 7, while the one in (a) is 8. However, (a) is better than (c) because the three nodes in (a) are from the same university.

The contributions of this chapter are summarized as follows:

1. We propose a new model for keyword search in graphs that produces r-cliques in which all pairs of content nodes are reasonably close to each other.

2. We prove that finding the r-clique with the minimum weight is an NP-complete problem by a reduction from 3-SAT.

3. An exact algorithm based on branch and bound is proposed for finding all r-cliques.

4. We propose an approximation algorithm that produces r-cliques with an approximation ratio of 2. The algorithm can produce all or top-k r-cliques in polynomial delay in ascending order of their weights.

5. To further speed up the approximation algorithm, another approximation algorithm for producing r-cliques in polynomial delay is proposed. The algorithm is l times faster and has the approximation ratio of $l - 1$, where l is the number of input keywords.

6. To reveal the relationship between the nodes in a generated r-clique, we propose to find a Steiner tree in the graph to connect the nodes in the r-clique after the r-clique is generated. Using a tree instead of a graph reduces the chance of including irrelevant nodes in the final answer.
7. Extensive experimental results on three real datasets along with a user study show the effectiveness of the proposed algorithms in terms of run time, compactness and precision.

The chapter is organized as follows. In Section 3.2, a formal problem statement is given. In Section 3.3, an algorithm based on branch and bound for finding all r-cliques is introduced. The procedure of finding top-k answers in polynomial delay is presented in Section 3.4. An algorithm that produces r-cliques with a 2 approximation ratio is presented in Section 3.5. The run time of this algorithm is improved in Section 3.6 with the cost of increasing the approximation ratio. A method for presenting an r-clique is given in Section 3.7. Graph-indexing method is discussed in Section 3.8. Experimental results are given in Section 3.9. Section 3.10 concludes this chapter.

3.2 Problem Statement

Given a data graph and a query consisting of a set of keywords, the problem of keyword search in a graph is to find a set of connected subgraphs that contain all or part of the keywords. It is preferred that the answers are presented according to a ranking mechanism. The data graph can be directed or undirected. The edges or nodes may have weights on them. In this work, the same as [51], we only consider undirected graphs with weighted edges. Undirected graphs can be used to model different types of unstructured, semi-structured and structured data, such as web pages, XML documents and relational datasets. It should be noted that our approach is easily adaptable to work with directed graphs[2].

Given a data graph G and a query consisting of a set of $l(\geq 2)$ keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), the problem tackled in this chapter is to find a set of r-cliques, preferably ranked in ascending order of their weights. An r-clique and its weight are defined below.

Definition 1 ***(r-clique)*** *Given a graph G and a set Q of input keywords, an r-clique of G with respect to Q is a set of content nodes in G that together cover all the input keywords in Q and in which the shortest distance between each pair of the content nodes is no larger than r. The shortest distance between two nodes is the sum of the weights of the edges in G on the shortest path between the two nodes.*

Definition 2 ***(Weight of r-clique)*** *Suppose that the nodes of an r-clique of a graph G are denoted as $\{v_1, v_2, \ldots, v_l\}$. The weight of the r-clique is defined as*

[2]For directed graphs, the shortest distance between two nodes in an r-clique should be no larger than r in both directions.

$$weight = \sum_{i=1}^{l} \sum_{j=i+1}^{l} dist(v_i, v_j)$$

where $dist(v_i, v_j)$ is the shortest distance between v_i and v_j in G, i.e., the weight on the edge between the two nodes in the r-clique.

r-cliques with smaller weights are considered to be better in this chapter. Thus, the core of our problem can be stated below in Problem 1.

Problem 1 *Given a distance threshold r, a graph G and a set of input keywords, find an r-clique in G whose weight is minimum.*

Theorem 1 *Problem 1 is NP-complete.*

Proof

We prove the theorem by a reduction from 3-satisfiability (3-SAT). We prove that the decision version of the problem presented below is NP-complete. Thus, as a direct result, Problem 1 is NP-complete too. □

Problem 2 *Given a distance threshold r, a graph G and a set of input keywords $Q = \{k_1, \ldots, k_l\}$, determine whether there exists an r-clique with weight w, for some constant w. The weight of the r-clique is defined in Definition 2.*

Theorem 2 *Problem 2, a decision version of Problem 1, is NP-complete.*

Proof

The problem is obviously in NP. We prove the theorem by a reduction from 3-satisfiability (3-SAT)[3]. First, consider a set of m clauses $D_p = x_p \vee y_p \vee z_p$ $(p = 1, \ldots, m)$ and $\{x_p, y_p, z_p\} \subset \{u_1, \overline{u}_1, \ldots, u_n, \overline{u}_n\}$. We set the distance between each variable and its negation (i.e., u_i and $\overline{u}_i$) to $2 \times w$. The distance between other variables is set to $\frac{w}{\binom{n+m}{2}}$. The distance of each variable to itself is set to zero. We define an instance of the above problem as follows. First, r is set to $2 \times w$. For each pair of variables u_i and $\overline{u}_i$, two nodes are created. Thus, we have $2 \times n$ nodes. For each pair of variables u_i and $\overline{u}_i$, we create one keyword k_i $(i = 1, \ldots, n)$. Thus, u_i and $\overline{u}_i$ have keyword k_i and the only holders of k_i are u_i and $\overline{u}_i$. In addition, for every clause D_p, we create one keyword k_{n+p} $(p = 1, \ldots, m)$ such that the holders of keyword k_{n+p} consists of the triplet of nodes associated with those of x_p, y_p and z_p. Therefore, the number of required keywords is $n + m$.

A feasible solution to the above problem with the weight at most w is any set of nodes such that from each pair of nodes corresponding to u_i and $\overline{u}_i$, exactly one is selected and from each triplet of nodes corresponding to x_p, y_p and z_p, one is selected. Thus, if there exists a subset of the weight at most w, then there exists a satisfying assignment for $D_1 \wedge D_2 \wedge \cdots \wedge D_m$. On the other hand, a satisfying assignment apparently determines a feasible set of nodes with the weight at most w. Therefore, the proof is complete. □

[3]It should be noted that the same approach is used in [4] for proving the NP-completeness of *multiple choice cover* problem.

Given a graph G and a set of l input keywords $(k_1, k_2, \ldots, k_l)$, the maximum number of possible r-cliques is $O(|C_{max}|^l)$ where $|C_{max}|$ is the maximum size of C_i $(1 \leq i \leq l)$ and C_i is the set of nodes in G containing keyword k_i. Thus, finding all the r-cliques from G is not feasible when l is large. In this work, we focus on finding top-k r-cliques ranked according to their weights. Since finding the best r-clique from a graph is an NP-complete problem, we hereby propose algorithms for finding approximate best solutions. However, in the next section, we propose an exact algorithm based on branch and bound for producing all r-cliques for evaluating the proposed approximation algorithms. In Section 3.4, we describe our algorithm for finding top-k answers which works by dividing the search space into subspaces. Then, in Sections 3.5 and 3.6, we propose two approximation algorithms for finding an approximate best answer from a search (sub)space.

3.3 Branch and Bound Algorithm

We present a branch and bound algorithm for finding all r-cliques in a graph. The algorithm is based on systematic enumeration of candidate solutions and at the same time using the distance constraint r to avoid generating subsets of fruitless candidates. Note that this method does not rank the r-cliques by their weights. The ranking, if needed, can be done as a post-processing process. This method has exponential run time in the worst case. After ranking the answers, it produces exact and optimal solutions. Please note that it is used just as a **baseline** to compare with the polynomial delay approximation algorithm proposed in the next sections.

The pseudo-code of the algorithm is presented in Algorithm 1. In the first step, the set of nodes that contain each keyword is extracted. This can be easily done using a pre-built inverted index that stores a mapping from a word in the dataset to the list of nodes containing the word. The set of nodes containing keyword k_i is stored in set C_i. C_i^j specifies the jth element of set C_i. The candidate partial r-cliques are stored in a list called $rList$. The basic idea of the algorithm is as follows. First, the content nodes containing the first keyword are added to $rList$. Then, for the second keyword, we compute the shortest distance between each node in C_2 and each node in $rList$. If the distance $\leq r$, a new candidate that combines the corresponding nodes in C_2 and $rList$ is added to a new candidate list called $newRList$. After all pairs of nodes in C_2 and $rList$ have been checked, the content of $rList$ is replaced by the content of $newRList$. The process continues in the same way to consider all of the remaining keywords. The final content of $rList$ is the set of all r-cliques.

To speed up this process, an index (described later in this chapter) is pre-built to store the shortest distance between each pair of nodes. Thus, the shortest path computation is at the unit cost. Assume that the maximum size of C_i ($1 \leq i \leq l$ where l is the number of keywords) is $|C_{max}|$. The complexity of the algorithm is $O(l^2|C_{max}|^{l+1})$.

Algorithm 1 Branch and Bound Algorithm

Input: the input graph G; the query $\{k_1, k_2, \ldots, k_l\}$ and r

Output: the set of all r-cliques

```
1: **for** $i \leftarrow 1$ **to** $l$ **do**
2:     $C_i \leftarrow$ the set of nodes in $G$ containing $k_i$
3: $rList \leftarrow$ empty
4: **for** $i \leftarrow 1$ **to** **size**($C_1$) **do**
5:     $rList$.**add**($C_1^i$)
6: **for** $i \leftarrow 2$ **to** $l$ **do**
7:     $newRList \leftarrow$ empty
8:     **for** $j \leftarrow 1$ **to** **size**($C_i$) **do**
9:         **for** $k \leftarrow 1$ **to** **size**($rList$) **do**
10:            **if** $\forall$ node $\in rList_k$ **dist**($node$,$C_i^j$) $\leq r$ (where $rList_k$ is the $k$th element of
               $rList$) **then**
11:                $newCandidate \leftarrow rList_k$.**add**($C_i^j$)
12:                $newRList$.**add**($newCandidate$)
13:    $rList \leftarrow newRList$
14: **return** $rList$
```

3.4 Top-k Polynomial Delay Algorithm

An efficient search engine should satisfy three properties [22]. First, it should be able to generate all answers without missing them. Second, the answers should be presented in an order with better answers ranked higher. Third, the search engine should produce the answers efficiently. A standard measure of efficiency is whether the search engine can produce answers with polynomial delay. In other words, the efficiency of a search engine is calculated based on the delay between producing two consecutive answers. If this delay is polynomial based on the input data, the algorithm is called a polynomial delay algorithm [22].

Our algorithm for producing a ranked list of search results is an adaption of Lawler's procedure [49] for computing the top-k answers to discrete optimization problems. In Lawler's procedure, the search space is first divided into disjoined sub-spaces; then the best answer in each subspace is found and used to produce the current global best answer. The sub-space that produces the best global answer is further divided into sub-subspaces and the best answer among its sub-subspaces is used to compete with the best answers in other sub-spaces in the previous level to find the next best global answer. Two main issues in this procedure are how to divide a space into subspaces and how to find the best answer within a (sub)space.

We first informally describe our idea of dividing the search space in our problem into

Table 3.1: An Example of dividing the search space

Subspace	Representative set
SB_0	$\{v_1\} \times \{v_2\} \times \{v_3\} \times \{v_4\}$
SB_1	$[C_1 - \{v_1\}] \times C_2 \times C_3 \times C_4$
SB_2	$\{v_1\} \times [C_2 - \{v_2\}] \times C_3 \times C_4$
SB_3	$\{v_1\} \times \{v_2\} \times [C_3 - \{v_3\}] \times C_4$
SB_4	$\{v_1\} \times \{v_2\} \times \{v_3\} \times [C_4 - \{v_4\}]$

subspaces using an example[4]. Suppose that the input query consists of four keywords, i.e., $\{k_1, k_2, k_3, k_4\}$. Let C_i be the set of nodes in graph G that contains input keyword k_i. Thus, the search space that contains the best answer can be represented as $C_1 \times C_2 \times C_3 \times C_4$. From this space, we use the *FindBestAnswer* procedure (to be described in the next section) to find the best (approximate) answer in polynomial time. Assume that the best answer is (v_1, v_2, v_3, v_4), where v_i is a node in graph G containing keyword k_i. Based on this best answer, the search space is divided into 5 subspaces SB_0, SB_1, SB_2, SB_3 and SB_4 as shown in Table 3.1, where SB_0 contains only the best answer. The union of the subspaces cover the whole search space.

After finding the best answer and dividing the search space into subspaces, the best answer in each subspace except subset SB_0 is found using the *FindBestAnswer* proce-

[4]Our approach to dividing a search space is similar to the idea used in [60].

dure. These best answers are inserted into a priority queue, where the answers are ranked in ascending order of their weights. Obviously, the second best answer is the one at the top of the priority queue. Suppose that this answer is taken from SB_2. After returning the second best answer, SB_2 is divided into 5 subspaces in the way similar to the one shown in Table 3.1. In each subspace (except the first one), the best answer is found and is inserted to the right place of the priority queue according to its weight. After the best answer for each subspace is inserted, the top answer in the queue is returned and removed from the queue, its corresponding space is divided into subspaces and the best answer (if any) in each new subspace is added to the priority queue. This procedure continues until the priority queue becomes empty.

The pseudo-code of our algorithm for finding top-k answers is presented in Algorithm 2 (i.e. $GenerateTopkAnswers$). The structure of the algorithm is similar to other polynomial delay algorithms [22, 60]. It is modified to perform in the setting of producing ranked r-cliques from a graph. The algorithm takes graph G, query $\{k_1, k_2, \ldots, k_l\}$, the distance threshold r and k as input. It searches for answers and outputs top-k of them in ascending order according to their weights. In lines 1 and 2, the algorithm computes sets C_i, the set of the nodes containing keyword k_i. This can be easily done using a pre-built inverted index. Then, the collection of sets C_i is called C in line 3. It should be noted that C is the whole search space that contains keyword nodes and the first best answer should be found in this space. In line 5, procedure $FindBestAnswer$ (to be discussed later) is

Algorithm 2 GenerateTopkAnswers Algorithm

Input: the input graph G; the query $\{k_1, k_2, \ldots, k_l\}$; r and k

Output: the set of top-k ordered r-cliques printed with polynomial delay

```
1: for $i \leftarrow 1$ to $l$ do
2:     $C_i \leftarrow$ the set of nodes in $G$ containing $k_i$
3: $C \leftarrow \langle C_1, C_2, \ldots, C_l \rangle$
4: $Queue \leftarrow \emptyset$
5: $A \leftarrow$ FindBestAnswer($C$, $G$, $l$, $r$)
6: if $A \neq \emptyset$ then
7:     insert $\langle A, C \rangle$ into $Queue$
8: while $Queue \neq \emptyset$ do
9:     $\langle A, S \rangle \leftarrow$ top element of $Queue$
10:    output($A$)
11:    $k \leftarrow k - 1$
12:    if $k = 0$ then
13:        return
14:    $\langle SB_1, SB_2, \ldots, SB_l \rangle \leftarrow$ ProduceSubSpaces($A$, $S$)
15:    for $i \leftarrow 1$ to $l$ do
16:        $A_i \leftarrow$ FindBestAnswer($SB_i$, $G$, $l$, $r$)
17:        if $A_i \neq \emptyset$ then
18:            $Queue$.insert($\langle A_i, SB_i \rangle$)
```

called to find the best answer in space C in polynomial time. If the best answer exits (i.e., A is not empty), A, together with its related space C, is inserted into $Queue$ in line 7. The $Queue$ is maintained in the way that its elements are ordered in ascending order of their weights. The while loop starting at line 8 is executed until the $Queue$ becomes empty or k answers have been outputted. In line 9, the top of the $Queue$ is removed. The top of the $Queue$ contains the best answer in the $Queue$ and the space that this answer is produced from. We assign this space to S and the best answer to A. The answer in A is outputted in line 10. Then if the number of answers has not reached k, procedure $ProduceSubSpaces$ is called to produce l new subspaces of S based on the current answer A. These subspaces are shown by SB_i. In lines 15-18, these new subspaces are explored. For each subspace, the best answer is found and inserted into the $Queue$ with its related subspace. [5] Clearly, if procedures $FindBestAnswer$ and $ProduceSubSpaces$ terminate in polynomial time, then algorithm $GenerateTopkAnswers$ produces answers with polynomial delay.

The pseudo-code of algorithm $ProduceSubSpaces$ is presented in Algorithm 3. It takes as input the best answer A of the previous step and the search space S from which A is generated. It produces l new subspaces, $\langle SB_1, \ldots, SB_l \rangle$. In the procedure, SB_i^j

[5]Note that, unlike tree-based methods, this procedure produces duplication-free answers (i.e., the set of content nodes in an answer is unique compared to other answers in the top-k list) if no content node contains more than one input keyword in the input graph. But if a node contains more than one input keyword, the procedure may produce duplicate answers although the answers are unique in terms of keyword-node couplings. In this regard, our result is the same as the top-k result in [60], where the authors consider such answers duplication-free because of different keyword-node couplings in the top-k answers. In Chapter 4, a procedure is proposed that produces completely duplication-free set of answers with respect to the set of content nodes in an answer.

specifies the j-th element in vector SB_i. It is a polynomial procedure and runs in $O(l^2)$.

Algorithm 3 ProduceSubSpaces Procedure

Input: the best answer of previous step, $A = \langle v_1, v_2, \dots v_l \rangle$, and search space $S = \langle S_1, \dots, S_l \rangle$

Output: l new subspaces

```
1: for $i \leftarrow 1$ to $l$ do
2:     for $j \leftarrow 1$ to $i - 1$ do
3:         $SB_i^j \leftarrow \{v_j\}$
4:     $SB_i^i \leftarrow S_i - \{v_i\}$
5:     for $j \leftarrow i + 1$ to $l$ do
6:         $SB_i^j \leftarrow S_j$
7: return $\langle SB_1, \dots, SB_l \rangle$ where $SB_i = \langle SB_i^1, \dots, SB_i^l \rangle$ representing space $SB_i^1 \times \dots \times SB_i^l$
```

3.5 Finding Best Answer from a Search Space with 2-approximation

Since finding the best r-clique is NP-complete, in this section we propose an approximation algorithm that produces an answer whose weight is at most twice that of an optimal r-clique. The basic idea of the algorithm is as follows. Given a search space $S = \langle S_1, \dots, S_l \rangle$, where S_i is a set of nodes containing input keyword k_i, the algorithm uses each node in S_i (for $i = 1, \dots, l$) as the initial node and *center* of a candidate answer. It then adds to the answer the node in each of other S_i's which is closest to the center. Thus, the candidate answer contains all the input keywords. The answer is evaluated using the sum of distances from the center to each of other nodes in the answer. The answer with the least sum of the center distances is outputted as an approximate best r-clique.

The pseudo-code of algorithm $FindBestAnswer$ is presented in Algorithm 4. In the code, variable s_i^j denotes the j-th node of set S_i, and $n(s_i^j, k)$ denotes the node in S_k which has the shortest distance to s_i^j, i.e., the nearest neighbor of s_i^j in S_k . In lines 1-3, the nearest neighbor of a node s_i^j in its own set S_i is set to itself. The rest of the code iterates on each node in S_i (for $i = 1, \ldots, l$), and considers such a node as the *center* of a candidate answer. For each of such nodes, s_i^j, the algorithm finds its nearest neighbor in S_k where $k \neq i$ (lines 9-15). If the shortest distance between s_i^j and its nearest neighbor in S_k is less than or equal to r, the neighbor is added to the candidate answer, and the weight of the answer is updated by adding the shortest distance between s_i^j and the neighbor. If the shortest distance is greater than r, the weight of the answer is set to ∞, and the formulation of this candidate answer is stopped. If, for all S_k ($k = 1, \ldots, l$ and $k \neq i$), the shortest distance between s_i^j and its nearest neighbor in S_k is no larger than r, a candidate answer is formed and the weight of the answer is the sum of the shortest distances between the center node s_i^j and its nearest neighbor in S_k for $k = 1, \ldots, l$. Finally, the candidate answer with the least weight is outputted as the

Algorithm 4 FindBestAnswer Procedure

Input: the search space $S = \langle S_1, S_2, \ldots, S_l \rangle$; the input graph G; the number of query keywords l and r

Output: the best r-clique in the search space S

```
1: for $i \leftarrow 1$ to $l$ do
2:     for $j \leftarrow 1$ to size($S_i$) do
3:         $n(s_i^j, i) \leftarrow s_i^j$
4: $leastWeight \leftarrow \infty$
5: $topAnswer \leftarrow \emptyset$
6: for $i \leftarrow 1$ to $l$ do
7:     for $j \leftarrow 1$ to size($S_i$) do
8:         $weight \leftarrow 0$
9:         for $k \leftarrow 1$ to $l$ ; $k \neq i$ do
10:            $shortestdist = \infty$
11:            for $t \leftarrow 1$ to size($S_k$) do
12:                $dist \leftarrow$ shortest distance from $s_i^j$ to $s_k^t$
13:                if $dist < shortestdist$ then
14:                    $shortestdist \leftarrow dist$
15:                    $n(s_i^j, k) \leftarrow s_k^t$
16:            if $shortestdist \leq r$ then
17:                $weight \leftarrow weight + shortestdist$
18:            else
19:                $weight \leftarrow \infty$
20:                break
21:        if $weight < leastWeight$ then
22:            $leastWeight \leftarrow weight$
23:            $topAnswer \leftarrow \langle n(s_i^j, 1), \ldots, n(s_i^j, l) \rangle$
24: return $topAnswer$
```

best answer (denoted as $topAnswer$ in the pseudo-code).

Clearly, all of the above operations can be done in polynomial time. Since a pre-built index (described later in this chapter) is used for finding the shortest path between a pair of nodes, the shortest path computation is at the unit cost. Thus, the complexity of this algorithm is $O(l^2|S_{max}|^2)$, where $|S_{max}|$ is the maximum size of S_i for $1 \leq i \leq l$.

It should be noted that the answer returned by this approximation algorithm may not be an r-clique. In the worst cast, the distance between a pair of nodes in the answer is $2 \times r$, as stated in the following theorem.

Theorem 3 *The distance between each pair of nodes in the answer produced by procedure $FindBestAnswer$ is at most* $2 \times r$.

Proof

In the answer produced by algorithm $FindBestAnswer$, there is a *center* node (i.e., s_i^j in Algorithm 4) that has distance less than or equal to r to each of the other nodes in the answer. Let's call this center node a. Assume that b and c are two other nodes in the answer. Since shortest distances satisfy the triangle inequality, we have:

$$d_{bc} \leq d_{ab} + d_{ac}$$

where d_{bc} is the shortest distance between nodes b and c and so on. Since $d_{ab} \leq r$ and $d_{ac} \leq r$, we have:

$$d_{bc} \leq d_{ab} + d_{ac} \leq r + r \leq 2 \times r$$

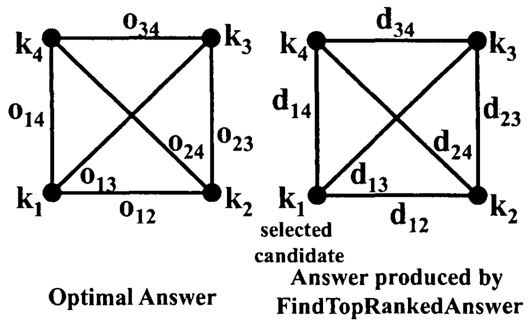


Figure 3.3: The optimal answer and the answer which is produced by procedure *FindBestAnswer*.

□

Although Algorithm 4 is not guaranteed to produce an r-clique, we will show in the experiment section that high percentages of the answers produced by Algorithm 4 are r-cliques. For the convenience reason, below we still refer to an answer from this algorithm as r-clique. The following theorem shows that Algorithm 4 produces an answer whose weight is at most twice that of an optimal r-clique.

To prove the theorem, we first give an example and then present a formal proof. Consider the example presented in Figure 3.3 with four input keywords. One of the answers is the optimal answer and the other one is the candidate answer produced by procedure *FindBestAnswer*. Without the loss of generality, we assume that the node for keyword k_1 is the best candidate node (i.e., the best s_i^j) selected by the procedure. Since the sum of the weights on edges connected to k_1, i.e. d_{12}, d_{13} and d_{14}, in the selected candidate is the smallest among all the content nodes whose connected edges have a weight less than or equal to r, the following expressions hold:

$$\begin{cases} k_1: & o_{12}+o_{13}+o_{14} \geq d_{12}+d_{13}+d_{14} \\ k_2: & o_{12}+o_{23}+o_{24} \geq d_{12}+d_{13}+d_{14} \\ k_3: & o_{13}+o_{23}+o_{34} \geq d_{12}+d_{13}+d_{14} \\ k_4: & o_{14}+o_{24}+o_{34} \geq d_{12}+d_{13}+d_{14} \end{cases} \tag{3.1}$$

Summing up both sides of the above equations, we have:

$$2(o_{12}+o_{13}+o_{14}+o_{23}+o_{24}+o_{34}) \geq 4(d_{12}+d_{13}+d_{14}) \tag{3.2}$$

Since the distance between each pair of nodes is the shortest distance between them, the triangle inequality is satisfied and the following equations hold:

$$\begin{cases} d_{12}+d_{13} \geq d_{23} \\ d_{12}+d_{14} \geq d_{24} \\ d_{13}+d_{14} \geq d_{34} \end{cases} \tag{3.3}$$

The weight of the selected candidate produced by procedure *FindBestAnswer* is $d_{12}+d_{13}+d_{14}+d_{23}+d_{24}+d_{34}$. Based on Equation 3.3, the candidate weight is at most $3 \times (d_{12}+d_{13}+d_{14})$. Thus, after some basic calculations and based on Equation 3.2, the following is valid:

$$\frac{2 \times 3}{4}(o_{12}+o_{13}+o_{14}+o_{23}+o_{24}+o_{34}) \geq 3 \times (d_{12}+d_{13}+d_{14}) \tag{3.4}$$

The left side of the equation is at most twice the weight of the optimal answer and the right side of the equation is at most the weight of the selected candidate. Thus, in the worst case, the weight of the selected candidate is twice the weight of the optimal answer. Now we are ready to present the formal proof in detail.

Theorem 4 *Procedure $FindBestAnswer$ produces an r-clique with 2-approximation.*

Proof

Consider two answers, one *optimal answer* and the answer produced by $FindBestAnswer$ (denoted here as *approx answer*). We denote the node in *approx answer* whose sum of shortest distances to the other nodes in *approx answer* is the smallest as *center node*. Note that this sum of shortest distances that the *center node* has in the *approx answer* is the smallest among the ones of all the other content nodes in the input graph in any other possible answers including the *optimal answer*. Without loss of generality, assume that the *center node* contains the first keyword, i.e. k_1. Let's call the shortest distances of the *center node* to other nodes in the *approx answer* $d_{12}, d_{13}, \ldots, d_{1l}$, where l is the number of input keywords. Thus, based on the $FindBestAnswer$ procedure, $\sum_{i=2}^{l} d_{1i}$ has the smallest value among all other content nodes in the graph. Thus, for each node containing $k_j (1 \leq j \leq l)$ in the *optimal answer*, we have:

$$o_{1j} + o_{2j} + \cdots + o_{j-1j} + o_{jj+1} + \cdots + o_{jl} \geq d_{12} + d_{13} + \cdots + d_{1l} \qquad (3.5)$$

where o_{ij} is the shortest distance between the node containing keyword k_i and the node containing k_j in the *optimal answer*. Thus,

$$\sum_{i=1}^{j-1} o_{ij} + \sum_{i=j+1}^{l} o_{ji} \geq \sum_{i=2}^{l} d_{1i} \qquad (3.6)$$

If we write the above equation for all l content nodes of the *optimal answer* and sum up both sides of the inequalities, we have:

$$2 \times \sum_{i=1}^{l} \sum_{j=i+1}^{l} o_{ij} \geq l \times \sum_{i=2}^{l} d_{1i} \tag{3.7}$$

Since each distance relates two content nodes, each distance appears in the left side of the equation twice. The left side of the above equation is twice the weight of the *optimal answer*. Thus, the following is valid:

$$2 \times (optimal\ weight) \geq l \times \sum_{i=2}^{l} d_{1i} \tag{3.8}$$

The weight of the *approx answer* is as follows:

$$approx\ weight = \sum_{i=1}^{l} \sum_{j=i+1}^{l} d_{ij} = \sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} d_{ij} \tag{3.9}$$

Since the distance between each pair of nodes in the *approx answer* is the shortest distance between them, the triangle inequality is satisfied:

$$d_{ij} \leq d_{1i} + d_{1j}, i \neq j \neq 1 \tag{3.10}$$

Thus, the following holds:

$$\sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} d_{ij} \leq \sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} (d_{1i} + d_{1j}) \tag{3.11}$$

In the right side of the above equation, each edge d_{1i} is appeared exactly $l - 1$ times. Thus, we have:

$$\sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} (d_{1i} + d_{1j}) = (l-1) \times \sum_{i=2}^{l} d_{1i} \tag{3.12}$$

As a result, we have:

$$approx\ weight \leq (l-1) \times \sum_{i=2}^{l} d_{1i} \tag{3.13}$$

Based on equations 3.8 and 3.13, we have:

$$approx\ weight \leq \frac{2 \times (l-1)}{l}(optimal\ weight) \tag{3.14}$$

It proves that the weight of the *approx answer* is at most twice the weight of the *optimal answer*. $\square$

3.6 Improving the Runtime Performance

Algorithm 4 finds an approximately best r-clique in polynomial time with 2-approximation, but the run time is quadratic in both l and $|S_{max}|$, where l is the number of input keywords and $|S_{max}|$ is the maximum size of S_i for $1 \leq i \leq l$. To improve its run time performance, in this section we propose a variation of Algorithm 4. The new algorithm is called $FindBestAnswerRare$ and is presented in Algorithm 5.

The main difference between this algorithm and Algorithm 4 is that instead of considering each node that contains at least one input keyword as the initial member (i.e., center) of a candidate answer, the new algorithm only considers the nodes which contain the rarest input keyword (i.e., the input keyword that is contained by the least number of nodes in the graph compared to other input keywords). In this way, the for-loop at line

Algorithm 5 FindBestAnswerRare Procedure

Input: the search space $S = \langle S_1, S_2, \ldots, S_l \rangle$; the input graph G; the number of query keywords l and r

Output: the best r-clique in the search space S

```
1: $rare \leftarrow \arg\min |S_i|, 1 \leq i \leq l$
2: for $j \leftarrow 1$ to size($S_{rare}$) do
3:     $n(s^j_{rare}, rare) \leftarrow s^j_{rare}$
4: $leastWeight \leftarrow \infty$
5: $topAnswer \leftarrow \emptyset$
6: for $j \leftarrow 1$ to size($S_{rare}$) do
7:     $weight \leftarrow 0$
8:     for $k \leftarrow 1$ to $l$ ; $k \neq rare$ do
9:         $shortestdist = \infty$
10:        for $t \leftarrow 1$ to size($S_k$) do
11:            $dist \leftarrow$ shortest distance from $s^j_{rare}$ to $s^t_k$
12:            if $dist < shortestdist$ then
13:                $shortestdist \leftarrow dist$
14:                $n(s^j_{rare}, k) \leftarrow s^t_k$
15:        if $shortestdist \leq r$ then
16:            $weight \leftarrow weight + shortestdist$
17:        else
18:            $weight \leftarrow \infty$
19:            break
20:    if $weight < leastWeight$ then
21:        $leastWeight \leftarrow weight$
22:        $topAnswer \leftarrow \langle n(s^j_{rare}, 1), \ldots, n(s^j_{rare}, l) \rangle$
23: return $topAnswer$
```

6 in Algorithm 4 is not needed and the algorithm becomes at least l times faster. The pseudo-code of Algorithm 5 is similar to that of Algorithm 4 with the following differences. (1) In line 1, the index of the rarest input keyword is identified and it is called $rare$. (2) Nearest neighbors are only calculated for the nodes that contain the $rare$st keyword. (3) Only the nodes that contain the $rare$st keyword are considered as the center of a candidate answer, and thus the outermost loop in Algorithm 4 starting at line 6 is removed. Hence, the complexity of this algorithm is $O(l \times |S_{max}| \times |S_{min}|)$, where $|S_{max}|$ and $|S_{min}|$ are the maximum and minimum size of S_i for $1 \leq i \leq l$ respectively.

The same as $FindBestAnswer$, the answer returned by this approximation algorithm may not be an r-clique. In the worst cast, the distance between a pair of nodes in the answer is $2 \times r$. It can be proved using the similar technique in Theorem 3. However, making Algorithm 4 faster comes with the cost of increasing the approximation ratio. In the next theorem, we formally prove that the approximation ratio is not worse than $(l - 1)$.

Theorem 5 *Procedure $FindBestAnswerRare$ produces an r-clique with $(l-1)$-approximation where l is the number of input keywords.*

Proof

Consider two answers, one *optimal answer* and the answer produced by procedure $FindBestAnswerRare$ (denoted here as *approx answer*). Below we show that the weight of the *approx answer* is at most $(l-1)$ times the weight of the *optimal answer* where l is the number of input keywords.

Without loss of generality, assume that the first keyword (i.e. k_1), is the rarest keyword. The node in the *approx answer* containing k_1 is called *center node*. Based on the $FindBestAnswerRare$ procedure, the sum of shortest distances from the *center node* to the other nodes in the *approx answer* is the smallest among the sums of shortest distances from all other content nodes that contain k_1 in the input graph to other nodes in an answer.

Let's denote the shortest distances between the *center node* to the other nodes in the *approx answer* as $d_{12}, d_{13}, \ldots, d_{1l}$. Also, we denote the shortest distances from the node in the *optimal answer* that contains k_1 to other nodes in the answer as $o_{12}, o_{13}, \ldots, o_{1l}$. Thus, the following holds:

$$o_{12} + o_{13} + \cdots + o_{1l} \geq d_{12} + d_{13} + \cdots + d_{1l} \tag{3.15}$$

Obviously,

$$\sum_{i=1}^{l} \sum_{j=i+1}^{l} o_{ij} \geq \sum_{i=2}^{l} o_{1i} = o_{12} + o_{22} + \cdots + o_{1l} \tag{3.16}$$

Therefore, we have

$$optimal\ weight \geq \sum_{i=2}^{l} d_{1i} \tag{3.17}$$

The weight of the *approx answer* is as follows:

$$approx\ weight = \sum_{i=1}^{l} \sum_{j=i+1}^{l} d_{ij} = \sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} d_{ij} \tag{3.18}$$

Since the shortest distances satisfy triangle inequality, we have:

$$d_{ij} \leq d_{1i} + d_{1j}, i \neq j \neq 1 \tag{3.19}$$

Thus, the following is valid:

$$\sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} d_{ij} \leq \sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} (d_{1i} + d_{1j}) \tag{3.20}$$

In the right side of the above equation, each edge d_{1i} appears exactly $l - 1$ times. Thus, we have:

$$\sum_{i=2}^{l} d_{1i} + \sum_{i=2}^{l} \sum_{j=i+1}^{l} (d_{1i} + d_{1j}) = (l-1) \times \sum_{i=2}^{l} d_{1i} \tag{3.21}$$

As a result, we have:

$$approx\ weight \leq (l-1) \times \sum_{i=2}^{l} d_{1i} \tag{3.22}$$

Based on equations 3.17 and 3.22, we have:

$$approx\ weight \leq (l-1) \times (optimal\ weight) \tag{3.23}$$

□

3.7 Presenting r-cliques

Each r-clique is a unique set of content nodes that are close to each other and cover the input keywords. However, sometimes it is not sufficient to only show the set of content nodes discovered. It is also important to see how these nodes are connected together in the input graph. To show the relationship between the nodes in an r-clique, we further find a Steiner tree from the input graph which connects the nodes in the r-clique with the minimum sum of edge weights. The leaves of the tree are the content nodes of the r-clique, and its internal node can be a content node in the r-clique or an intermediate node that connects the content nodes.

The reason for choosing a Steiner tree instead of a graph to present an r-clique is that it potentially minimizes the number of intermediate nodes, which decreases the chance of having irrelevant nodes in the answer presented to the user. In the next section, we will show that the community-based method [60] (which returns a graph) tends to include more irrelevant nodes in its answer.

The algorithm for finding a Steiner tree given an r-clique is presented below and is based on the algorithm in [47]. Given a set of nodes, S, that belong to graph G, the Steiner tree problem is to find a tree of graph G that spans S with the minimal total distance on the edges of the tree. This is a well known NP-complete problem [41]. A heuristic algorithm was introduced in [47] to find a Steiner tree from a graph G given a set S of nodes in G. The algorithm in [47] first finds the shortest path in G between each pair of nodes in S and builds a complete graph, G_1, whose nodes are the nodes in S and whose edge between each two nodes is weighted by the total distance on the shortest path between the two nodes in G. It then finds a minimal spanning tree, T_1, of G_1, and constructs a subgraph G_2 of G by replacing each edge of T_1 by its corresponding shortest path in G. Finally, it finds a minimal spanning tree, T_2, of G_2, and constructs a Steiner tree from T_2 by deleting leaves and their associated edges from the tree so that all the leaves are Steiner points.

Algorithm 6 Generating Steiner Tree Algorithm based on an algorithm introduced in [47]

```
Input: an $r$-clique and graph $G$
Output: the Steiner tree of $G$ that spans the nodes in the $r$-clique
1: Let $G_1$ be a complete graph whose nodes are the nodes in the input $r$-clique and
   whose edge between a pair of nodes is weighted by the shortest distance between the
   two nodes in $G$.
2: Find the minimal spanning tree $T_1$ of $G_1$.
3: Create graph $G_2$ by replacing each edge in $T_1$ by its corresponding shortest path in
   $G$. The shortest path can be obtained by using the neighbor index on $G$ described in
   the next section.
4: Find the minimal spanning tree $T_2$ of $G_2$.
5: Create an Steiner tree from $T_2$ by removing the leaves (and the associated edges) that
   are not in the $r$-clique.
```

We make use of this procedure to find a Steiner tree for an r-clique. The pseudo-code of the algorithm is presented in Algorithm 6. The input to our procedure is an r-clique and the graph G from which the r-clique was generated. The output of the algorithm is a Steiner tree of G that spans all the nodes in the r-clique. The Steiner tree produced by this heuristic algorithm is not necessarily minimal, but its total distance on the edges is at most twice that of the optimal Steiner tree [47]. The algorithm terminates in polynomial time [47].

A major difference of our method from other keyword search methods that generate Steiner trees is that we generate a Steiner tree based on an r-clique, which contains a very small subset of content nodes in the original graph G. The number of nodes in an r-clique is no more than the number of input keywords. Other tree-based keyword search methods need to explore at least all the content nodes in G or the entire graph to find a Steiner tree to cover the input keywords. Since our r-clique finding algorithm is also fast due to the fact that only the content nodes are explored during the search, the total time spent on finding r-cliques and then trees is much less than finding Steiner trees directly from G.

3.8 Neighbor Indexing Method

In the above algorithms, we need to compute the shortest distance between a pair of nodes. Calculating the shortest path on the fly is time-consuming and not necessary when the input graph is not frequently changed during the search process. An index that stores the shortest distance and path between nodes improves the performance of the algorithm. A straight forward indexing method is to calculate and store the shortest path between each pair of nodes. However, this index needs $O(n^2)$ storage, where n is the number of nodes in graph G. This index is very large and not feasible for graphs with a large number of nodes.

We use a simple and fast indexing method that pre-computes and stores the shortest distances for only the pairs of nodes whose shortest distance is within a certain threshold R. The index is called *neighbor index*. The value of R should be bigger than the value of r used in the r-clique finding algorithms. This requires the estimation of possible r values based on the graph structure and user preferences and may be estimated using the domain knowledge. At the same time, we should keep it as small as possible to keep the index in a feasible size. The idea of indexing the graph using a distance threshold has been used in [51, 60].

The *neighbor index* of a graph G with respect to the distance threshold R is structured as follows. For each node n, a list is created to contain the nodes that are within R distance from node n. This list is called the *neighbor list* of n. In each node m on the *neighbor list* of node n, the shortest distance between n and m is stored and also a pointer to the node right before m on the shortest path from n to m is stored. The pointed node p must be within R distance from n, is thus on n's neighbor list and contains a pointer to the node right before p on the shortest path between n and p. The space complexity of this index is $O(mn)$, where n is the number of nodes in G and m is the average number of nodes on a neighbor list. To build the index we use the Dijkstra's algorithm to compute the shortest path between each pair of nodes.

When finding r-cliques, the *neighbor index* is used to find the shortest distance between two nodes n and m by searching the neighbor list of n for node m. If the neighbor list contains node m, the stored shortest distance is returned. Otherwise, nodes n and m are not within R distance from each other. The shortest path between n and m (which is only used in the Steiner tree finding algorithm) can be found by following the pointer stored in the m node in n's neighbor list, which points to the node right before m on the shortest path.

3.9 Experimental Evaluation

We implement all of the algorithms proposed in this chapter. To evaluate the quality of the answers generated by our approximation algorithms, we use the branch and bound algorithm to find all of the r-cliques for an input graph. Exact answers can be obtained by ranking the r-cliques generated by the branch and bound algorithm. None of the other algorithms for keyword search in graphs produces r-cliques as their answers. Thus, comparing the results of our algorithms with those of other works is not a straight forward task. However, we implement a graph-based method that produces communities as answers [60] and compared it to our r-clique methods. The reason we choose this community-based method is that (1) it is among the most recent work in graph keyword search, (2) the definition of the community and its weight are more related to those of the r-clique than those in other approaches, and (3) it uses a polynomial delay algorithm, similar to our Algorithm 2, to produce top-k answers. To keep the comparison fair, both of the r-clique and community methods use the same graph indexing structures.

All of the algorithms are implemented using Java. The experiments are conducted on an Intel(R) Core(TM) i7 2.86GHz PC with 4GB of RAM. In this section, the results of the algorithms and the factors affecting the performance of the algorithms are discussed. The factors include the value of r, the number of keywords (l) and the frequency of keywords (i.e., the percentage of the nodes in the graph containing the keywords). Throughout this section, the branch and bound algorithm is called *B&B*. Our top-k polynomial delay algorithm that uses Algorithm 4 to produce r-cliques is called *r-clique*. Our top-k algorithm that uses Algorithm 5 (which produces r-cliques starting with only the nodes containing the rarest input keyword) is called *r-clique-rare*. In addition, the algorithm in [60] that produces top-k communities with polynomial delay is called *community*.

3.9.1 Data Sets and Queries in Experiments

We use three real data graphs in our experiments: DBLP, IMDb and Mondial. All the three graphs are used as an undirected and weighted graph. The original graphs do not have edge weights. The weights are added as follows. The weight of the edge between two nodes v and u is computed as $(\log_2(1 + v_{deg}) + \log_2(1 + u_{deg}))/2$, where v_{deg} and

Table 3.2: Keywords used in DBLP data set.

Frequency	Keywords
0.0003	distance, discovery, scalable, protocols
0.0006	graph, routing, space, scheme
0.0009	fuzzy, optimization, development, support, environment, database
0.0012	modeling, logic, dynamic, application
0.0015	control, web, parallel, algorithms

Table 3.3: Keywords used in IMDb data set.

Frequency	Keywords
0.0003	game, summer, bride, dream
0.0006	Friday, street, party, heaven
0.0009	girl, lost, blood, star, death, all
0.0012	city, world, blue, American
0.0015	king, house, night, story

u_{deg} are the degrees of nodes v and u respectively. Note that the same weighting function was used in [16, 32, 60].

The DBLP graph is produced from the DBLP XML data[6]. The dataset contains information about a collection of papers and their authors. It also contains the citation information among papers. The *papers* and *authors* are connected together using the *citation* and *authorship* relations. The numbers of tuples of the 4 relations *author*, *paper*, *authorship* and *citation* are 613K, 929K, 2,375K, and 82K respectively. The set of input keywords used in our experiments and their frequencies in the input graph are presented in Table 3.2. The queries used in our experiments are generated from this set of keywords with the constraint that in each query all keywords have the same frequency (in order to better observe the relationship between run time and keyword frequency). Note that the set of input keywords and the way to generate queries are the same as the ones in [60].

The IMDb dataset contains the relations between movies and the users of the IMDb website that rate the movies[7]. The number of tuples of 3 relations *user*, *movie* and *rating* are 6.04K, 3.88K and 1,000.21K, respectively. The set of input keywords and the frequencies are presented in Table 3.3. Note that the set of input keywords is the same as the one used in [60].

The Mondial dataset contains geographical data[8]. It is highly cyclic and contains

[6]http://dblp.uni-trier.de/xml/

[7]http://www.grouplens.org/node/73

[8]http://www.dbis.informatik.uni-goettingen.de/Mondial/

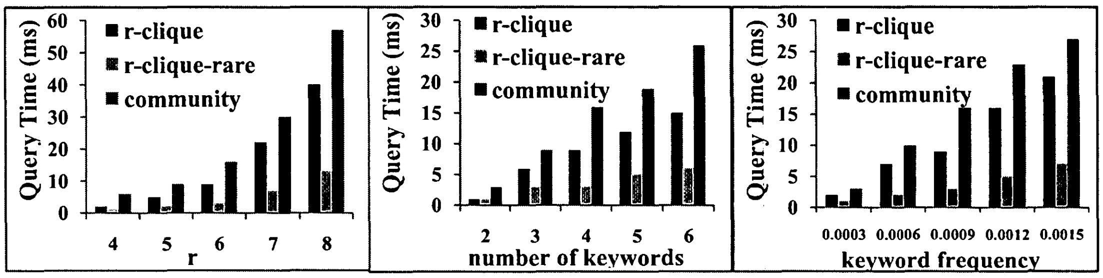


Figure 3.4: Run time on DBLP for algorithms that produce top-k answers with polynomial delay. When not changing, the number of keywords is set to 4, r is set to 6 and keyword frequency is 0.0009.

27 relations and 10K nodes. The query keywords used for this data set are randomly selected, which is the same way as in [22]. Since Mondial is very cyclic and contains many relations [22], r is set to a high value which is twice the value of r for the DBLP and IMDb datasets.

Among the three datasets, DBLP is the largest and contains more textual information. Thus, some of the results are only presented for the DBLP dataset.

3.9.2 Run Time Comparison

In this section, we compare *r-clique*, *r-clique-rare*, and *community* in terms of run time. The average time for producing one answer in finding top-50 answers is used as their run time. If there is no answer for the query, the time of completing the program is considered as the run time. For the r-clique methods, the time also includes the time for generating

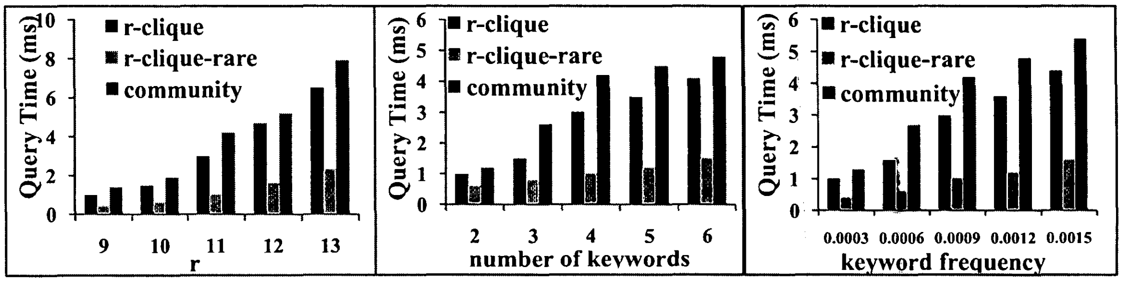


Figure 3.5: Run time on the IMDb data set of algorithms producing top-k answers with polynomial delay. When not changing, the number of keywords is 4, r is 11 and keyword frequency is 0.0009.

Steiner trees as final answers. The run time of the three algorithms for producing top-k answers on the DBLP, IMDb and Mondial datasets is shown in Figure 3.4, Figure 3.5 and Figure 3.6 respectively. For the DBLP and IMDb data sets, we depict the run time with respect to increasing values of r, the number of query keywords and keyword frequency. For the Mondial data set, the query keywords are completely randomly chosen from the whole data set regardless of their frequencies (to be the same as in [22]). Also, the same as in [22], for each number of query keywords ranging from 2 to 6 or each r values, four different random queries are evaluated and the average run time is taken.

We can see that *r-clique* and *r-clique-rare* produce results faster than *community*. The main reason for this is that the two r-clique methods explore only the content nodes during their search for answers. As expected, the run time of *r-clique-rare* is much smaller than *r-clique*. It is because *r-clique-rare* only uses the content nodes that contain

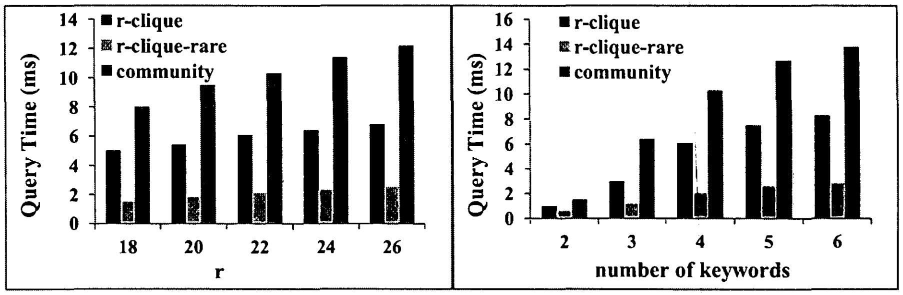


Figure 3.6: Run time on the Mondial data set of algorithms producing top-k answers with polynomial delay. When not changing, the number of keywords is 4 and r is 22.

the input keyword with the rarest frequency as the seed for a candidate answer. By increasing the value of r or the frequency of keywords, the run time of all the three algorithms increases. This is because there are more candidates and nodes to evaluate. These results agree with the findings in [60]. By increasing the number of keywords, the run time (i.e., the average run time for producing one answer) of all the three algorithms also increases. This means that average delay increases when the number of keywords increases. This is because more nodes need to be evaluated in each step when there are more keywords in the query. It should be mentioned that this result does not agree with the results presented in [60] for generating top-k communities.

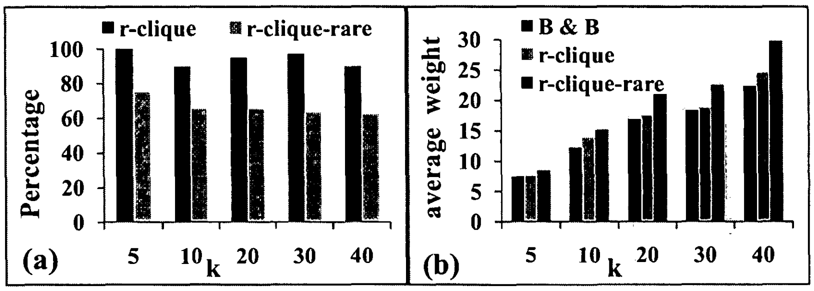


Figure 3.7: Quality of the *r-clique* and *r-clique-rare* algorithms on DBLP. The number of keywords is set to 4, r is 8 and the frequency of keywords is 0.0009.

3.9.3 The Quality of the Approximation Algorithm Compared with Exact Answers

In this section, the quality of the answers generated by the approximation algorithms is evaluated. We use the branch and bound algorithm (*B&B*) to produce all the r-cliques and compare them with the answers from the *r-clique* and *r-clique-rare* algorithms. Figure 3.7 (a) shows the percentage of answers produced by the approximation algorithms which are actually r-cliques. In other words, it shows the percentage of answers of the approximation algorithms whose distance between each pair of content nodes are less than or equal to r. The results show that at least 90% of the answers of Algorithm 4 and at least 62% of the answers of the Algorithm 5 are r-cliques. Figure 3.7 (b) shows the average weight of the answers produced by the *B&B*, *r-clique* and *r-clique-rare* algorithms for different k values. To get the top-k results for *B&B*, we rank the answers from *B&B*

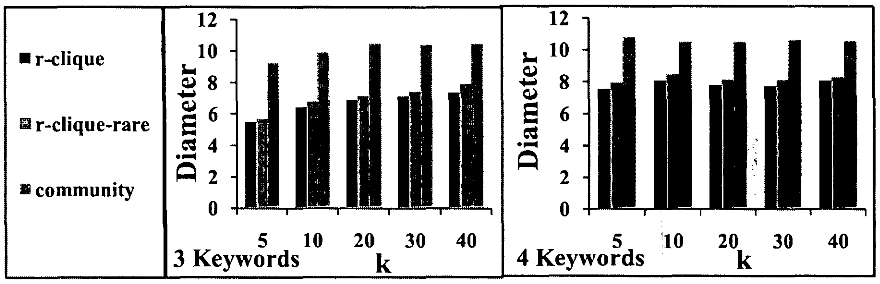


Figure 3.8: The average diameter of answers on DBLP. The frequency of keywords is 0.0009 and r is 8.

based on their weight. Figure 3.7 (b) shows that the weights of the answers produced by *r-clique* and *r-clique-rare* are close to those of exact answers produced by *B&B*. Although according to Theorem 3 the weight of an answer from the *r-clique* method can be twice that of the corresponding answer from *B&B*, our results show that the ratio of the weights is at most 1.12 in practice. Similarly, the ratio of the weights of the answers from *r-clique-rare* to those from *B&B* is at most 1.33 in this experiment, although it can be 3 (i.e., $l - 1$) according to Theorem 5. These results suggest the high quality of the proposed approximation algorithms.

3.9.4 Compactness of Answers

In this section, we evaluate the quality of the answers produced by *r-clique*, *r-clique-rare*, and *community*, in terms of their compactness. A well known measure for estimating the proximity of a subgraph is the *diameter* of the subgraph, defined as the largest shortest

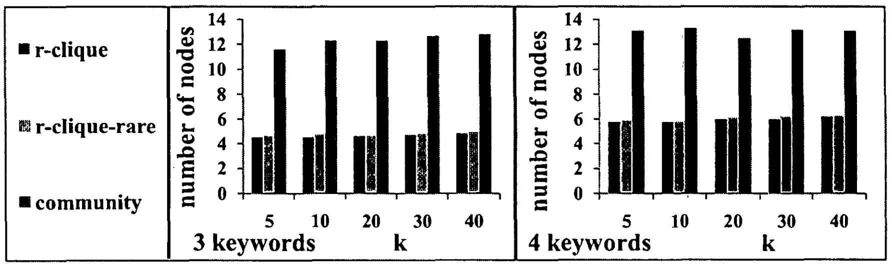


Figure 3.9: The average number of nodes in final answers of different algorithms on DBLP. The frequency of keywords is 0.0009 and r is 8.

distance between any two nodes in the subgraph. Generally, the smaller the diameter, the closer the nodes are to each other. When calculating the diameter for *r-clique* and *r-clique-rare*, we use all the nodes in the final answer, i.e., the nodes in the Steiner tree presented to the user. The average diameters of the answers of different algorithms are shown in Figure 3.8, which shows that the nodes in an answer produced by *r-clique* and *r-clique-rare* are closer to each other than those from *community* for different k values and different numbers of keywords. The average number of nodes in the answers produced by each algorithm is shown in Figure 3.9. Since a community includes all of the nodes whose distance to each content node is no larger than r, the number of nodes in a community is higher than that in our methods that use trees to present the final answers.

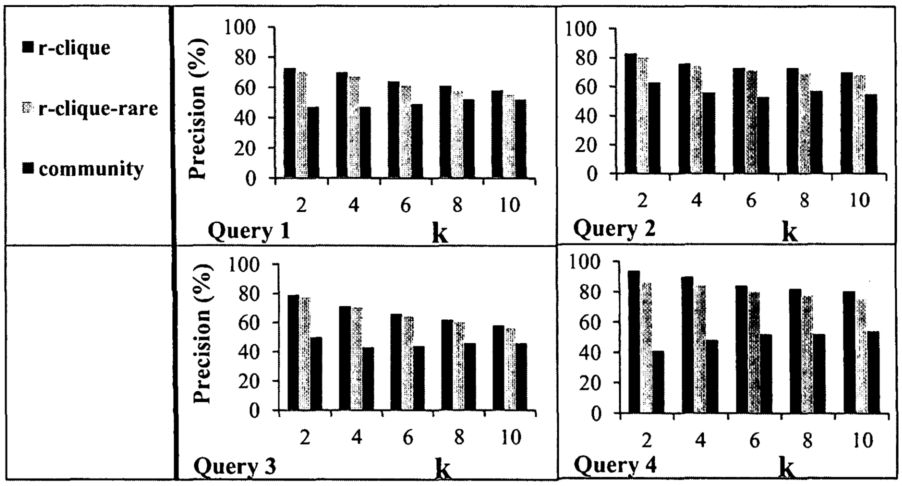


Figure 3.10: Top-k precision of answers with different values of k with the first method.

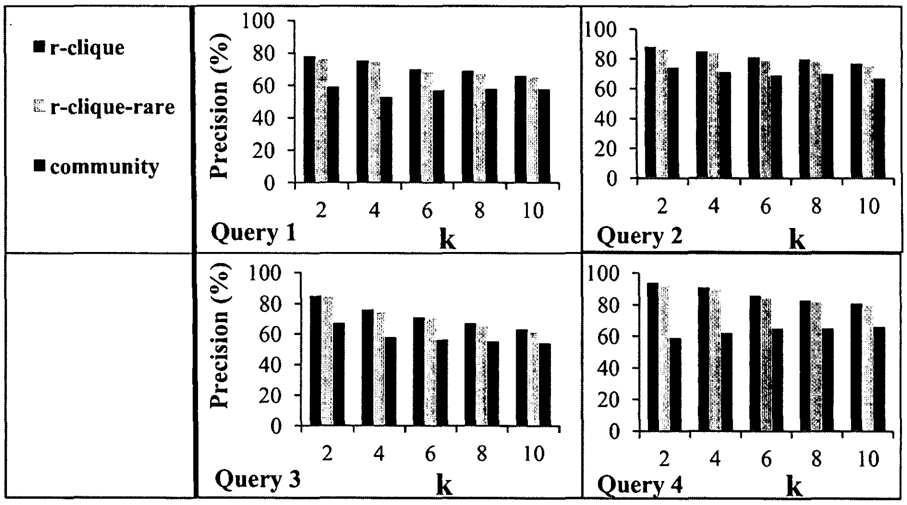


Figure 3.11: Top-k precision of answers with different values of k with the second method.

Table 3.4: Set of queries used for finding the accuracy of the results in DBLP data set.

Query	Keywords
1	parallel, graph, optimization, algorithm
2	dynamic, fuzzy, logic, algorithm
3	graph, optimization, modeling,
4	development, fuzzy, logic, control

3.9.5 Search Accuracy from a User Study

We further compare the *r-clique*, *r-clique-rare*, and *community*, algorithms in terms of how relevant their answers are to the query. Search results are best evaluated by measuring the relevancy of the results to the query. A common metric of relevance used in information retrieval is top-k precision, defined as the percentage of the answers in the top-k answers that are relevant to the query. To evaluate the top-k precision of the algorithms, we conducted a user study. We designed 4 meaningful queries from the lists of keywords in Table 3.2 for the DBLP dataset in order for human users to be able to evaluate the search results. The four queries are listed in Table 3.4. In the experiment, r is set to 8 and top-10 answers are produced for each query from each algorithm.

We asked 8 graduate students in computer science and electrical engineering at two universities to judge the relevancy of the answers. The users are asked to evaluate the

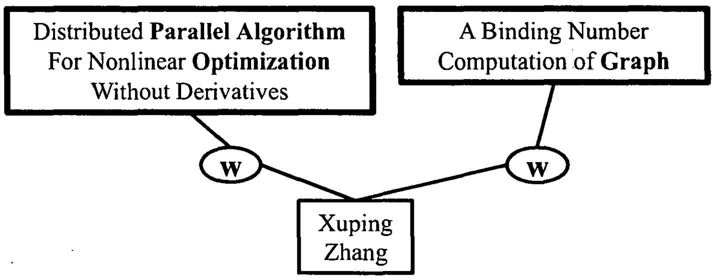

Distributed **Parallel Algorithm** For Nonlinear **Optimization** Without Derivatives
A Binding Number Computation of **Graph**
w
w
Xuping Zhang

Figure 3.12: Best r-clique answer to the query consisting of *parallel*, *graph*, *optimization* and *algorithm.*

answers using two methods. In the first method, for each answer, the user assigns a score between 0 and 1 to each paper (i.e., node) in the answer where 1 means completely relevant and 0 means completely irrelevant to the query. Then, the average score of the papers in an answer is calculated as the relevancy score of the answer. This score may vary among the users. We use the average of the relevancy scores from the 8 users as the final relevancy score of the answer. The top-k precision is computed as the sum of the relevancy scores of the top-k answers divided by k. In the second method, users assign a score between 0 and 1 to the whole answer based on the relevancy and understandability of the answer.

The top-2 to top-10 precisions for each query are presented in Figure 3.10 and Figure 3.11 for the first and second methods respectively. Clearly, *r-clique* and *r-clique-rare* achieve better precisions than *community* in all the queries for all the k values. The reason for the community method to have a lower precision is that a community may

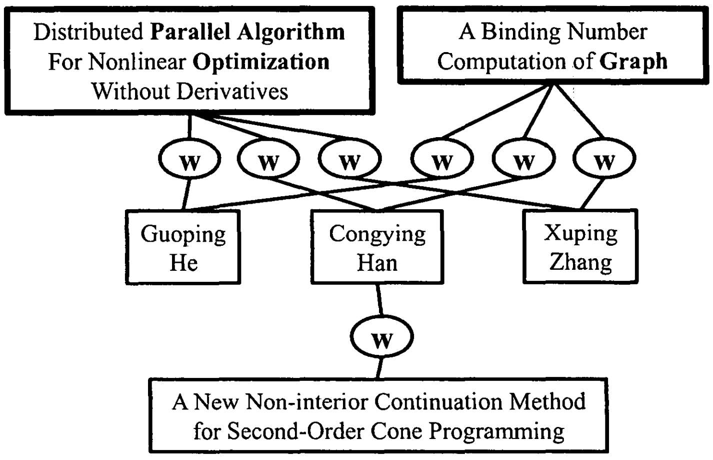

Distributed Parallel Algorithm For Nonlinear Optimization Without Derivatives
A Binding Number Computation of Graph
w
Guoping He
Congying Han
Xuping Zhang
A New Non-interior Continuation Method for Second-Order Cone Programming

Figure 3.13: Best community answer to the query consisting of *parallel*, *graph*, *optimization* and *algorithm*.

contain some center nodes and these centers are determined only based on their distance to the content nodes. If a node's distance to each of the content nodes in the answer is within a threshold, it is included in the community as a center. However, such a node may not be relevant to the query. By looking at the individual answers, we find that the community method indeed returns papers that are considered irrelevant to the query by the users.

3.9.6 Qualitative Evaluation

We compare the *r-clique* and *community* algorithms via an example. The top answer returned by *r-clique* for the first query in the user study is shown in Figure 3.12. The two boxes at the top are content nodes, each containing the title of a paper. The node at the bottom is the mediator node generated by our Steiner tree algorithm given the two content nodes. It is a common author of the two papers. The "W" symbol on an edge indicates the "writing" relationship. Clearly, our r-clique based method is able to reveal a relationship between the two content nodes. Figure 3.13 illustrates the top answer from the *community* algorithm. The top two nodes are content nodes, and the others are center nodes because each of them is within r distance from each of the content nodes. As can be seen, the community contains more nodes than the answer from the r-clique method. The three middle nodes are the three common authors of the two papers and the bottom node is another paper written by one of the authors, which is not relevant to the query. The advantage of this answer is that it reveals more common authors of the two papers (assuming this is useful for the user), but the disadvantage is that it also includes an irrelevant node. Having irrelevant nodes in an answer can make the answer hard to understand. Most of the users in our user study prefers the answer in Figure 3.12 over this one.

3.10 Conclusion and Future Work

We propose a novel and efficient method for keyword search on graph data. A problem with existing approaches is that, while some of the nodes in the answer are close to each other, others may be far from each other. To address this problem, we introduced the concept of r-cliques as the answer for keyword search in graphs. A benefit of finding r-cliques is that only content nodes need to be explored during the search process, which leads to significant runtime improvement. We propose a procedure that produces r-cliques in polynomial delay. Two approximation algorithms are proposed to find a single best answer in the search space. For evaluating the quality of the approximation algorithms, an exact algorithm for finding all r-cliques is proposed. Our experiments showed that the quality of the answers from the proposed approximation algorithms in comparison with the exact one is high in terms of the percentage of r-cliques and the average weight in the top ranked answers. To reveal the relationship between the nodes in an r-clique, we proposed to generate a Steiner tree based on the r-clique. Our experimental results showed that finding r-cliques is more efficient and produces more compact and more relevant answers than the method for finding communities [60].

As the future work, we plan to improve the approximation ratio of the proposed approximation algorithms or prove that the ratio is tight. Another interesting direction for future work is to improve the indexing method to efficiently update the index in case of updating the input graph. Building keyword search engines over graph data using *MapReduce* programming model could be another extension to this work. In this case, the search engine is able to handle graphs with billions of nodes/edges in a distributed environment.

4 Efficient Duplication Free and Minimal Keyword Search in Graphs

Keyword search over a graph searches for a subgraph that contains a set of query keywords. A problem with most of existing keyword search methods is that they may produce duplicate answers that contain the same set of content nodes (i.e., nodes containing a query keyword) although these nodes may be connected differently in different answers. Thus, users may be presented with many similar answers with trivial differences. In addition, some of the nodes in an answer may contain query keywords that are all covered by other nodes in the answer. Removing these nodes does not change the coverage of the answer but can make the answer more compact. These answers are more desirable in some applications. The answers in which each content node contains at least one unique query keyword are called *minimal answers* in this chapter. We define the problem of finding duplication-free and minimal answers, and propose algorithms for finding such answers efficiently. Extensive performance studies using two large real data sets confirm the efficiency and effectiveness of the proposed methods.

4.1 Introduction

Keyword search is a well known method for extracting relevant knowledge from a set of documents in information retrieval. Given a graph where nodes are associated with text, keyword search over the graph finds a subgraph that contains a set of query keywords. Due to the fact that many types of data can be represented by graphs, keyword search over graphs has received much attention in recent years. Most of the work in this area find minimal connected trees (e.g, Steiner trees with the minimum sum of edge weights [8, 16, 22, 24, 32]) or subgraphs that minimize a proximity function (e.g., the sum of distances from the nodes in the answer to a center node [60]). However, these methods may generate many trees or subgraphs with the same set of content nodes (i.e., nodes containing at least one query keyword) even though these answers may have different intermediate nodes connecting the content nodes.

The following example illustrates the duplication problem for a tree-based method. Suppose the nodes in an input graph are web pages. Two nodes are connected by an edge if there is a link from one page to the other. Consider Figure 4.1. The user is interested in finding pages that contain keywords k_1 and k_2. Two nodes m_{k1} and n_{k1} contain keyword k_1 and another two nodes m_{k2} and n_{k2} contain keyword k_2. The left graph in the figure contains 4 trees that cover m_{k1} and m_{k2}, where each branch from m_{k1} to m_{k2} is a tree. The right graph contains a single tree that covers n_{k1} and n_{k2}. Assume that the weight

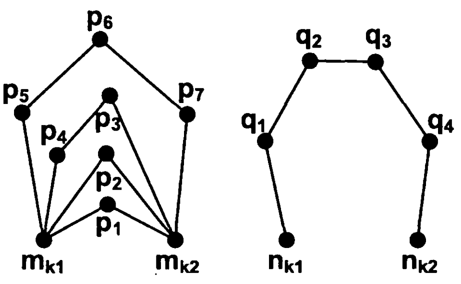


Figure 4.1: Duplication problems with tree answers.

on each edge is the same. According to the ranking function used in the tree approaches, the tree that contains n_{k1} and n_{k2} in the right graph is produced **after** the first four trees that cover m_{k1} and m_{k2} on the left, because it has more edges than the other four trees. However, all the four trees on the left have the same set of content nodes. Since the users usually want to see different groups of content nodes that are close to each other and might not be interested in browsing multiple relations to see how the nodes that contain input keywords are related to each other, the above search results might not be desirable[9]. Producing results with distinct sets of content nodes can prevent the search engine from overwhelming the user with many similar answers.

In addition to producing redundant results, current tree and graph-based methods may produce non-minimal answers. In other words, a content node in an answer may cover input keywords which are all covered by other content nodes. However, minimal answers may be preferred in some situations. Suppose that a customer wants to buy

[9]If a user wants to explore different relationships among the content nodes, the method in [43] can be used to produce a set of Steiner trees that connect a set of specified nodes together.

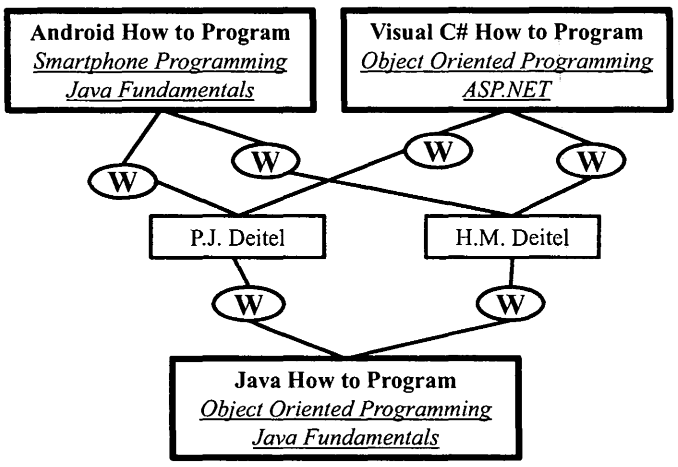


Figure 4.2: non-minimal answer for query: *Smartphone Programming*, *Java Fundamentals*, *Object Oriented Programming* and *ASP.NET* over a graph connecting books via authors.

a set of items from stores and wants to find a set of stores that together have all the items he/she wants to buy. Assume that the information about the stores is stored in a graph, where a node represents a store and contains the list of items that the store sells, and an edge between two nodes is weighted by the distance between the two stores. The customer issues a query specifying the set of items he/she wants to buy. It would be better that the search result is a list of stores in which each store has at least one unique item in the query that other stores do not have because there is no need to go to a store that does not have a unique item in the query. Another example is to determine required textbooks that together cover all the topics in a course. Assume that an online bookstore (e.g., *Amazon.com*) maintains its product information in an underlying graph where a node represents a book and contains the topics the book covers, and two books are connected by an edge if they share an author. Assume that the topics for a course are *Smartphone Programming*, *Java Fundamentals*, *Object Oriented Programming* and *ASP.NET*. A search over the graph allows us to find a set of books that not only covers all the topics but may also share the same author(s), which is preferred because the writing style of the books may be consistent. A possible answer to this query is shown in Figure 4.2, where the three books share the same authors and together cover all the topics. But the topics covered by "Java How to Program" are also covered by the two other books. Thus, from the money-saving prospective it is not necessary to require the students to buy this book. In this type of applications, minimal answers are desired.

In this chapter, we first propose a new approach to keyword search in graphs that produces duplication-free answers. Each answer produced by our approach has a unique set of content nodes. We also define *minimal answers*, in which each node contains at least one input keyword that other nodes do not. We propose two algorithms that convert an answer to a minimal answer. We prove that the problem of finding a minimal answer while minimizing the proximity function that we use is NP-complete. Thus, one of the algorithms we propose is a greedy algorithm that searches for a sub-optimal minimal answer. We prove that this greedy algorithm has a bounded approximation ratio. Finally, for finding top-k duplication-free and minimal answers, we propose two approaches. The first approach is faster but may miss some answers. The second approach takes more time in theory but can produce all the answers if needed. Our extensive experiments show the efficiency and effectiveness of the proposed methods.

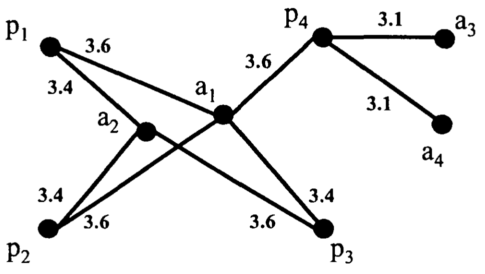
The chapter is organized as follows. In the next section, we motivate our problem using a real example which shows previous methods produce duplicate and non-minimal answers. In Section 4.3, we give formal problem statements. In Section 4.4, a procedure for finding duplication free answers in polynomial delay is presented. An algorithm for finding the best answer in each search space is given in Section 4.5. Finding minimal answers is discussed in Section 4.6. Other issues including graph indexing and presenting the answers are discussed in Section 4.7. Experimental results are given in Section 4.8. Section 4.9 concludes this chapter.

4.2 Motivation

In this section, a real example is presented to show that previous works produce duplicate and non-minimal answers. We further show that distinct root tree approach is not complete.

4.2.1 Producing Duplicate and Non-Minimal Answers by Previous Works

The following example shows the existing graph keyword search methods generate duplicate and non-minimal answers. Consider a small part of the DBLP dataset, which contains four authors and four papers. The paper titles, author names and a weighted graph that connect the authors and papers are shown in Figure 4.3. The edge weights are computed in the same way as in [35, 60]. Assume that the input keywords are k_1:*dynamic*, k_2:*fuzzy*, k_3:*logic*, k_4:*design* and k_5:*optimization*. Among all the subsets of the nodes, only $\{p_2, p_4\}$ covers all the input keywords and is also minimal. Other subsets either do not cover all the input keywords or are not minimal. The top-5 answers of the dynamic programming algorithm in [16] for finding Steiner trees are given in Table 4.1, which shows that all the answers contain the same set of content nodes, although they have different roots connecting the content nodes. The top-5 answers of the BLINKS algorithm [24] are shown in Table 4.2, which shows that the sets of content nodes of the last three answers are exactly the same. In addition, none of the five answers is minimal. The top-5 answers of the community-finding method [60] are shown in Table 4.3. The second column of the table presents the association of each keyword with a node in the answer and the third column shows the set of content nodes. Some of these top-5 answers are duplicated and some of them are not minimal. The r-clique method [35] uses the same method for dividing the search space into sub-spaces when finding top-k answers as the community-finding method. Thus, it has a similar problem regarding the search for duplication free and minimal answers.



PID	Title	AID	Name
p_1	A Framework for Studying the Effects of **Dynamic** Crossover, Mutation, and Population Sizing in Genetic Algorithms	a_1	Michael A. Lee
p_2	**Dynamic** Control of Genetic Algorithms Using **Fuzzy Logic** Techniques	a_2	Hideyuki Takagi
p_3	Neural Networks and Genetic Algorithm Approaches to Auto **Design** of **Fuzzy** Systems	a_3	Henrik Esbensen
p_4	The **Design** of Hybrid Fuzzy Evolutionary Multiobjevtive **Optimization** Algorithms	a_4	Laurent Lemaitre

Figure 4.3: A sample graph from the DBLP dataset.

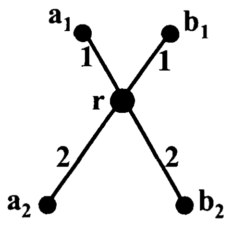


Figure 4.4: An example for showing the incompleteness of distinct root trees approach (e.g. BLINKS [24]). a_1 and a_2 contain keyword k_1 and b_1 and b_2 contain keyword k_2. Using distinct root semantics, the answer which has a_2 and b_2 as the content nodes is never produced.

Table 4.1: Steiner trees generated by dynamic programming.

No.	Root	Leaf Nodes (Content Nodes)
1	p_4	p_2, p_4
2	p_2	p_2, p_4
3	a_1	p_2, p_4
4	a_3	p_2, p_4
5	a_4	p_2, p_4

4.2.2 Incompleteness of Distinct Root Tree Approach

To show that distinct root tree approach is not complete and might miss some combination of content nodes, an example is presented in Figure 4.4. a_1 and a_2 contain keyword k_1 and b_1 and b_2 contain keyword k_2. The graph has the following five nodes: $\{r, a_1, a_2, b_1, b_2\}$. Thus, the number of answers is at most five. Five answers with the associated root, the set of content nodes and their weights are presented in Table 4.4. The tree which has $\{a_2, b_2\}$ as the content nodes is not produced. The reason is that any tree that has $\{a_2, b_2\}$ as the content nodes has the weight of at least 4. Thus, this combination of content nodes is never produced using the distinct root semantics.

Table 4.2: Distinct root trees generated by BLINKS.

No.	Root	Leaf Nodes (Content Nodes)
1	p_2	p_2, p_3, p_4
2	p_4	p_1, p_2, p_4
3	a_1	p_1, p_2, p_3, p_4
4	p_3	p_1, p_2, p_3, p_4
5	a_2	p_1, p_2, p_3, p_4

4.3 Problem Statement

Given a data graph whose nodes are associated with text and a query consisting of a set of keywords, the problem of keyword search in a graph is generally to find a subgraph that contains all or part of the keywords. The data graph can be directed or undirected. The edges and/or nodes may have weights on them. In this work, the same as [16, 35, 51] and Chapter 3, we consider undirected graphs with weighted edges, where two nodes are connected by an edge if there is a relationship between them and the edge weight represents the distance between the two nodes. Undirected graphs can be used to model different types of unstructured, semi-structured and structured data, such as web pages, XML documents and relational datasets. It should be noted that our approach is adaptable

Table 4.3: Answers from the community-finding method.

No.	Keyword-Node Association	Content Nodes
1	$(k_1, p_1), (k_2, p_4), (k_3, p_2), (k_4, p_4), (k_5, p_4)$	p_1, p_2, p_4
2	$(k_1, p_2), (k_2, p_2), (k_3, p_2), (k_4, p_4), (k_5, p_4)$	p_2, p_4
3	$(k_1, p_2), (k_2, p_4), (k_3, p_2), (k_4, p_4), (k_5, p_4)$	p_2, p_4
4	$(k_1, p_1), (k_2, p_2), (k_3, p_2), (k_4, p_4), (k_5, p_4)$	p_1, p_2, p_4
5	$(k_1, p_2), (k_2, p_2), (k_3, p_2), (k_4, p_3), (k_5, p_4)$	p_2, p_3, p_4

to work with directed graphs[10].

Definition 3 *(Answer) Given a graph G and a set of query keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), an* Answer *to Q in G is a set of content nodes in G that together cover all of the input keywords in Q.*

An *Answer* has a *weight* which can be defined according to the application need based on the weights of the edges in G that connect the nodes in the *Answer*. The above definition does not require the nodes in an *Answer* to be connected with each other either directly or indirectly in G, but *Answers* with nodes connected to each other can be preferred over those with disconnected nodes by using a weight function.

[10]Algorithms 7, 9, 11 and 12 proposed in this chapter are independent of graph type. But the weight function and its related procedures (Algorithms 8 and 10) need to be adapted to work with directed graphs. For example, the weight of an answer can be defined using the weights of edges in both directions.

Table 4.4: Five answers of distinct root tree approach for the graph of Figure 4.4

No.	Root Node	Leave Nodes (Content Nodes)	Weight
1	r	$\{a_1, b_1\}$	2
2	a_1	$\{a_1, b_1\}$	2
3	b_1	$\{a_1, b_1\}$	2
4	a_2	$\{a_2, b_1\}$	3
5	b_2	$\{a_1, b_2\}$	3

Problem 3 ***(Duplication free keyword search)*** *Given a graph G, an integer k and a set Q of query keywords, find top-k* **unique** Answers *of Q in G whose weights are optimal.*

An *Answer* is *unique* if it appears at most once in the top-k list. The next definition deals with the minimality of the *Answer*.

Definition 4 ***(minAnswer)*** *Given a graph G and a set of query keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), a* minAnswer *of Q in G is an* Answer *of Q in G in which each content node covers at least one query keyword that other content nodes do not cover.*

In other words, each content node of a *minAnswer* uniquely contributes to cover at least one query keyword.

Problem 4 ***(Duplication free and minimal keyword search)*** *Given a graph G, an integer k and a set Q of input keywords, find top-k* **unique minAnswers** *of Q in G whose* *weights are optimal.*

To focus on the generality of the above keyword search problems, we intentionally avoided defining the weight of an *Answer* in Definitions 3 and 4. Below we give two definitions of weight functions, used in [24, 35, 60] and Chapter 3, to measure the *proximity* of the nodes in an *Answer*. Note that other weight functions can be used with our definitions. Also, most of the algorithms proposed in this chapter (i.e., Algorithms 7, 9, 11 and 12) are independent of the weight function. Only Algorithms 8 and 10 depend on the weight function.

Definition 5 ***(sumDistance)*** *Suppose that the set of nodes in an $Answer$ in graph G is denoted as* $V = \{v_1, v_2, \ldots, v_l\}$*. The* sumDistance *of the $Answer$ is defined as*

$$sumDistance = \sum_{i=1}^{l} \sum_{j=i+1}^{l} dist(v_i, v_j)$$

where $dist(v_i, v_j)$ is the shortest distance between v_i and v_j in G, i.e., the sum of weights on the shortest path between v_i and v_j in G (See Chapter 3 and [35]).

Definition 6 ***(centerDistance)*** *Suppose that the set of nodes in an $Answer$ in graph G is denoted as* $V = \{v_1, v_2, \ldots, v_l\}$*. The* centerDistance *of the $Answer$ is defined as*

$$centerDistance = \min_{c \in G} \sum_{i=1}^{l} dist(c, v_i)$$

where $dist(c, v_i)$ is the shortest distance between a node c in G and v_i. The node in G that achieves the minimum distance is called the center *of the $Answer$.*

Table 4.5: An overview of Algorithms 8-12(N/A means not applicable).

Alg.	Dup. Free	Minimal	Minimize $sumDistance$	Approx. Ratio	Complete
Alg. 8	Yes	No	Yes, bounded approx.	2	Yes
Alg. 9	No with Alg. 8	Yes	No	N/A	N/A
Alg. 10	No with Alg. 8	Yes	Yes, bounded approx.	$(\log n)\frac{d_{max}}{d_{min}}$	N/A
Alg. 11	Yes	Yes	N/A in general	N/A	No
Alg. 12	Yes	Yes	N/A in general	N/A	Yes

The *centerDistance* is used in [24, 60]. In [24], the *center* is the *root* of the answer tree. Note that the *center* may/may not be a node in the $Answer$.

When using $sumDistance$ or $centerDistance$ to define the weight of an *Answer*, $Answers$ with smaller weights are considered to be better because the nodes in an *Answer* are closer to each other when its weight is smaller.

4.3.1 An Overview of the Proposed Algorithms

In this chapter, we propose six algorithms to solve Problems 3 and 4. Algorithm 7 is a general framework for generating top-k duplication-free answers by wisely dividing the search space. It calls Algorithm 8 (which is a 2-approximation algorithm for finding a single answer that minimizes $sumDistance$) to find top-k duplication-free answers in polynomial delay. Thus, Algorithms 7 and 8 together solve Problem 3.

Table 4.6: Run Time Complexity of Algorithms 8-12.

Alg.	Complexity Type	Run Time Complexity
Alg. 8	Polynomial	$O(l^2 \times \|D_{max}\|^2)$
Alg. 9	Polynomial	$O(n^2)$
Alg. 10	Polynomial	$O(n^2)$
Alg. 11	Polynomial	$O(l^2 \times \|D_{max}\|^2)$
Alg. 12	FPT	$O((\prod_{i=1}^{s} \|K_i\|) \times l^2 \times \|D_{max}\|^2)$

Note: l is the number of query keywords, $n(\leq l)$ is the number of nodes in the input answer for Algorithm 10, d_{max} and d_{min} are the max and min distances between any pair of nodes in the input answer for Algorithm 10, $s < l$, $\sum_{i=1}^{s} |K_i| < l$, $D_{max} \ll$ the number of nodes in graph.

To generate *minAnswers*, Algorithms 9 and 10 are proposed to convert the answers generated by Algorithm 8 into a *minAnswer*. Algorithm 9 does not optimize a weight function, while Algorithm 10 finds a *minAnswer* that also minimizes the $sumDistance$ function with a bounded approximation ratio. However, simply converting Algorithm 8's answer to a *minAnswer* with Algorithm 9 or 10 may lead to generation of duplicate answers in the top-k procedure.

To generate top-k *duplication-free minAnswers* (i.e., to solve Problem 4), Algorithms 11 and 12 are proposed to replace Algorithm 8 in Algorithm 7. Both algorithms are general frameworks for confining or dividing the search space to ensure minimality and no duplication in the top-k answers generated by Algorithm 7. They call a modified version of Algorithm 8 (which calls Algorithm 10) to generate a *minAnswer* that also minimizes $sumDistance$. The difference between Algorithms 11 and 12 is that Algorithm 11 is faster (completely polynomial) but may miss some answers, while Algorithm 12 is complete (i.e., it allows all the possible answers to be considered), but is a fixed-parameter tractable (FPT) algorithm. An overview of Algorithms 8-12 is given in Table 4.5. Run time complexity of the algorithms are summarized in Table 4.6

4.4 Finding Top-k Duplication Free Answers in Polynomial Delay

As stated in Section 3.4, an efficient search engine should satisfy three properties [22]. First, it should be able to generate all answers without missing them. Second, the answers should be presented in an order with better answers ranked higher. Third, the search engine should produce the answers efficiently. Assume that the maximum number of nodes containing a query keyword in the input graph is m. Based on the definition of $Answer$, the total number of $Answers$ might be up to m^l, where l is the number of query keywords. Apparently, producing all of the $Answers$ may overwhelm the user since m and/or l can be large. Thus, it is important to produce top-k $Answers$ (or all the answers if fewer than k answers exist) in a ranked order. The efficiency of a search engine is commonly measured based on the delay between producing two consecutive answers. If this delay is polynomial based on the input data, the algorithm is called a polynomial delay algorithm [22, 31].

The same as Section 3.4, our algorithm for producing top-k duplication-free answers is an adaption of Lawler's procedure [49] for finding top-k answers to discrete optimization problems. However, Algorithm 2 in Section 3.4 might produce duplicate answers. Lawler generalized Yen's algorithm in [69] which finds the k shortest loopless paths in a graph. In Lawler's procedure, the search space is divided into disjoined sub-spaces. The best answer in each subspace is found and used to produce the current best global answer. The sub-space that produces the best global answer is further divided into sub-subspaces and the best answer among its sub-subspaces is used to compete with the best answers in other sub-spaces in the previous level to find the next best global answer. Two main issues in this procedure are how to divide a space into subspaces and how to find the best answer within a (sub)space. To have duplication free answers, the procedure for dividing the search space into sub-spaces must produce **disjoint** sub-spaces so that the same answer cannot be generated from different sub-spaces.

Lawler's procedure has been used to generate top-k answers in graph keyword search in [35, 60] and Section 3.4, in which a search (sub)space is represented by $C_1 \times C_2 \times \cdots \times C_l$, where C_i is the set of nodes containing query keyword k_i, and the space is divided by taking away certain node(s) from C_i to form a subspace based on the best answer in the space being divided. A problem with this strategy is that a node taken away from C_i may appear in C_j (where $i \neq j$) if the node contains more than one query keyword (i.e., it belongs to more than one C_i for $1 \leq i \leq l$), and thus the same set of content nodes may be generated from different subspaces if a node contains more than one query keyword, although different answers have different keyword-node associations. Since we aim at generating unique sets of content nodes, a different strategy for dividing a search space is needed to avoid duplicate answers.

We first illustrate our idea of dividing the search space into disjoint subsets using an example. Given a set of input keywords, we first use the *FindBestAnswer* procedure (to be described later) to find the best answer $\{a, b, c\}$ in the input graph G, where a, b, and c are nodes in G. Then we divide the set of remaining answers to be found into three subsets: (1) the answers that contain a and b but no c, (2) the ones that contain a but no b, and (3) the ones that contain no a. Clearly, (1), (2) and (3) are disjoined, and they, together with $\{a, b, c\}$, comprise the set of all possible answers. Each subset has **constraints**, which can be represented using an **inclusion set** containing the nodes that must be included and an **exclusion set** containing the nodes that must be excluded[11]. Table 4.7 shows the constraints of these three subsets.

After dividing the search space into disjoint subsets based on the global best answer, the best *Answer* in each subspace is found using the *FindBestAnswer* procedure. These best *Answers* are inserted into a priority queue, where the *Answers* are

[11]The idea of using the inclusion and exclusion sets to represent constraints is inspired by [45]. However, the constraints in [45] are described using *edges* (instead of *nodes* as in our approach) for finding a different type of answers.

Table 4.7: Dividing the search space into disjoint subspaces based on the best $Answer$ $\{a, b, c\}$.

Subspace	Inclusion set	Exclusion set
SB_1	$Inc_1 = \{a, b\}$	$Exc_1 = \{c\}$
SB_2	$Inc_2 = \{a\}$	$Exc_2 = \{b\}$
SB_3	$Inc_3 = \{\emptyset\}$	$Exc_3 = \{a\}$

ranked in ascending order of their weights. Obviously, the second best $Answer$ is the one at the top of the priority queue. Suppose that this $Answer$ is taken from SB_2 and contains p content nodes. After returning the second best answer, SB_2 is divided into p subspaces in the way similar to the one shown in Table 4.7. In each subspace, the best $Answer$ is found and is added to the priority queue. At this state, the priority queue has $2 + p$ elements: two elements from the first step and p elements from this new step[12]. Then, the top $Answer$ is returned and removed from the queue, its corresponding space is divided into subspaces and the best $Answer$ (if any) in each new subspace is added to the priority queue. This procedure continues until the priority queue becomes empty or top-k $Answers$ are found.

The pseudocode of this procedure for enumerating top-k $Answers$ is described in Algorithm 7. The algorithm first computes the set C of nodes that contain at least one

[12]This assumes that all of the subspaces contain at least one $Answer$. In some cases, the subspace does not have any $Answer$.

input keyword. This can be easily done using a pre-built inverted index. In line 5, procedure $FindBestAnswer$ (to be described in the next section) is called to find the best answer from the whole search space (i.e., C). It takes input graph G, query Q, C, an inclusion set and an exclusion set as input, and returns the best answer in the search space specified by C. Since the first best answer is found in the whole search space, empty inclusion and exclusion sets are passed to the procedure in line 5. If the best answer exists (i.e., $A \neq$ NULL), A, together with the inclusion and exclusion sets (the constraints for the space from which A is generated), are inserted into $Queue$ in line 7. The $Queue$ is maintained in the way that its elements are ordered in ascending order of their weights. The while loop starting at line 8 is executed until the $Queue$ becomes empty or k answers have been outputted. In line 9, the top of the $Queue$ is removed, which contains the best answer (A) in the $Queue$ and its inclusion (Inc) and exclusion (Exc) sets. The answer in A is outputted. Then, if the number of answers has not reached k, the nodes in A are assigned to $n_1, n_2 \ldots n_p$ where p is the number of nodes in A. In lines 15-21, p new inclusion and exclusion sets are produced based on the nodes in A and the inclusion and exclusion sets for the space A was generated from. The new subspaces are specified by these new constraints. For each new subspace, if the intersection of its inclusion and exclusion sets is empty, the best answer is found and it is inserted into the $Queue$ with the constraints of its related subspace. Clearly, if procedure $FindBestAnswer$ runs in polynomial time, Algorithm 7 produces answers with polynomial delay.

Since for each best answer A the union of the sub-spaces created based on A plus answer A itself is the same as the search space from which A is found, no answer is excluded from search spaces in the next iterations. Thus, Algorithm 7 produces top-k or all answers (if fewer than k answers exist) if $FindBestAnswer$ finds the best answer in a search (sub)-space. In addition, the sub-spaces produced based on answer A are all disjoint and none of them contains A. Therefore, they do not lead to the same answer and the set of produced answers is duplication free. In addition, this duplication free search procedure is independent of the procedure for finding the best answer and the weight function used to measure the quality of an answer.

4.5 Finding the Best Answer in Each Search Space

Algorithm 7 calls the $FindBestAnswer$ procedure to find the best answer in a search space specified by a set of content nodes and the constraints (i.e., the inclusion and exclusion sets). The best answer must contain the nodes in the inclusion set, exclude the nodes in the exclusion set and also have an optimal weight. Depending on the weight function used, $FindBestAnswer$ can be designed differently. Below, we present an algorithm that produces an answer satisfying the constraints and minimizing the $sumDistance$ function. We present a modification of this algorithm which minimizes the $centerDistance$ later in this section.

Algorithm 7 Generate Duplication Free Top-k $Answers$

Input: the input graph G; the query $Q = \{k_1, k_2, \ldots, k_l\}$; k

Output: the set of top-k ordered $Answers$ printed with polynomial delay

```
1: $C \leftarrow$ an empty set for storing content nodes
2: for $i \leftarrow 1$ to $l$ do
3:     add the nodes in $G$ containing $k_i$ to $C$
4: $Queue \leftarrow$ an empty priority queue
5: $A \leftarrow$ FindBestAnswer($G$, $Q$, $C$, $\emptyset$, $\emptyset$)
6: if $A \neq$ NULL then
7:     insert $\langle A, \emptyset, \emptyset \rangle$ into $Queue$
8: while $Queue \neq \emptyset$ do
9:     $\langle A, Inc, Exc \rangle \leftarrow$ top element of $Queue$
10:    print($A$)
11:    $k \leftarrow k - 1$
12:    if $k = 0$ then
13:        return
14:    $\{n_1, n_2, \ldots, n_p\} \leftarrow$ content nodes of $A$
15:    for $i \leftarrow 1$ to $p$ do
16:        $Inc_i \leftarrow Inc \cup \{n_1, \ldots, n_{p-i}\}$
17:        $Exc_i \leftarrow Exc \cup \{n_{p-i+1}\}$
18:        if $Inc_i \cap Exc_i = \emptyset$ then
19:            $A_i \leftarrow$ FindBestAnswer($G$, $Q$, $C$, $Inc_i$, $Exc_i$)
20:            if $A_i \neq$ NULL then
21:                insert $\langle A_i, Inc_i, Exc_i \rangle$ into the right place of $Queue$ according to $A_i$'s
                   weight
```

Algorithm 8 FindBestAnswer minimizing the *sumDistance* function

Input: the input graph G; the query Q; the set of content nodes C; the set of inclusion nodes Inc; the set of exclusion nodes Exc

Output: the best (approximate) $Answer$ satisfying both Inc and Exc constraints

```
1: $Cov \leftarrow$ set of keywords covered by $Inc$
2: $\{k_1, k_2, \ldots, k_t\} \leftarrow \{Q - Cov\}$
3: for $i \leftarrow 1$ to $t$ do
4:     $D_i \leftarrow$ nodes of $C$ having keyword $k_i$ and $\notin Exc$
5: $D \leftarrow \bigcup_{i=1}^{t} D_i$
6: $F \leftarrow Inc \cup D$
7: if $F = \emptyset$ then
8:     return NULL
9: $leastWeight \leftarrow \infty$
10: $bestAnswer \leftarrow$ NULL
11: for each node $f_i$ in $F$ do
12:     $weight \leftarrow 0$
13:     $answer \leftarrow \emptyset$
14:     for each node $n_j$ in $Inc$ do
15:         $weight \leftarrow weight + d(f_i, n_j)$
16:         $answer = answer \cup \{n_j\}$
17:     for $j \leftarrow 1$ to $t$ do
18:         $dist \leftarrow \infty$
19:         $nearest \leftarrow$ NULL
20:         for each node $d_k$ in $D_j$ do
21:             if $d(f_i, d_k) < dist$ then
22:                 $dist = d(f_i, d_k)$
23:                 $nearest = d_k$
24:         if $nearest \notin answer$ then
25:             $weight \leftarrow weight + dist$
26:             $answer = answer \cup \{nearest\}$
27:     if $weight < leastWeight$ then
28:         $leastWeight \leftarrow weight$
29:         $bestAnswer \leftarrow answer$
30: return $bestAnswer$
```

4.5.1 Minimizing the $sumDistance$ function

In Section 3.2 we proved that minimizing $sumDistance$ is an NP-complete problem, with respect to the number of query keywords, and proposed an approximation algorithm that finds an answer with an approximation ratio of 2. The search space in that algorithm is a Cartesian product $C_1 \times C_2 \times \cdots \times C_l$, where C_i is a subset of nodes containing keyword k_i and excluding certain nodes. However, a node excluded from C_i may appear in C_j if the node contains both k_i and k_j. Since our answers must completely exclude the nodes specified by the exclusion set, we modify the algorithm in Section 3.5 to consider the constraints specified by the inclusion and exclusion sets.

The pseudo-code of the modified algorithm, $FindBestAnswer$, is presented in Algorithm 8. It takes an input graph G, a query Q, a set of content nodes C, and the inclusion and exclusion sets (Inc and Exc) as input and produces the best (approximate) $Answer$ as output in polynomial time. The algorithm approximates the $sumDistance$ of an answer using the sum of distances from each node in the answer to a center node within the answer. In the pseudo-code, set F is the search space, which consists of all the nodes in the inclusion set and the set of content nodes containing the query keywords not covered by the inclusion set and not belonging to the exclusion set. In the code, D_i is the set of nodes that contain keyword k_i (which is not covered by the inclusion set) but do not belong to the exclusion set. For each node f_i in F, an answer is formed by using

f_i as the center and including all the nodes in the inclusion set and adding the node in each D_i that is closest to f_i. The final answer is the one with the least sum of distances between each node in the answer and its center. In the code, $d(x, y)$ is the shortest distance between nodes x and y, which can be efficiently obtained by consulting a pre-built index which is described in Section 3.8)[13].

Clearly, the answer produced by this algorithm satisfies the inclusion and exclusion constraints. Since all the nodes in F have been considered as a center candidate, it can be proved that the *sumDistance* of the produced answer is no more than $\frac{2\times(l-1)}{l}\times$ the *sumDistance* of an optimal answer, where l is the number of query keywords. Thus, the produced answer has a weight that is at most twice that of an optimal answer. The proof is similar to the one in Section 3.5 and we omit it here. The complexity of this algorithm is $O(|F| \times l \times |D_{max}|)$ where $|F|$ is the size of the set F, l is the number of query keywords and $|D_{max}|$ is the maximum size of D_i for $1 \leq i \leq t$. Since $|F| \leq (l \times |D_{max}|) + |Inc|$ and $|Inc| \leq l - 1$, $|F| = O(l \times |D_{max}|)$. Thus, the complexity of Algorithm 8 is $O(l^2 \times |D_{max}|^2)$.

4.5.2 Minimizing the *centerDistance* function

Authors of [24, 60] proposed algorithms to minimize the *centerDistance* function. Here, we briefly describe how to modify Algorithm 8 to work with the *centerDistance*

[13]Using a pre-built index to obtain the shortest distance between nodes has also been used in [35, 51, 60].

function. In line 11 of Algorithm 8, all of the content nodes of the uncovered keywords and the inclusion set are checked for finding the best approximate $Answer$ that minimizes $sumDistance$. However, for minimizing $centerDistance$, each node in the input graph G should be considered as a center of a possible answer. Therefore, instead of browsing only the nodes in set F in line 11, the loop iterates through all of the nodes in graph G and checks each of them for finding the best center and its associated answer. Thus, the time complexity of the algorithm becomes $O(N \times l \times |D_{max}|)$, where N is the number of nodes in graph G. Since $N > F$, the revised Algorithm 8 for minimizing $centerDistance$ is slower than Algorithm 8 for minimizing $sumDistance$. Note that the same as [24, 60], the modified Algorithm 8 for minimizing $centerDistance$ returns the exact answer in polynomial time.

4.6 Finding Minimal Answers

Some of the $Answers$ returned by Algorithm 8 and existing algorithms may not be a $minAnswer$. That is, the input keywords in some nodes of an $Answer$ may all be covered by other nodes in the answer. If these nodes are removed from the answer, the remaining set of nodes still covers all the input keywords. Below we first present two algorithms for converting an $Answer$ to a $minAnswer$. However, the converted $minAnswer$ may violate the inclusion constraint for finding duplication-free answers. We then propose two approaches to solve the problem.

4.6.1 Generating Minimal Answers

Algorithm 9 ConvertToMinAnswerGeneral - General Procedure

Input: the set of content nodes as $Answer$; the query Q

Output: a $minAnswer$

```
1: for each node $n_i$ in $Answer$ do
2:     $K = \emptyset$
3:     for $j \leftarrow 1$ to $i-1$ do
4:         $K = K \cup$ keywords$(n_j)$
5:     for $j \leftarrow i+1$ to size($Answer$) do
6:         $K = K \cup$ keywords$(n_j)$
7:     if keywords($n_i$) $\subseteq K$ then
8:         remove $n_i$ from $Answer$
9: return $Answer$
```

The problem of finding a minimal answer from an *Answer* can be solved in polynomial time as shown in Algorithm 9. The algorithm checks each node in the *Answer* to see if the input keywords the node contains are all covered by other nodes. If yes, it removes the node. The complexity of this algorithm is $O(n^2)$ where n is the number of nodes in the input $Answer$.

Lemma 1 *Algorithm 9 produces a* $minAnswer$ *in which each node contains at least one* *unique input keyword. In addition, all the input keywords are covered in the $minAnswer$.*

Proof

Let n_i be a node in the answer produced by the algorithm. Assume that when the algorithm checks whether n_i should be removed, T was the intermediate answer at that time. Since n_i was not removed, **keywords**(n_i)$\not\subseteq$ **keywords**(T − $\{n_i\}$), where **keywords**(n_i) is the set of input keywords n_i contains and **keywords**(T − $\{n_i\}$) is the set of input keywords contained in $T - \{n_i\}$. Since the output $minAnswer$ is a subset of T, $minAnswer - \{n_i\}$ must be a subset of $T - \{n_i\}$. Thus, **keywords**(n_i)$\not\subseteq$ **keywords**($minAnswer$ − $\{n_i\}$), which means that n_i contains at least one unique keyword that the rest of nodes in $minAnswer$ does not contain. Also, since the algorithm only removes a node when its input keywords are completely covered by the rest of nodes in T, the set of input keywords covered by T does not change after a node is removed from T. Thus, the final $minAnswer$ covers the same set of input keywords as the input $Answer$. □

An $Answer$ may contain multiple $minAnswers$. The answer returned by Algorithm 9 may not be optimal with respect to a weight function such as $sumDistance$. Below we first prove that the problem of finding a $minAnswer$ with the minimum $sumDistance$ and $centerDistance$ is an NP-complete problem, and then present a greedy algorithm to solve the problem.

Theorem 6 *The problem of producing a $minAnswer$ from an $Answer$ while minimizing*

$sumDistance$ is NP-complete.

Proof

We prove the theorem by a reduction from the *set cover* problem. Given a set of m elements (universe) and n sets whose union is the universe, the *set cover* problem is to identify the *smallest* number of sets whose union still contains all elements in the universe. Consider the set of input keywords in our problem as a universe. The nodes in an $Answer$ can be considered as the sets of keywords whose union is the universe because they cover all the input keywords. Assume that the shortest distance between each pair of nodes in an $Answer$ is the same. Then finding a $minAnswer$ from the $Answer$ is equivalent to finding the minimal number of nodes that cover all the input keywords (i.e., the universe). This is because a $minAnswer$ with a smaller number of nodes has a smaller $sumDistance$ when the shortest distance between each pair of nodes is the same. Since the set cover problem is NP-complete [64], finding a $minAnswer$ while minimizing $sumDistance$ is NP-complete. □

Theorem 7 *The problem of producing a $minAnswer$ from an $Answer$ while minimizing $centerDistance$ is NP-complete.*

Proof

We prove the theorem by a reduction from the *set cover* problem. Given a set of m elements (universe) and n sets whose union is the universe, the *set cover* problem is to identify the *smallest* number of sets whose union still contains all elements in the universe. Consider the set of input keywords in our problem as a universe. The nodes in an $Answer$ can be considered as the sets of keywords whose union is the universe because they cover all the input keywords. Assume that the shortest distance between each node in the $Answer$ and the $center$ is the same. Then finding a $minAnswer$ from the $Answer$ is equivalent to finding the minimal number of nodes that cover all the input keywords (i.e., the universe). This is because a $minAnswer$ with a smaller number of nodes has a smaller $centerDistance$ when the shortest distance between each node and the $center$ is the same. Since the set cover problem is NP-complete [64], the problem of finding a $minAnswer$ while minimizing $centerDistance$ is NP-complete. □

Since the problem is NP-complete, we design a greedy algorithm to find a $minAnswer$ that may be sub-optimal in minimizing $sumDistance$. The algorithm is presented in Algorithm 10. It first uses a greedy set-covering procedure (Lines 1-6) to reduce the number of nodes in $Answer$ while still covering all the input keywords. The procedure chooses nodes to form an answer A as follows: at each stage, choose the node that contains the largest number of uncovered keywords. However, A may not be a $minAnswer$ because the above procedure is a greedy procedure for minimizing the number of nodes. Thus, we further sort the nodes in A based on their sum of distances to other nodes in descending order, and then call $ConvertToMinAnswerGeneral$ (i.e., Algorithm 9) to convert A into a $minAnswer$.

The complexity of the algorithm is $O(n^2)$, where n is the number of nodes in the input $Answer$. Also, since the set-covering procedure (Lines 1-6) chooses nodes from $Answer$ until all the input keywords are covered and Lemma 1 states that the $minAnswer$ produced by $ConvertToMinAnswerGeneral$ covers all the keywords, the $minAnswer$ produced by this algorithm covers all the input keywords.

Algorithm 10 ConvertToMinAnswer - Greedy Procedure for Minimizing $sumDistance$

Input: the set of content nodes as $Answer$; the query Q

Output: a $minAnswer$ with (sub)optimal $sumDistance$

```
1: $A \leftarrow \emptyset$
2: **while** $Q \neq \emptyset$ **do**
3:     **select** a node $n \in Answer$ that maximizes $|\textbf{keywords}(n) \cap Q|$
4:     $Answer \leftarrow Answer - \{n\}$
5:     $Q \leftarrow Q - \textbf{keywords}(n)$
6:     $A \leftarrow A \cup \{n\}$
7: **for** each node $n_i$ in $A$ **do**
8:     calculate $n_i$'s sum of distances to all the other nodes in $A$
9: **sort** nodes in $A$ based on their sum of distances to other nodes in descending order
   and put them in a list $T$.
10: $minAnswer = ConvertToMinAnswerGeneral(T, Q)$
11: **return** $minAnswer$
```

Theorem 8 *Algorithm 10 generates a $minAnswer$ that minimizes $sumDistance$ with* *the approximation ratio of* $(\log n)\frac{d_{max}}{d_{min}}$ *where* n *is the number of nodes in the input* $Answer$ *and* d_{max} *and* d_{min} *are the maximum and minimum distances between any pair of nodes in* $Answer$.

Proof

Assume that the number of nodes of an optimal $minAnswer$ that minimizes $sumDistance$ is opt_n and the number of nodes of the $minAnswer$ produced by Algorithm 10 is $approx_n$. Also assume that the number of nodes of an optimal answer that minimizes the number of nodes (which is the objective of the set cover problem) is opt_{scn} and the number of nodes of the approximate answer produced by lines 1-6 (i.e., the greedy set cover procedure) is $approx_{scn}$. It has been proved that the number of nodes of the answer obtained by the greedy set cover algorithm is at most $\log n$ times that of the optimal answer [64], where n is the number of nodes in the input $Answer$. That is, $approx_{scn} \leq \log n \times opt_{scn}$. Since the later steps of Algorithm 10 may further reduce the number of nodes from the answer generated by the greedy set-cover procedure, $approx_n \leq approx_{scn}$. Also, it is obvious that $opt_{scn} \leq opt_n$. Thus, we have $approx_n \leq \log n \times opt_n$. For a query with l keywords, the $sumDistance$s of the optimal and the approximation answers satisfy the following inequalities: 1) $sumDistance_{opt} \geq [\binom{l}{2} - l + opn_n] \times d_{min}$ and 2) $sumDistance_{approx} \leq [\binom{l}{2} - l + approx_n] \times d_{max}$ where d_{max} and d_{min} are the maximum and minimum distances between any pair of nodes in the $Answer$, respectively. Since $approx_n \leq \log n \times opt_n$, we have:

$$\frac{sumDistance_{approx}}{sumDistance_{opt}} \leq \frac{[\binom{l}{2} - l + (\log n \times opt_n)] \times d_{max}}{[\binom{l}{2} - l + opn_n] \times d_{min}}$$

Therefore, the following can be easily derived:

$$\frac{sumDistance_{approx}}{sumDistance_{opt}} \leq (\log n)\frac{d_{max}}{d_{min}}.$$

□

It should be noted that Algorithm 10 is guaranteed to generate a $minAnswer$. The approximation is in terms of minimizing the weight of $minAnswers$.

Since the weight of a $minAnswer$ may be smaller than that of the $Answer$ the $minAnswer$ is generated from, Algorithm 10 should be called after line 26 of Algorithm 8 using $answer \leftarrow ConvertToMinAnswer\ (answer, Q)$. After that, the weight of the answer should be updated as well. Thus, in Algorithm 8 the generated $minAnswer$ of each candidate is used to compete with the $minAnswers$ of other candidates so that the $minAnswer$ with the smallest weight among the candidates can be returned by Algorithm 8.

Since the number of nodes in $Answer$ is at most the number of input keywords, the time complexity of Algorithm 8 becomes $O(|F| \times (|D_{max}| \times l + l^2))$, where l is the number of input keywords, $|D_{max}|$ is the maximum size of D_i (the set of the nodes containing keyword k_i) and $|F|$ is the size of set F. As we discussed in previous section,

$|F| = O(l \times |D_{max}|)$. Therefore, the time complexity of Algorithm 8 becomes $O(l^2 \times D_{max} \times (D_{max} + l))$. Since l can be much smaller than $|D_{max}|$ ($l \ll |D_{max}|$), time complexity of Algorithm 10 is the same as Algorithm 8 and is equal to $O(l^2 \times |D_{max}|^2)$.

The same strategy can be applied for minimizing $centerDistance$ in Algorithm 10. The only difference is that in line 8, the distance to the center node is taken into account and the sorting of line 9 is based on this distance. Similar to Theorem 8, the approximation ratio of the algorithm for minimizing the $centerDistance$ is $\frac{d'_{max}}{d'_{min}}$ where d'_{max} and d'_{min} are the maximum and minimum distances between any nodes in the $Answer$[14] and the center node, respectively.

Theorem 9 *The above modification of Algorithm 10 generates a $minAnswer$ that minimizes $centerDistance$ with the approximation ratio of $\frac{d'_{max}}{d'_{min}}$ where n is the number of nodes in the input $Answer$ and d'_{max} and d'_{min} are the maximum and minimum distances between any nodes in the $Answer$ and the center node, respectively.*

Proof

For a query with l keywords, the $centerDistance$s of the optimal and the approximation answers satisfy the following inequalities: 1) $centerDistance_{opt} \geq l \times d'_{min}$ and 2) $centerDistance_{approx} \leq l \times d'_{max}$ where d'_{max} and d'_{min} are the maximum and minimum distances between any nodes in the $Answer$ and the center node, respectively. If center

[14]If the center node is part of the $Answer$, it is excluded for computing d'_{min}.

node is part of the $Answer$, it is excluded for computing d'_{min} because in that case d'_{min} is equal to zero. Therefore, the following can be easily derived:

$$\frac{sumDistance_{approx}}{sumDistance_{opt}} \leq \frac{d'_{max}}{d'_{min}}.$$

$\square$

4.6.2 Producing Top-k / All Minimal Answers

To generate all or top-k duplication-free $minAnswers$, Algorithm 7 is needed to divide the search space and call Algorithms 8 and 10 to find a $minAnswer$ in each subspace. This procedure works fine for finding the first best $minAnswer$ in the whole search space. However, for finding subsequent answers, the search space is divided into subspaces, each with inclusion and exclusion constraints, and the best answer from each subspace is generated to compete for the next best answer. This requires that the $minAnswer$ generated from each subspace contains all the nodes in the inclusion set of that subspace. However, when generating a $minAnswer$ from an $Answer$ and when the inclusion set is not empty, Algorithm 10 may delete some of the inclusion nodes if their keywords are covered by other nodes in the $Answer$. This may lead to generating duplicate answers by Algorithm 7. The problem can be illustrated with the following example.

Consider the graph in Figure 4.5. It contains 6 nodes: a, b, c, d e and f. Assume that

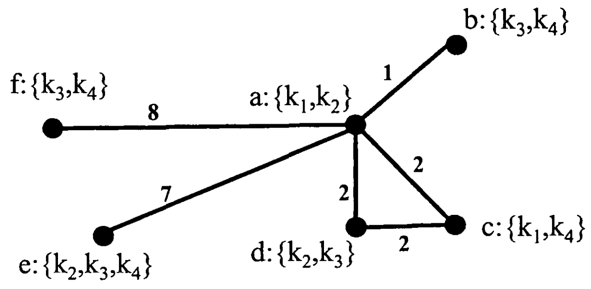


Figure 4.5: An example for clarifying the problem of violating inclusion property.

the query consists of 4 keywords k_1, k_2, k_3 and k_4. The first best answer generated by Algorithm 8 is $\{a, b\}$. Since it is a $minAnswer$, the algorithm for finding an $minAnswer$ also returns $\{a, b\}$ as the first best minimal answer. For finding the second best answer, the search space is divided into two subspaces. The first subspace has the constraints $Inc_1 = \{a\}$ and $Exc_1 = \{b\}$ and the second one $Inc_2 = \{\emptyset\}$ and $Exc_2 = \{a\}$. The procedure for finding the best answer in the first subspace returns $\{a, c, d\}$ as the best $Answer$, which is then converted to $\{c, d\}$ by Algorithm 10 as the $minAnswer$. Since node a is removed from the answer, the Inc_1 constraint is violated.

To solve this problem, we change Algorithm 8 so that all the inclusion nodes in the $Answer$ produced by the algorithm must contain at least one unique input keyword. In this way, the inclusion nodes in the answer cannot be removed when converting the $Answer$ to a $minAnswer$. Below we propose two approaches that use this strategy. The first one is called the *incomplete* approach. It is faster but may miss some answers. The second approach is called the *complete* approach. It considers all the answers but has higher time complexity than the first approach. The algorithms for both approaches are named $FindMinimalAnswer$ below. They are called in Algorithm 7 at the places where Algorithm 8 was called. Both approaches are independent of the weight function used to measure the quality of the answer.

4.6.2.1 Incomplete Approach

Based on the way the search space is divided in Algorithm 7, the nodes in the inclusion set of a subspace are part of a previously-generated $minAnswer$. Thus, each node in an inclusion set has at least one unique keyword among other nodes in the set. If in Algorithm 8 each D_i contains only the nodes that do not contain any keyword that an inclusion node contains, the inclusion nodes will keep their uniqueness and will not be removed when converting the $Answer$ to a $minAnswer$. This is the idea of the incomplete approach.

The pseudo-code of this approach is presented in Algorithm 11. Its inputs are the same as the ones for Algorithm 8. It first collects the keywords covered by the inclusion nodes into $CovKeywords$. Then it calls procedure $FindBestAnswerCovConstraint$ to generate a $minAnswer$. Procedure $FindBestAnswerCovConstraint$ is similar to procedure $FindBestAnswer$ (i.e., Algorithm 8) with two differences. The first difference is that in addition to other inputs, it also takes set $CovKeywords$ as input and in line 4 of procedure $FindBestAnswer$ the algorithm also excludes from D_i all the nodes that contain a keyword in $CovKeywords$. Since D_is store the candidate nodes to be added to the $answer$, this exclusion guarantees that no node with a keyword in $CovKeywords$ is added to the $answer$. The second difference is that the procedure calls Algorithm 10 after line 26 to convert a candidate answer to a $minAnswer$ and then calculates the weight of the $minAnswer$. The best $minAnswer$ is returned. In section 4.6.1, we have showed that calling Algorithm 10 within Algorithm 8 does not change the complexity of Algorithm 8. Thus, the time taken by Algorithm 11 is the same as Algorithm 8 and is equal to $O(l^2 \times |D_{max}|^2)$.

Algorithm 11 FindMinimalAnswer, Incomplete Approach

Input: the input graph G; the query Q; the set of content nodes C; the set of inclusion nodes Inc; the set of exclusion nodes Exc

Output: the best *minAnswer* satisfying both Inc and Exc constraints

```
1: $CovKeywords \leftarrow$ set of keywords covered by $Inc$
2: $minAnswer \leftarrow$ FindBestAnswerCovConstraint($G$, $Q$, $C$, $Inc$, $Exc$,
   $CovKeywords$)
3: return $minAnswer$
```

However, Algorithm 11 may miss some answers because it puts a too strong constraint on the search space and removes some good candidate nodes. Consider the example in Figure 4.5. The best answer in the first subspace ($Inc_1 = \{a\}$ and $Exc_1 = \{b\}$) is set $\{a, e\}$. However, since a belongs to the inclusion set, Algorithm 11 removes all of the nodes that contain a keyword in a, i.e. k_1 or k_2. Thus, e is removed from the search space because it contains k_2. Therefore, Algorithm 11 is not able to produce answer $\{a, e\}$. It produces $\{a, f\}$, which has higher weight than $\{a, e\}$.

4.6.2.2 Complete Approach

To solve the missing-answer problem of the *incomplete* approach, we propose the *complete* approach. Since each node in the inclusion set has at least one unique keyword, we first compute the set of unique keywords for each node in the inclusion set and then calculate the Cartesian product of these sets. For example, if $Inc = \{a, b\}$ and a and b uniquely contain $\{k_1, k_2\}$ and $\{k_3, k_4\}$ respectively, the Cartesian product of $\{k_1, k_2\}$ and $\{k_3, k_4\}$ is $\{k_1, k_3\}$, $\{k_1, k_4\}$, $\{k_2, k_3\}$ and $\{k_2, k_4\}$. Then, for each set s in the Cartesian product, procedure $FindBestAnswerCovConstraint$ is called with s as the input value for $CovKeywords$ to generate a $minAnswer$ whose non-inclusion nodes do not contain any keyword in s. Among all of the $minAnswers$ (each generated based on an element in the Cartesian product), the best $minAnswer$ is returned as the solution.

The pseudo-code of the complete approach is presented in Algorithm 12. It first gets the set of inclusion nodes as $\{n_1, n_2, \ldots, n_s\}$. Then, for each content node $n_i \in Inc$, it gets the unique keywords covered by n_i and stores them in K_i. The Cartesian product of $\{K_1, K_2, \ldots, K_s\}$ is calculated and stores in $CKeywordSet$ in line 3. For each member $CovKeywords_i$ of $CKeywordSet$, a $minAnswer$ is found by calling

$FindBestAnswerCovConstraint$ in line 8. Procedure $FindBestAnswerCovConstraint$ is the same as the one used in the *incomplete* approach. It finds a $minAnswer$ and makes sure that its non-inclusion nodes do not contain any keywords in $CovKeywords_i$. If the $minAnswer$ is not NULL and its weight outperforms previous minimal answers, $leastWeight$ and $bestMinAnswer$ are updated accordingly. The algorithm returns the $minAnswer$ with the smallest weight among all the $minAnswers$ corresponding to the members of the Cartesian product.

Since in each element $CovKeywords_i$ of the Cartesian product, each inclusion node has a unique keyword, the keyword will remain unique in the $Answer$ generated by $FindBestAnswerCovConstraint$ because the nodes containing that keyword will not be added to the $Answer$. Hence, the inclusion nodes in the $Answer$ cannot be removed when converting the $Answer$ to the $minAnswer$. Therefore, Algorithm 12 does not violate the inclusion constraint. In addition, since all possible combinations of the unique keywords of the nodes in the inclusion set are evaluated, no answer is missed. For the example in Figure 4.5, the *inclusion* set of the first subspace is $Inc_1 = \{a\}$. Since a contains keywords k_1 and k_2, the Cartesian product $CKeywordSet$ is $\{\{k_1\}, \{k_2\}\}$. When $\{k_1\}$ is used as the value of $CovKeywords_i$ by calling $FindBestAnswerCovConstraint$, $\{a, e\}$ is returned as the $minAnswer$, which is the best answer in the subspace that was missed by the incomplete approach.

The time complexity of the algorithm is $O((\prod_{i=1}^{s} |K_i|) \times l^2 \times |D_{max}|^2)$, where s is

Algorithm 12 FindMinimalAnswer, Complete Approach

Input: the input graph G; the query Q; the set of content nodes C; the set of inclusion nodes Inc; the set of exclusion nodes Exc

Output: the best *minAnswer* satisfying both Inc and Exc constraints

```
1: $\{n_1, n_2, \ldots, n_s\} \leftarrow$ set of nodes of $Inc$
2: $\forall i, 1 \leq i \leq s, K_i \leftarrow$ unique keywords of $n_i$
3: $CKeywordSet \leftarrow$ Cartesian product of $\{K_1, K_2, \ldots, K_s\}$
4: $leastWeight \leftarrow \infty$
5: $bestMinAnswer \leftarrow$ NULL
6: for $i \leftarrow 1$ to size($CKeywordSet$) do
7:   $CovKeywords_i \leftarrow CKeywordSet$.get($i$)
8:   $minAnswer \leftarrow$ FindBestAnswerCovConstraint($G$, $Q$, $C$, $Inc$, $Exc$, $CovKeywords_i$)
9:   if $minAnswer \neq$ NULL then
10:    $weight \leftarrow$ weight of $minAnswer$
11:    if $weight < leastWeight$ then
12:      $leastWeight \leftarrow weight$
13:      $bestMinAnswer \leftarrow minAnswer$
14: return $bestMinAnswer$
```

the average number of nodes in an inclusion set and $|K_i|$ is the number of unique input keywords in the ith inclusion node. Note that $\sum_{i=1}^{s} |K_i| \leq l-1$, where l is the number of input keywords. When the number of input keywords is small, the maximum cardinality of the Cartesian product is small. For example, for six keywords, the worst case happens when the inclusion set contains two nodes, one containing 3 unique keywords and the other containing 2 unique keywords. In this case, $\prod_{i=1}^{2} K_i = |K_1| \times |K_2| = 6$. Similarly, when $l = 3, 4, 5$ or 7, $\prod_{i=1}^{s} K_i$ is at most $1, 2, 4$, or 8, respectively. Thus, since the number of query keywords is usually small in practice, Algorithm 12 is fixed-parameter tractable (FPT) [17].

4.7 Discussion of Some Issues

4.7.1 Graph Indexing

In Algorithms 2 and 4, we need to compute the shortest distance between two nodes in the input graph. Calculating the shortest path while searching for the best answer is expensive. Since the shortest distance between any two nodes in a graph is independent of the query, we pre-built an index that stores the shortest distances between nodes. A straight forward indexing method is to calculate and store the shortest path between each pair of nodes. However, this index needs $O(n^2)$ storage, where n is the number of nodes in graph G. This index is very large and not feasible for graphs with a large number of nodes. We use the *neighbor indexing method* which is described in Section 3.8 to pre-compute and store the shortest distances and paths for the pairs of nodes whose shortest distance is within a certain threshold r_{max}. Note that the idea of indexing the graph using a distance threshold has also been used in [51, 60].

4.7.2 Presenting the Answers

The $Answers$ and $minAnswers$ produced by our algorithms are a set of content nodes. Often it is important to see how these nodes are connected to each other in the input graph. The *neighbor index* that we use stores not only the shortest distances but also the shortest paths between nodes. Thus, the relations between the nodes can be revealed using the index. In this work, we use two approaches to revealing the relationships between nodes in an answer. In the first approach, a Steiner tree that connects the nodes in an answer with the minimum weight is created **after** the answer is generated and the user indicates that he/she would like to see the connections. Figures 4.13 and 4.14 depict two trees created by our tree-generating procedure for two answers used in our user study (to be described later). Note that generating a Steiner tree from an answer is much faster than generating a tree directly from the input graph. We use an algorithm described in Section 3.7 to generate the Steiner tree. In the second approach a multi-center sub-graph is generated to reveal the relations among content nodes in an answer. A center for each answer is any node in the graph with the distance up to r to any content node in the answer [60]. A path between each pair of content nodes and centers is added to the sub-graph. The advantage of using multi-center graphs rather than trees is that it reveals more relations. The disadvantages is that it might add some irrelevant nodes to the answer [35] and the size of an answer can be overwhelmingly large. Some other methods can also be used.

4.8 Experimental Evaluation

We implemented all the algorithms presented above. In addition, for the purpose of comparison and showing that previous approaches produce duplicate and non-minimal answers, we implemented four algorithms in the literature: *Dynamic* [16], $BLINKS$ [24], *Community* [60], and r-clique [35]. All of the algorithms are implemented in Java. The experiments are conducted on an Intel(R) Core(TM) i7-2720QM 2.20GHz computer with 16GB of RAM[15].

4.8.1 Data Sets and Queries

Two real world data sets, DBLP and IMDb, are used in our experiments. The DBLP graph is produced from the DBLP XML data[16]. More details about this dataset can be found in Section 3.9.1. We used two approaches for assigning weights to the edges of

[15]The reason for using a 16GB RAM is that the *Dynamic* method stores the whole graph in the main memory. Other methods including ours use proper indexing, for which smaller RAM can be used.

[16]http://dblp.uni-trier.de/xml/

the graph. In the first approach, the weight of the edge between two nodes v and u is $(\log_2(1+v_{deg}) + \log_2(1+u_{deg}))/2$, where v_{deg} and u_{deg} are the degrees of nodes v and u respectively. This approach is called *logarithmic* edge weight and was used in [16, 32, 35, 60] . The second approach simply assigns the uniform weight of 1 to each edge. It is called *uniform* edge weight and was used in [51]. The set of input keywords used in our experiments and their frequencies in the input DBLP graph are shown in Table 3.2 in Section 3.9.1. The queries used in our experiments are randomly generated from this set of keywords with the constraint that in each query all keywords have the same frequency (in order to better observe the relationship between run time and keyword frequency). Note that the input keywords shown in Table 3.2 were generated by the authors of [60] and used to generate queries in [35, 60]. We use the same set of input keywords and the same way to generate queries to make our results comparable to others.

The IMDb dataset contains the relations between movies and the users of the IMDb website that rate the movies[17]. More details about this dataset can be found in Section 3.9.1. The edges of the graph are weighted in the same way as for the DBLP graph. For the IMDb dataset we only present the results of query with keywords *house, king, night, city, city, world* and *story*. The same set of input keywords is used in [35, 60] and in Section 3.9.1.

[17]http://www.grouplens.org/node/73

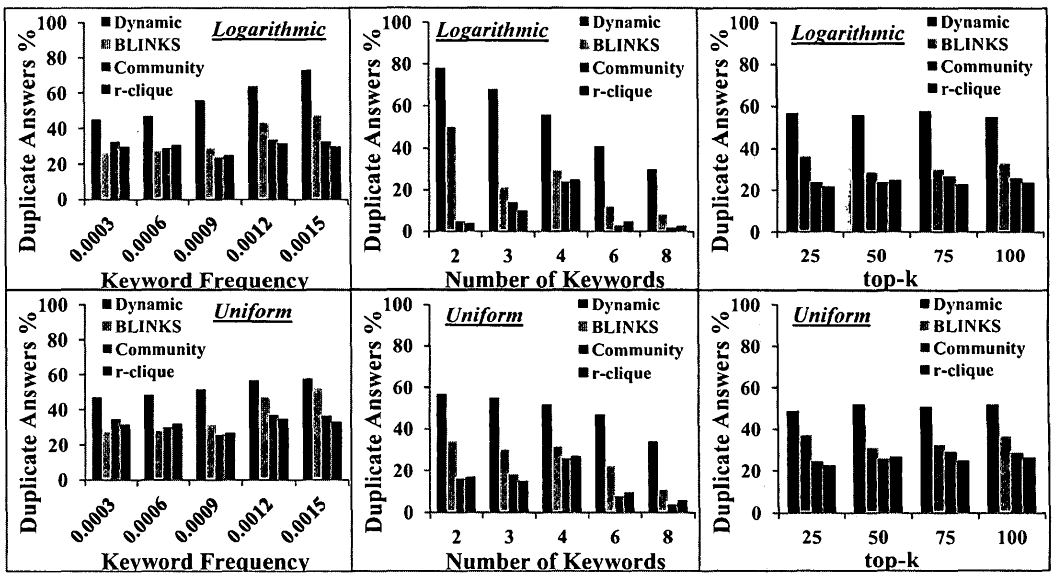


Figure 4.6: Percentage of duplicate answers of different methods with different edge weights on DBLP dataset.

4.8.2 Duplication of Previous Approaches

Our top-k method is guaranteed to generate duplication-free answers. In this section, we show the rates of duplicate answers of previous methods. Figure 4.6 shows the percentage of duplicate answers for four previous methods on the DBLP dataset with two different edge weights, different values of keyword frequency, different numbers of query keywords and different k values.[18] Two answers are considered duplicates if they have the same set of content nodes. The rate of duplicate answers in the *Dynamic* method

[18]Unless it is mentioned otherwise, in our results for DBLP, when not changing, the number of keywords is 4, keyword frequency is 0.0009 and top-50 answers are found. For the Community and r-clique methods, the r_{max} value is 8 and 5 for the *logarithmic* and *uniform* edge weights respectively.

[16] is higher than $BLINKS$ [24], $Community$ [60] and r-cliques [35]. This is because it finds minimum cost connected trees, and in most of the cases, the same set of content nodes are connected via different connections. $BLINKS$ also has a high rate of duplication. It is due to its policy of defining trees based on a unique root. The same set of content nodes may have a different root. The $Community$ and r-clique methods have the smallest rate of duplication among the existing methods because they divide the search space more wisely. But they still have some duplications. By increasing the frequency of keywords, the duplication rate of $Dynamics$ and $BLINKS$ increases. By increasing the number of keywords, the duplication rate generally decreases for $Dynamic$ and $BLINKS$. Changing the value of k does not have a significant effect on the duplication rate. All these previous methods have duplications for any value of k in the top-k answers.

The percentage of duplicate answers for 4 different methods on the IMDb dataset is shown in Figure 4.7, in which the edge weights are $logarithmic$ and r_{max} is 11 for the $Community$ and r-clique methods. The $Community$ and r-clique methods do not produce any duplicate answer for the queries used due to small numbers of content nodes (e.g., only 23 nodes contain keyword $house$). In addition, for 5 and 6 keywords, the duplication rate of all methods is close to zero due to small numbers of content nodes.

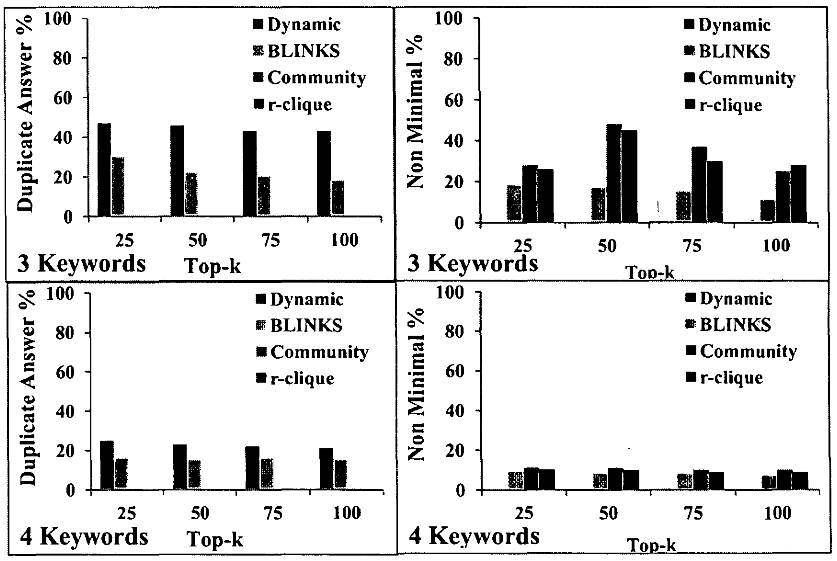


Figure 4.7: Percentage of duplicate and non-minimal answers in different methods on IMDb dataset with *logarithmic* edge weigh.

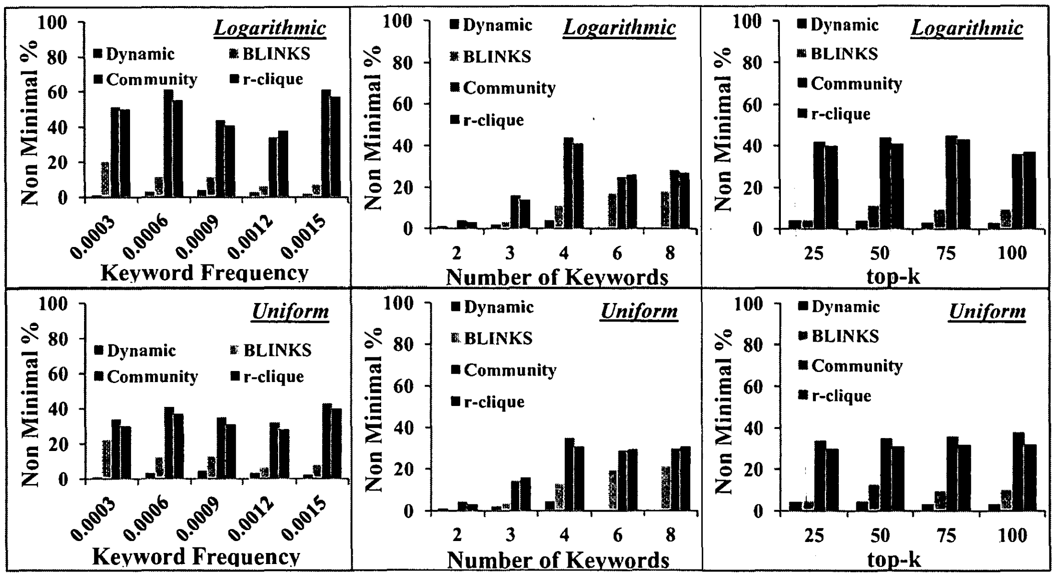


Figure 4.8: Percentage of non-minimal answers of different methods with different edge weights on DBLP dataset.

4.8.3 Non-Minimality of Previous Approaches

Both the complete and incomplete approaches proposed in this chapter are guaranteed to generate only $minAnswers$. In Figure 4.8 we show the rates of non-minimal answers of four previous methods on the DBLP dataset with two different edge weights. The rates of non-minimal answers in $Community$ and r-clique are higher than those of $BLINKS$ and $Dynamic$. This is because for each keyword, $Community$ finds the closest keyword holder to the center of the community. However, the keyword may be covered by another node associated with another keyword in the answer. This leads to non-minimal answers. The similar scenario occurs for the results of r-clique. In $Dynamic$, when merging two trees, their keywords cannot overlap. This leads to a very small rate of non-minimality. This is also valid on the IMDb dataset (Figure 4.7).

4.8.4 Run-time Comparison

One way to produce duplication-free answers is to post-process the answers generated from a keyword search method by removing the duplicates. In this section, we would like to see if our approach (which avoids generating duplicates) is faster than using the post-pruning method. Below we compare the run time of our methods to that of the r-clique and *Community* methods with the post-pruning procedure. When comparing with *Community*, we use the $centerDistance$ function which is the weight function

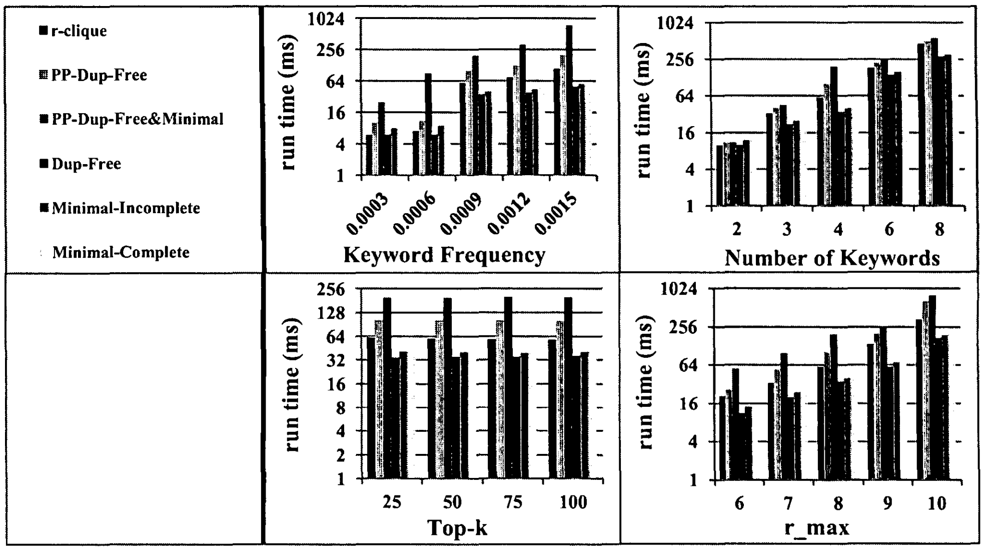


Figure 4.9: Run time of different methods with *sumDistance* proximity function and *logarithmic* edge weight on DBLP dataset. r_{max} is a distance threshold used in r-clique.

used in *Community*. Note that minimize the *centerDistance* function is slower than minimizing *sumDistance*. Note that the *centerDistance* weight function is also used in *BLINKS*. We do not compare with *Dynamic* because *Dynamic* is too slow for its results to be put into the same graph with others. We do not directly compare with the original *BLINKS* algorithm because *BLINKS* generates much fewer answers than others. That is, if we allow all the methods to generate all the possible answers, *BLINKS* only generates a subset of them while ours generates them all (i.e., *BLINK* misses some answers.[19]) Thus, due to the incompleteness of *BLINKS*, we do not com-

[19]This is due to its use of distinct root semantic for producing answers. The number of answers produced by *BLINKS* is $O(n)$ where n is the number of nodes in the graph. However, the number of answers in our model is $O(|D_{max}|^l)$ where $|D_{max}|$ is the maximum size of D_i for $1 \leq i \leq l$ and l is the number of query keywords. See Section 4.2 for an example on the incompleteness of *BLINKS*.

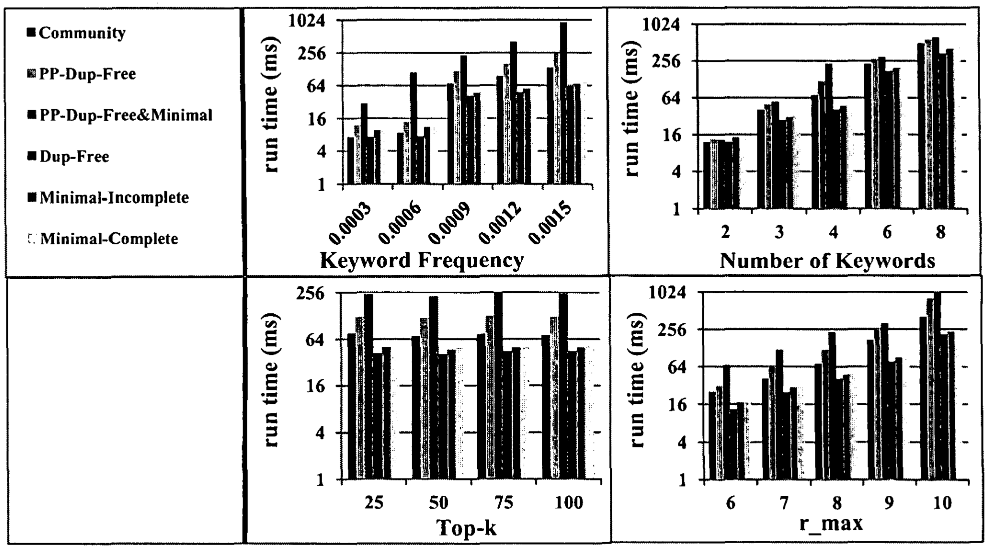


Figure 4.10: Run time of different methods with *centerDistance* proximity function and *logarithmic* edge weight on DBLP dataset. r_{max} is a distance threshold used in *Community*.

pare with the post-pruning version of original *BLINKS*, but its weight function is used in the *Community* method to compare with our approach with a modified Algorithm 8 that minimizes *BLINKS*' weight function.

Figures 4.9 and 4.10 shows the run time of different methods on DBLP with the *logarithmic* edge weight. The first method is r-clique (or *Community* in the second chart) which may generate duplicate and non-minimal answers. *PP-Dup-Free* refers to the r-clique (or *Community* in the second chart) method that post-prunes duplicate answers. *PP-Dup-Free&Minimal* refers to the r-clique (or *Community* in the second chart) method that post-prunes both duplicate and non-minimal answers. *Dup-Free* refers

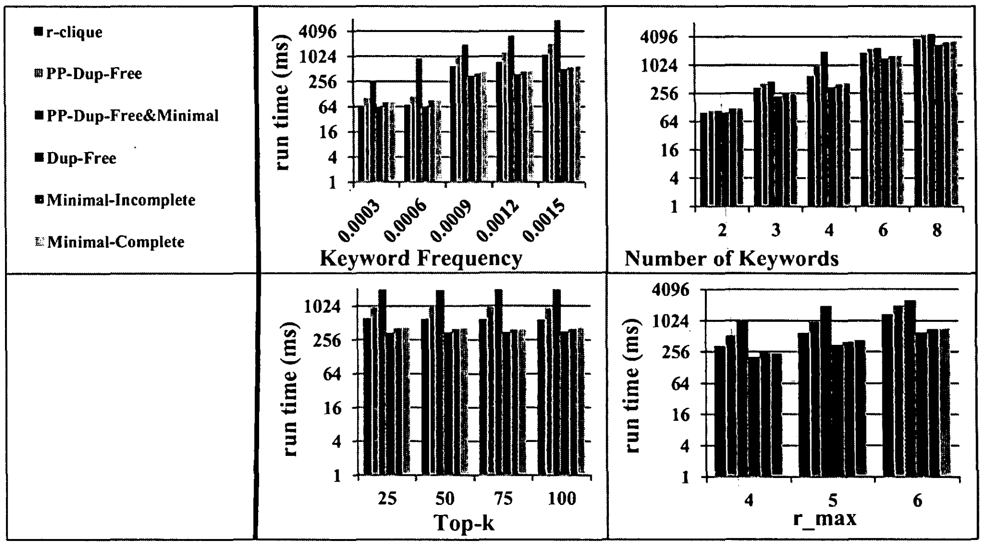


Figure 4.11: Run time of different methods with $sumDistance$ proximity function and $uniform$ edge weight on DBLP dataset. r_{max} is a distance threshold used in r-clique.

to our procedure for finding duplication free answers (i.e., Algorithms 7 and 8). The last two methods refers to our two approaches for finding duplication-free and minimal answers: the incomplete and complete approaches. To make fair comparisons, all of the methods use the same indexing method described in [35]. All the run times are the average time for producing one answer and presented in the logarithmic scale.

The run time of r-clique and $Community$ are slower than our duplication free method (*Dup-Free*) for all the three different settings. Sine they both use the same proximity measure, it seems to be a surprise. However, it is due to the fact that our method divides the search space into sub-spaces more wisely. The number of subspaces is usually smaller in our method. For example, for four keywords, assume that the best answer A contains only two nodes. The r-clique and *Community* methods divides the search space into four subspaces (equal to the number of keywords). But our procedure divides the search space into two subspaces (equal to the number of nodes in A). Since the number of nodes is always no larger than the number of keywords, we gain better performance.

The results show that finding duplication free answers with post-processing is two to four times slower than our procedure. Finding duplication free and minimal answers using post processing is three to ten times slower than each of our approaches. By increasing the frequency of keywords, the number of keywords or the value of r_{max}, the run time increases. In addition, the run time (for producing one answer) does not change when the value of k changes. It shows that they all scale well with any number of required answers. This is also the case for the $uniform$ edge weight (See Figure 4.11).

4.8.5 Incomplete vs. Complete Approaches

The incomplete approach is faster in theory but it may miss some answers. On the other hand, the complete approach can produce all answers, but is slower. Figures 4.9, 4.10 and 4.11 show that for up to 8 keywords, the run time difference between the two approaches is less than 5% (which may be hard to see on the log scale in the figures). This is due to the small cardinality of the Cartesian product when the number of keywords is small and also because the worse case rarely happens in practice. For 8 to 10 keywords, our experiments show that the run time difference is up to 20%. In terms of missing answers,

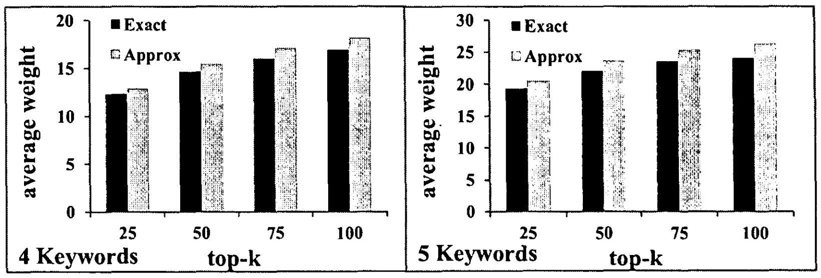


Figure 4.12: Average *sumDistances* of results from the exact and greedy algorithms for producing minimal answers on DBLP with *logarithmic* edge weight. The r_{max} value is 8, keyword frequency is 0.0009 and the number of keywords is 4.

based on our experiments, the incomplete approach misses few answers for up to six keywords (less than 1% comparing to the complete approach). For 7 to 10 keywords, the incomplete approach misses up to 5% of the answers. Thus, the performances of the two approaches are close in practice.

4.8.6 The Quality of the Approximation Algorithm for Producing Minimal Answers

To evaluate the quality of the *minAnswer* generated by the greedy Algorithm 10, we used exhaustive search to find the optimal (exact) answer that minimizes *sumDistance*. Figure 4.12 shows the average weight of the answers produced by the exact and greedy algorithms for different values of k. The results shows that the difference of the two algorithms is at most 10% in practice, suggesting the high quality of the proposed greedy

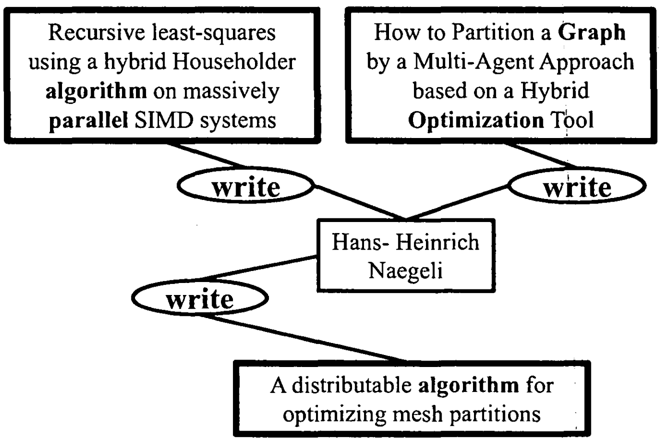

Recursive least-squares using a hybrid Householder **algorithm** on massively **parallel** SIMD systems
How to Partition a **Graph** by a Multi-Agent Approach based on a Hybrid **Optimization** Tool
write
write
Hans- Heinrich Naegeli
write
A distributable **algorithm** for optimizing mesh partitions

Figure 4.13: A tree generated from a non-minimal answer.

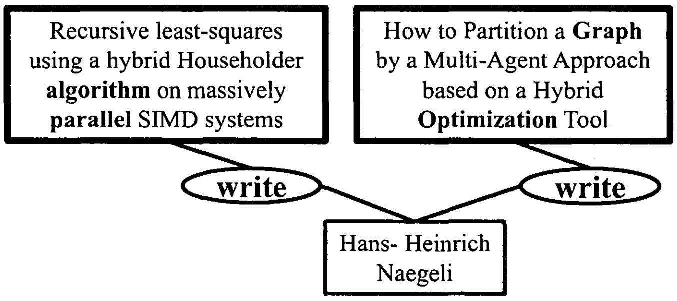

Recursive least-squares using a hybrid Householder **algorithm** on massively **parallel** SIMD systems
How to Partition a **Graph** by a Multi-Agent Approach based on a Hybrid **Optimization** Tool
write
write
Hans- Heinrich Naegeli

Figure 4.14: A tree generated from a minimal answer.

algorithm. Similar results are obtained for the $centerDistance$ function.

4.8.7 The Quality of the Minimal Answers

Finding duplication free answers is well motivated. Clearly, users prefer answers without duplication. However, it may be unclear whether users prefer minimal (more compact) over non-minimal (less compact) answers. To investigate this issue, we conducted a user

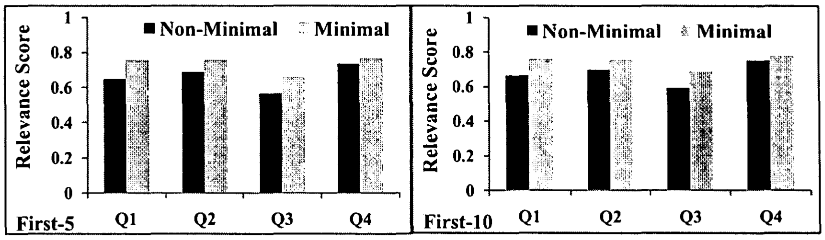


Figure 4.15: Results of the user study.

study that compares *minimal* and *non-minimal* answers in terms of their relevancy to the query. For this purpose we used 4 meaningful queries for the DBLP dataset as shown in Table 3.4 in Section 3.9.1 and applied our Algorithms 7 and 8 to find duplication-free answers. We collected the first 10 non-minimal answers from the top-100 answers for each query, and used Algorithm 10 to convert them into minimal answers. We asked 8 users (who are graduate students in computer science but not involved with this work) to compare each pair of non-minimal and minimal answers by giving each answer a relevance score between 0 and 1 with 1 meaning completely relevant and 0 completely irrelevant to the query. Each answer is presented to the user as a Steiner tree generated using the first answer presentation method discussed in Section 4.7. Figure 4.13 shows a tree generated from a non-minimal answer for the first query (i.e. "parallel graph optimization algorithm") and Figure 4.14 shows the tree for its corresponding minimal answer.

For each answer we use the average of the relevance scores from the 8 users as the relevance score of the answer. For each query, we compute the average of the relevance scores of its first k non-minimal answers, and the average of the relevance scores of their corresponding minimal answers, where $k = 5$ or 10. These average relevance scores are presented in Figure 4.15. *minimal* answers receive higher relevance scores than *non-minimal* ones in all the queries. This indicates that users prefer more compact answers as long as the set of nodes cover all of the query keywords. Also, larger answers have higher chance to include irrelevant nodes.

To further study the quality of the minimal answers, a state-of-the-art IR score is used to evaluate the answers. The IR scores are calculated based on the method used in [26]. The IR-score of a content node v for query Q is calculated as follows:

$$Score(v, Q) = \sum_{k \in Q \cap v} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s\frac{cs}{AV_{cs}}} \times \ln \frac{N + 1}{df}$$

where, for a word k that appears in both v and Q, tf is the frequency of k in v, df is the number of nodes of the same type as v that contains k[20], cs is the size of v in characters, AV_{cs} is the average size of all of the nodes with the same type as v in characters, N is the total number of nodes with the same type as v and s is a constant. The same as in [26], we set s to 0.2. Then, the combined score of the answer A that contains p content nodes is calculated as follows:

$$CombinedScore(A, Q) = \frac{\sum_{i=1}^{p} Score(v_i, Q)}{p}$$

[20]For example, if v is a paper, df is the number of the papers containing keyword k in the dataset.

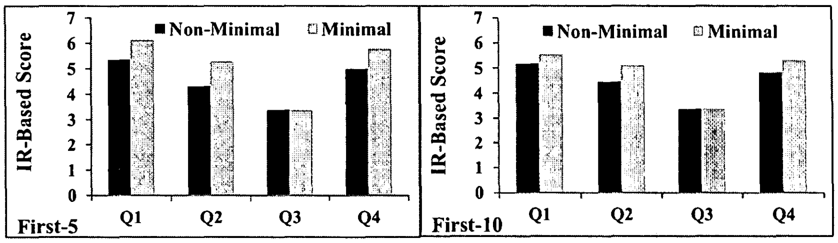


Figure 4.16: Results of the IR-Based ranking.

The IR scores of minimal and non-minimal answers for the queries in Table 3.4 are presented in Figure 4.16. The result suggests that the IR scores of the minimal answers are generally higher than the non-minimal answers (except for the third query in which the IR-scores of both of the answer sets are very close.).

4.9 Conclusion and Future Work

We proposed novel and efficient methods for keyword search in graphs. A problem with existing approaches is that they may produce duplicate answers that have the same set of content nodes with trivial differences in their connections. To address this problem, we introduced a procedure that produces duplication free answers by wisely dividing the search space. In addition, since users are usually interested in exploring more compact answers [35] and in some applications (such as textbook selection) answers with unique contributions from each node are preferred, we defined *minimal answers* and proposed two algorithms for converting an answer to a minimal answer and two approaches to finding top-k or all duplication free and minimal answers. Our algorithms are guaranteed to generate duplication-free and minimal answers. We presented the rates of duplicate and non-minimal answers produced by previous approaches. We compared the run-time of our proposed methods to that of using post-pruning techniques to remove duplicate answers. We showed that our approaches are faster than post-pruning techniques. We also showed that our greedy algorithm for minimizing the weight of a minimal answer produces minimal answers whose weights are close to the optimal weights produced by the exact algorithm. Finally, we show that the minimal answers have higher quality than non-minimal answers through a user study and a state-of-the-art IR weighting function. Our user study indicates that users prefer minimal answers to non-minimal ones. As a future work, we plan to improve the approximation ratio of the proposed algorithms. We also plan to apply our procedure for finding minimal answers in other domains other than keyword search over graphs and study the effectiveness of our approach.

5 Meaningful Keyword Search in Relational Databases with Large and Complex Schema

Keyword search over relational databases offer an alternative way to SQL to query and explore databases that is effective for lay users who may *not* be well versed in SQL or the database schema. This becomes more pertinent for databases with large and complex schemas. An answer in this context is a join tree spanning tuples containing the query's keywords. As many answers can result of varying quality, and the user is often only interested in seeing the top-k answers, how to gauge the relevance of answers to rank them is of paramount importance.

We focus on the relevance of join trees as the fundamental means to rank the answers We devise means to measure relevance of relations and foreign keys in the schema over the information content of the database. This can be done offline with no need for external models. We compare against a gold standard that we create from a real workload over TPC-E and prove the effectiveness of our measures. Finally, we test performance of our measures against existing techniques to demonstrate a marked improvement, and perform a user study to establish naturalness of the ranking.

5.1 Introduction

5.1.1 Motivation

Much of the world's high-quality data remains under lock and key in relational databases. Access is gained through relational query languages such as SQL. This can suffice for people who are well versed in both SQL and in the schemas of the databases in which they have an interest. However, a lay user—anyone who does not know SQL *or* who is not well versed in the given schema—is effectively locked out. As the schemas of the databases that organizations field become increasingly more complex year by year, we all effectively become lay users. *Keyword search over relational databases* was proposed a decade ago [1, 27] to offer an alternative way to query a database that neither requires mastery of a query language such as SQL, nor deep knowledge of the database's potentially quite complex schema.

For keyword search over databases, the database schema is thought of as a *graph*: the relations are the *nodes*, and the foreign key relationships between them are the directed *edges*. A *query* is simply a set of words (*keywords*). The concept of what constitutes an *answer*, however, is more involved. For an answer, we want to convey elements from the database—namely tuples—that *cover* the keywords of the query, and a natural

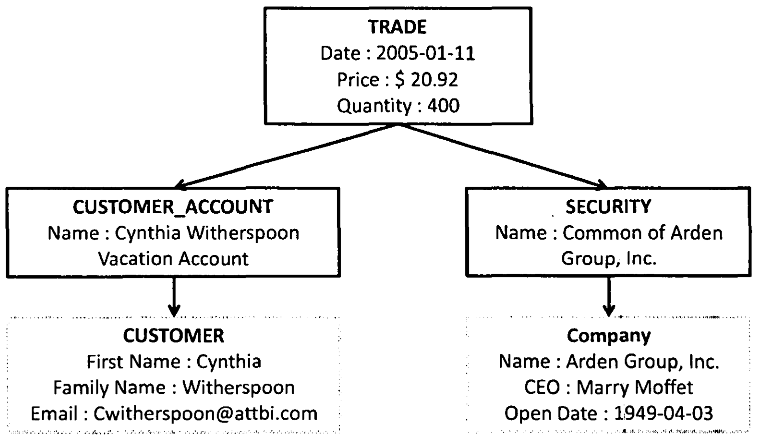


Figure 5.1: An answer considered non-minimal by DISCOVER.

structure—a sub-graph of the database's schema—that spans those elements. This sub-graph is commonly called a *network* in the literature [27, 28], but is effectively a *tree*, called a *join tree* in [1].

What is an admissible answer is usually further restricted. We are not interested in *any* tree; some may only very loosely connect the tuples containing our keywords. Previous work restricts answers over *minimal* trees [27, 28], meaning there is no answer over a sub-tree of the tree in question. However, there is another aspect of an anwer: relevance to the query. By restricting answers with a minimal structure, some relevant answers may be missed.

Consider the keyword query *"Cynthia Arden"* over the TPC-E[21] schema. The TPC-E benchmark simulates the *on-line transaction processing* (OLTP) workload of a broker-

[21] http://www.tpc.org/tpce/

age firm.[22] The keywords *Cynthia* and *Arden* may appear in different relations. Each could refer to the name of a *customer*, *broker*, *company*, or CEO of a *company*, or be in the title of a *news_item*. Of course, each of these different relations for *Cynthia* and for *Arden* potentially lead to rather different answers. For example, Figure 5.1 shows an answer for the keyword query *"Cynthia Arden"* on TPC-E, which says that a customer *Cynthia* buys the stocks of a company named *Arden Group*. This is an interesting and relevant relationship between *Cynthia* and *Arden* assuming the user wants to find out the relationship between customer *Cynthia* and company *Arden Group*. However, such an answer may be missed if we restrict the answer to be structually minimal while covering all the query keywards. This is because word *Cythia* also appears in an intermediate node of the tree and thus the *customer* node is pruned.

Another issue in keyword search is to score answers for *relevance*. Some answers are more relevant than others, and thus the answers should be presented in order of descending relevance. The relevance of an answer depends on the value of many factors: the tuples in the answer and how the keywords appear in them, the bridging tuples that do not contain keywords, and the join tree. How to measure the value of each of these is open to question. How to then combine these measures into a single relevance score is also open to debate.

[22]The schema contains 33 tables that can be clustered into four parts [68]: *customer*, *market*, *broker*, and *dimension*. The TPC-E schema is shown in Figure 5.3. The database models information about financial transactions such as traded companies, fees of brokers, customer accounts and their related holdings, and the type of traded securities.

Prior work has addressed relevance. In [27], they offer the simplistic solution of scoring relevance as the reciprocal of the number of edges in an answer's tree. This heuristic assumes that fewer joins involved mean the tuples are related more closely. However, this relevance scoring method has the following problem. Consider the query $\{Anderson\ Joseph\}$ over the TPC-E schema. Figure 5.2 shows four possible join trees of different sizes that could produce answers that connect company $Anderson$ and customer $Joseph$. If we ranks the trees according to their size (i.e., the number of edges or nodes), the first tree (a) which connects a customer and a company when they have the same status gets the highest rank. However, based on the TPC-E schema description, this is not a strong relationship. *Status_Type* is a dimension table and it is connected to six tables in the schema and stores the status value for other entities (such as companies or customers). An answer derived from this tree would say that both $Anderson$ and $Joseph$ are $active$. Looking at the *customer* and a *company* tables in TPC-E, it turns out that all the customers and companies have the *Active* status. Therefore, any given customer and any given company in the TPC-E database share the same status through the *Status_Type* table. Thus, the found relationship between $Anderson$ and $Joseph$ is not interesting or relevant.

In [28], the authors take a very different track: they apply information-retrieval (IR) measures to the answers to determine an individual measure per answer. This leverages approaches from IR that work well in other domains of keyword search. An answer

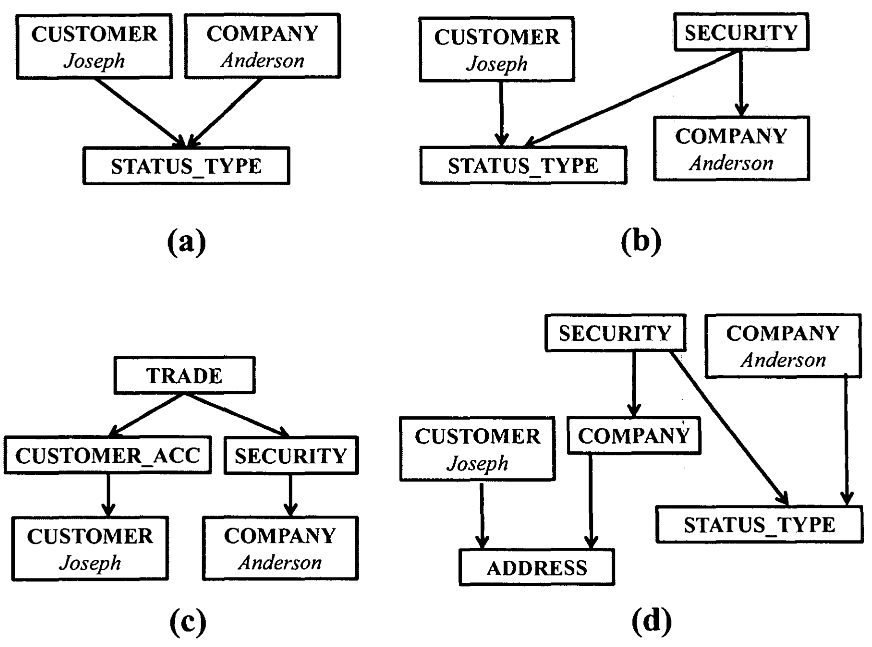


Figure 5.2: Four possible trees for the query *"Andersen:company Joseph:customer"*.

in this case, however, need not cover *all* the query's keywords to score well. And the approach de-emphasizes the importance of the tree.

We believe that for keyword search in relational databases the relevance of the tree from which answers derive is paramount. We hypothesize that relevance as measured by adapted IR techniques is much less effective. Previous work on relevance has been tested over simple schemas. For application over more complex schemas, the importance of the schema (thus, answers's trees) becomes more pronounced. It is *how* the tuples are related (via the joins) that is meaningful for search in the relational domain, after all.

5.1.2 Objectives and Contributions

While solutions have been proposed for the above problems, it is well recognized that there is room to improve. In this chapter, we tackle the problems by (1) defining answers that incorporate the user's interest and (2) devising meaningful relevance scores for answers.

We address the first issue by allowing the user to specify the role of each query keyword. A first step to keyword query evaluation in a relational database is to determine the tuples in the database that contain keywords from the query.[23] We call a relation that contains a tuple containing a keyword a potential *role* for that keyword. Finding the potential roles for the query's keywords can be done efficiently, by using an inverted index structure or existing built-in support for full-text keyword search in the RDBMS [61].

In keyword search, the user does not commonly specify the roles; just the keywords. However, adding a role selection step is beneficial. First, the user focuses the query to roles of interest. Second, this will admit meaningful answers that would not otherwise be found. The answer in Figure 5.1 is *not* found by DISCOVER [27] because of the minimality constraint they add to their definition of "answer", and that roles are not part of the definition. This minimality issue is addressed in depth in Section 5.2.

[23]Keywords may appear in different columns and in different tables. This ambiguity may resolved by the user, or by using automated techniques such as in [7].

Allowing the user to select a role for each query keyword may require the user have some knowledge of the database. However, a user-interface for the system could present options in a way that the user still need not be familiar with the database schema. Alternatively, in [7], the authors propose a method for ranking the role of each keyword, which may be used to automatically select a role for each query keyword.

To tackle the second problem (i.e., devising meaningful relevance scores for answers), we seek to measure importance of edges and nodes (foreign keys and relations, respectively) in the graph (the database schema). The relevance of an answer's tree can be then determined based on the importance value of its nodes and edges. We adapt and leverage techniques from recent work on *database summarization* [67, 68] for this.

Our relevance measures only rely on the database's schema and data. We do not rely on other models, workloads and logs, or other external information. Our node and edge relevance can be computed offline and efficiently.

Our contributions are as follows.

1. *Model for keyword queries in relational databases.*

   Redefine *answer* via roles (as discussed above and defined in Section 5.2) that captures important answers missed by previous techniques.

2. *Schema-based ranking.*

   (a) Devise importance measures for nodes, importance measures for edges, and a hybrid measure of the two.

(b) Devise relevance measures for join trees derived from the schema relevance. Consider the effect of penalizing larger trees.

(c) Construct a *gold standard* for relevance of nodes and edges from an extensive workload of real SQL queries. This is used to evaluate the effectiveness of our measures which do not require such external information.

3. *Evaluation.*

   Establish the efficacy of our approach by evaluation.

   (a) Perform a comprehensive evaluation based on TPC-E to demonstrate the viability of our methods and to compare against existing methods. The experiments are over a much larger and more complex schema (TPC-E) than the experiments in previous work.[24]

   (b) Run a user study to establish the meaningfulness of answers and ranking generated by our system.

The chapter is organized as follows. In Section 5.2, we present our framework. In Section 5.3, we devise our measures for relevance of nodes and edges, and for join trees. We show how to compute these efficiently. In Section 5.4, we discuss the comparison of the methods against the gold standard, we evaluate our methods against existing methods, and we present the results of the user study. In Section 5.5, we conclude.

[24]In [9], keyword search over the Credit Suisse data-warehouse is considered, which has a complex schema. However, the authors use meta-data and patterns to build a model. We assume no extra information.

5.2 Framework

5.2.1 Problem Statement

A relational database schema consists of a set of n relation schemas, denoted as $\{R_1, R_2, \ldots, R_n\}$, where each R_i is described by a set of attributes. Two relation schemas, R_i and R_j, may be related by a *foreign key relationship*, denoted as $R_i \leftarrow R_j$, where the primary key of R_i is referenced by the foreign key of R_j. Thus, a relational database schema can be considered as a graph $G = \langle V, E \rangle$, where nodes are relation schemas and edges represent the foreign key relationships.

A relational database is an instance of a relational database schema G. It consists of a set of relations, where each relation $r(R_i)$ is a set of tuples conforming to the relation schema R_i in G. Given a relational database D and a set of $l(\geq 2)$ keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), the problem of keyword search in D is to find a set of tuples that are connected via foreign key relationships and cover all the keywords in Q. The most representative algorithm for this problem is DISCOVER [27], which finds *minimal total joining networks of tuples* defined as follows.

Definition 7 ***Minimal Total Joining Network of Tuples (MTJNT):*** *Given a database with schema graph G and a query Q containing a set of keywords, a minimal total joining network of tuples is a tree T of tuples that satisfy the following conditions:*

- ***Joinable:*** *for each edge (t_i, t_j) in T, where $t_i \in r(R_i)$ and $t_j \in r(R_j)$, there is an* *edge $R_i \leftarrow R_j$ or $R_i \rightarrow R_j$ in G and $t_i \bowtie t_j \in r(R_i) \bowtie r(R_j)$.*

- ***Total:*** *each keyword in Q is contained in at least one tuple in T.*
- ***Minimal:*** *if a tuple in T is removed, T is either not joinable or not total.*

The DISCOVER algorithm generates all the MTJNTs given a database and a query. It does not specify the role of a query keyword (e.g., it does not care in which relation a query keyword should appear). Thus, an answer that contains a tuple with a keyword in an interesting relation may not be found if the answer is not *minimal*. This occurs when an intermediate tuple connecting the tuples in the interesting relations with the query keywords also contains a query keyword, which leads to a tuple in an interesting relation being pruned to make the network minimal. To illustate this issue, an example is shown in Figure 5.1. Assume that the user is interested to see the relationships between a *Cynthia:customer* and a *Arden:company*. One of the most interesting relationships in TPC-E is that*Cynthia* buys the stocks of *Arden*, which is shown in Figure 5.1. However, it cannot be discovered by the DISCOVER algorithm since both keywords (*Cynthia* and *Arden*) appear in the intermediate tuples (i.e. *customer_account* and *security*) and thus the answer is considered non-minimal by DISCOVER.

In our framework, the role of a query keyword can be specified by the user or automatically detected using a method such as the one in [7]. For each query keyword, our algorithm first finds the list of relations that contain the keyword using an inverted index or the built-in support for full-text keyword search in DBMS [61], and then a relation is chosen (either by the user or automatically) from the list as the ***role of the keyword***. With specified keyword roles, our algorithm searches for answers that are defined as follows.

Definition 8 ***Minimal Joining Network of Tuples Covering Roles:*** *Given a database D with schema graph G and a query Q containing a set of keywords $\{k_1, k_2, \ldots, k_l\}$ and their respective roles $\{r_1, r_2, \ldots, r_l\}$ (where r_i's are relations in D), a minimal joining network of tuples for query Q is a tree T of tuples that satisfy the following conditions:*

- ***Joinable:*** *for each edge (t_i, t_j) in T, where $t_i \in r(R_i)$ and $t_j \in r(R_j)$, there is an edge $R_i \leftarrow R_j$ or $R_i \rightarrow R_j$ in G and $t_i \bowtie t_j \in r(R_i) \bowtie r(R_j)$.*
- ***Role and keyword covering:*** *for each query keyword role r_i, there exists a node t_j in T such that $t_j \in r_i$ and t_j contains keyword k_i.*
- ***Minimal:*** *if a tuple in T is removed, T is either not joinable or does not cover all the roles or all the keywords.*

For brevity, we refer to the answer defined in Definition 8 as ***final answer*** in this chapter. Given a database, there may be many final answers to a query. Instead of producing all the answers which may overwhelm the user, the goal of our algorithm is to produce top-k most meaningful final answers. In the next section, we describe our procedure for generating top-k final answers.

5.2.2 Methodology

The final answers defined above can be generated through a sequence of join operations on the database. To generate such answers, we first generate *minimal joining networks of schemas (MJNSs)* that represent the join operations for producing the final answers. Below we define MJNS and then present our method for generating MJNSs.

Definition 9 ***Minimal Joining Network of Schemas (MJNS)****: Given a database D with schema graph G and a set r of query keyword roles $\{r_1, r_2, \ldots, r_l\}$ (where r_i's are relations in D), a minimal joining network of schemas that cover r is a tree T of relation schemas in G that satisfy the following conditions:*

- ***Joinable:*** *each edge in T is an edge in G. That is, each edge in T represents a foreign key relationship.*
- ***Role covering:*** *for each query keyword role r_i, its schema is in T.*
- ***Minimal:*** *if a relation schema in T is removed, T is either not joinable or does not cover all the roles in r.*

Note that our MJNSs bear similarity to the *candidate networks (CN)* used in the DISCOVER algorithm [27] with the following differences. First, a CN is defined as a network of tuple sets, while our MJNS is defined at the schema level. Second, our MJNS must cover a set of specified roles (i.e., it must contain the schemas of a set of specified relations), while a CN does not have to. This second difference allows us to find some interesting final answers that DISCOVER misses as discussed earlier.

To generate MJNSs, we use a breadth-first search algorithm similar to the CN generator of DISCOVER. Our algorithm starts with a role schema as an initial tree T and extends T with a relation schema in G that has a foreign key relationship with a node in T. The expansion of T stops once the schemas of all the roles are covered in T. To avoid generating duplicate trees, each generated tree is assigned an ID based on tree isomorphism during the execution of the algorithm. The ID of a tree is checked with the existing IDs that are generated so far and the current tree is accepted if it is not generated previously. Figure 5.2 shows 4 MJNSs generated for query $\{Andersen, Joseph\}$ and their respective roles $\{Company, Customer\}$ over the TPC-E database.

After MJNSs are generated, final answers can be produced by creating an execution plan to evaluate the MJNSs. Note that an MJNS may or may not produce a final answer[25]. But a final answer can be produced by one and only one MJNS. The union of the final answers produced by all the MJNSs is the set of all possible final answers.

Since many final answers can be generated for a query and some answers may not be interesting, we aim at producing top-k most interesting final answers. To achieve this purpose, we first limit the number of nodes in an MJNS (which is a strategy taken by DISCOVER as well). This limits the size of a final answer too. The rationale is two-

[25]In [27] it is claimed that all the candidate networks generated by DISCOVER lead to generation of an MTJNT (the final answer of DISCOVER). But its CN generation algorithm works at the schema level without checking whether a join in CN can produce an answer. Thus, there is no guarantee that a CN can produce at least one final answer.

fold. First, if two tuples in a final answer are far away from each other, it is not easy to interpret the answer [70]. Second, executing the query associated with a large MJNS is time consuming. Thus, a size control parameter, D_{max}, is used to specify the maximum number of allowed nodes in an MJNS. In addition and **more importantly**, we rank the generated MJNSs according to an interestingness measure so that final answers from the top-ranking MJNS are produced first, and if the number of final answers produced so far is less than k, the next MJNS is used to produce more final answers until k final answers are produced.

The overall procedure of our seach method is described below. Given a database D, a query containing keywords $\{k_1, \ldots, k_l\}$, an answer size control parameter D_{max}, and a maximum number k of final answers to be returned,

1. For each keyword k_i, find the relations in D containing k_i using an inverted index or the built-in support for full-text keyword search in DBMS [61];
2. Select a relation containing k_i as the role of k_i either by the user or automatically using a role-ranking method in [7];
3. Generate the MJNSs that cover all the selected roles and whose size is no more than D_{max};
4. Rank the generated MJNSs according to an interestingness measure;
5. For each MJNS m_i in the ranked list, evaluate m_i to generate a set s_i of final answers, rank the answers in s_i according to a content-based IR-style ranking mea-

sure, and add the answers in the ranked order into the final answer set A. This procedure stops until either A contains k answers or all m_i's have been evaluated.

The main focus of this chapter is on Step 4: how to rank MJNSs so that the most interesting answers will be presented to the user first and less interesting ones can be pruned. Note that the IR-style ranking measure in Step 5 is a secondary ranking measure for locally ranking the final answers generated from each MJNS, which is exactly the same as the method in [28]. In the next section, we present a number of measures for ranking MJNSs in Step 4.

5.3 Ranking Models

In this section, we propose a few methods for ranking minimal joining networks of schemas (MJNSs). The proposed methods work only based on the database schema and its given instance, assuming that no extra information (e.g., query logs or inheritance relationships among the tables) is available. The methods use information-theoretic measures to evaluate the importance of a relation (i.e., table) and/or an edge in the given database, and rank the MJNSs based on the importance of the relations and/or edges in an MJNS. We classify the proposed methods into three categories: (1) ranking based on the importance of the nodes in MJNS, (2) ranking based on the importance of the edges in MJNS, and (3) ranking based on the importance of both nodes and edges in an MJNS.

5.3.1 Ranking by Importance of Nodes

Given a database D, this type of methods assumes that the importance of an MJNS M is related to the importance of the tables in D that instantiate the schemas in M. There are two recent works in the literature that study the problem of measuring table importance in a database [30, 67]. Both methods define the importance of a table as the steady state probability of the table in a random walk over the database graph G. The purpose of the work in [30] is to generate a forms-based database query interface and that in [67] is to summarize a relational database.

The method in [30] assumes that the importance of a table is proportional to the number of tuples, the number of attributes that it contains and the number of connections (i.e., foreign key joins) to other tables. Based on the assumption, a probability matrix for the random walk is built. As discussed in [67], this is a reasonable assumption for an XML schema on which the method was evaluated, but they may fail to produce good results on relational databases, especially on data warehouses that have dimension tables (e.g. *address* and *zip_code* in TPC-E). Dimension tables usually have many connections to other tables. However, such connections hardly form an interesting relationship among the tuples they connect. For example, the first MJNS in Figure 5.2 connects the customer and the company through the *status_type* dimensional table. As mentioned before, it is not an interesting relationship between the given query keywords. The way the probability matrix is defined in [30] results in dimension tables gaining more importance than fact tables, which is not desirable in relational databases and data warehouses.

In [67], four methods for measuring the importance of a table are presented and shown to outperform the method in [30] for *summarizing* relational databases. Their experiments show that one of the methods, called *variable entropy transfer (VE)*, outperforms the three other methods (See [67] for details of the three other methods). None of these methods have been used for keyword search. We propose to use the VE method to evaluate the importance of each table and rank MJNSs according to their table importance. We further propose a modified version of the VE model, called *key entropy transfer (KE)*. Below we first describe the VE method and then present the KE method. In our experiments, both methods are compared to the three other methods introduced in [67].

5.3.1.1 The VE Method

Consider $G_D = (V_D, E_D)$ as an undirected graph representing a relational database D, where V_D is the set of nodes representing relations (i.e., tables) in D and E_D is the set of edges representing foreign key relationships. Both VE [67] and KE methods build a node-to-node transition probability matrix M based on the entropies of table attributes, and perform a random walk on G_D with M. The steady-state probabilties of the random walk are then assigned as the importance scores of the tables.

Let $r.A$ denote an attribute A in table r, and let a represent a value of $r.A$. The *entropy* of $r.A$ is defined as follows:

$$H(r.A) = -\sum_{\forall a \in r.A} p(a) \times \log p(a) \tag{5.1}$$

where $p(a)$ is the probability that a occurs in column $r.A$ (i.e. $P(r.A = a)$).

For each table in D, a primary key is created that consists of all of the attributes in the table, and a self-loop using this key is added to the corresponding node in G_D. This is done even when the table already has a primary key. The purpose of the self-loop is to keep some of the table's information within the table during the random walk (i.e., to make the random walk have more probability to stay at a node). Assume that table r contains m attributes $\{r.A_1, \ldots, r.A_m\}$, excluding the self loop attribute. The *information content* of table r in the VE model is defined in [67] as follows:

$$IC_{VE}(r) = \log |r| + \sum_{i=1}^{m} H(r.A_i) \tag{5.2}$$

where $|r|$ is the number of tuples in r, and $\log |r|$ is exactly equal to the entropy of the self loop attribute. The IC_{VE} value of a table measures the importance of the table without considering the foreign key relationships between tables.

To take into account such relationships, the *information transfer* rate on a join edge starting from $r.A$ (no matter if $r.A$ is a primary or foreign key in the connection) is defined as follows:

$$T(r.A \rightarrow r'.A') = \frac{H(r.A)}{log|r| + \sum_{i=1}^{m} n_{X_i} \cdot H(r.X_i)}$$
$$= \frac{H(r.A)}{IC_{VE}(r) + \sum_{i=1}^{m}(n_{X_i} - 1) \cdot H(r.X_i)} \quad (5.3)$$

where n_X denotes the total number of join edges involving attribute $r.X$, including the self loop. Since the self loop contains all the attributes of r, $n_{X_i} \geq 1$. Note that $T(r.A \rightarrow r'.A')$ does not depend on $r'.A'$, which means that the same amount information is transferred from $r.A$ along any edge starting from A.

A *transition probability matrix* for the random walk is then built by

$$\Pi(r, r') = \sum_{r.A - r'.A'} T(r.A \rightarrow r'.A') \quad r \neq r' \quad (5.4)$$

where the sum ranges over all the join edges between r and r' in the database graph G_D. Additionally, $\Pi(r, r)$ is defined as:

$$\Pi(r, r) = 1 - \sum_{r \neq r'} P(r.A \rightarrow r'.A') \quad (5.5)$$

The probability matrix is an $n \times n$ matrix, where n is the number of tables in the database. It is a matrix with numbers between 0 and 1 and each row sums up to 1. If two tables r and r' are connected together in G_D, $\Pi[r, r']$ is greater than zero and its value determines the amount of information transferred along the (r, r') edge. If more than one join edge exists between two tables, the information on each join edge is summed up.

The importance of a table r is defined as its stable-state probability of a random walk on G_D with the probability matrix Π. For any given probability matrix Π over a connected and non-bipartite graph G, there exits a unique stationary distribution $\beth$ [55]. Therefore, the table's importance in the above model is well defined. The stationary distribution vector can be obtained by applying an eigenvector calculation method. We use an iterative method used in [67]. The method starts with an arbitrary non-zero vector $\beth_0$[26]. Then, $\beth_{i+1} = \beth_i \times \Pi$ is computed repeatedly until the distance between $\beth_i$ and $\beth_{i+1}$ is less than a given threshold. Setting the threshold to zero results in stopping the method when the stationary distribution is reached.

5.3.1.2 The KE Method

To compute $IC_{VE}(r)$ in the VE method, entropies of all the attributes in r needs to be computed, including the numeric attributes. To compute the entropy for numeric attribute, discretization should be performed first in order to compute the probabilities involved in the entropy computation. Thus, the entropy value of a numeric attribute greatly depends on the discretization method used or the parameter value used in a discretization method (such as the number of intervals in the equi-width discretization method). To avoid such dependencies, for a numeric attribute, we set its entropy to the maximum value, which is $log|r|$. We further discovered in our experiments that for a non-joinable

[26]The final value of stationary distribution $\beth$ does not depend on the initial value $\beth_0$. However, we set $\beth_0$ to $IC_{KE}(R)$.

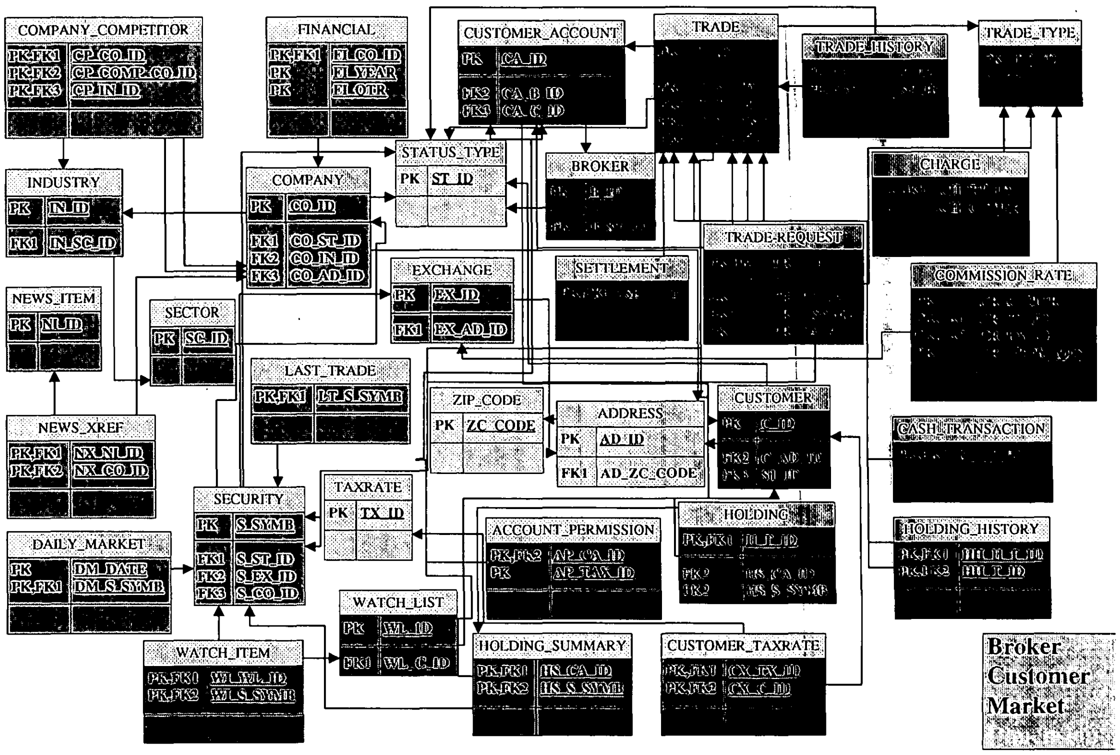


Figure 5.3: The TPC-E schema graph. (Figure from [68].) The direction of the edges are from the foreign key to the primary key.

attribute (i.e., an attribute that is not a primary or foreign key), using its true entropy or the maximum entropy (i.e., $log|r|$) does not make much difference in terms of finding meaningful MJNSs (which will be shown in our experimental results). Thus, to speed up the entropy computation, we use $log|r|$ as the entropy value for all the non-joinable attributes.

Thus, we define the *information content* of a table r as follows:

$$IC_{KE}(r) = (|C_r^{nj}| + 1) \times \log |r| + \sum_{A \in C_r^j} H(A) \tag{5.6}$$

where C_r^{nj} and C_r^j denote the set of non-joinable and joinable columns of table r, respectively. (Each column in C_r^j is part of at least one edge and is either a primary or foreign key attribute.) In this equation, each non-joinable column has the $\log |r|$ contribution to the importance of the table.

Accordingly, the information transfer rate along a join edge $(r.A, r'.A')$ is defined as follows:

$$T(r.A \rightarrow r'.A') = \frac{H(r.A)}{IC_{KE}(r) + \sum_{X \in C_r^j} [(n_{r.X} - 1) \times H(r.X)]} \tag{5.7}$$

where $n_{r.X}$ is the total number of join edges that attribute $r.X$ is involved, including the self loop primary key.

Based on T, the probability matrix used in the random walk is built in the same way as described in Equations 5.4 and 5.5. The importance of a table is the table's stable-state probability of the random walk on G_D with the probability matrix. Since the entropies of only primary and foreign keys are truly computed, this method is referred to as the *key entropy (KE)* method.

5.3.1.3 Ranking MJNSs

The importance scores of the tables in a database can be computed offline before a keyword search starts[27]. During the keyword search we use the scores to rank the minimal joining networks of schemas (MJNSs). Given a set of query keywords and their corresponding roles, all the MJNSs generated in the procedure described in Section 5.2.2 share the same set of role relation schemas. Thus, to rank the MJNSs, we only need to consider the non-role relation schemas in each MJNS. We use the average importance score of the tables associated with these non-role relation schemas to compute an importance score for an MJNS M, as defined below:

$$Score_{table}(M) = \frac{\sum_{r \notin Roles} TableScore(r)}{n} \tag{5.8}$$

where $Roles$ is the set of role relations, n is the number of non-role relation schemas in M, and $TableScore(r)$ is the importance score of table r computed using either the VE or KE method. Note that if a non-role relation schema appears more than once in the given MJNS, it is counted more than once (equal to the number of occurrences in the MJNS). This is reasonable since the more a relation schema appears, the more important it is for connecting the role relation schemas together in the MJNS.

[27]The scores should be updated periodically if the database content changes.

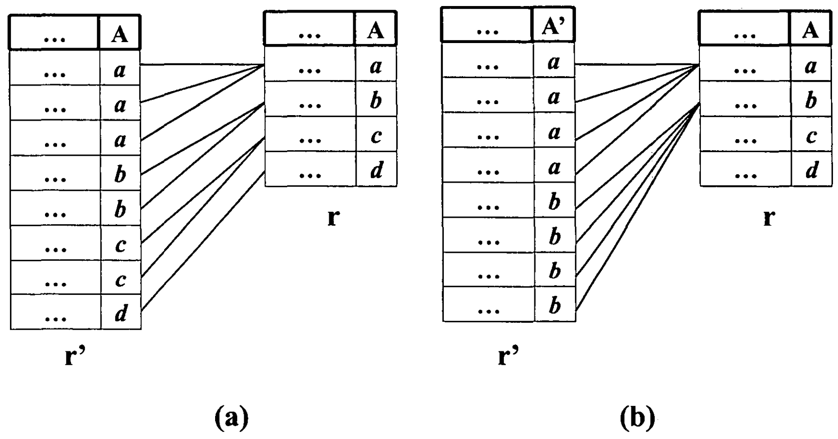


Figure 5.4: Two different instances of the foreign key connections between tables r and r'

5.3.2 Ranking by Importance of Edges

Another approach to ranking the MJNSs is based on the importance (strength) of the edges (i.e. foreign key connections) that connect the nodes in an MJNS.

A schema edge weighting function was recently introduced in [68] based on the information theory [63]. However, the function was used for *summarizing* schema graphs and has not been applied to keyword search. Below we first introduce this function and then present two new measures. In our experiments, all the three methods are evaluated.

5.3.2.1 Edge Strength by Information Theory

Let r and r' denote two relations in a database, and $r.A$ and $r'.A'$ denote the two attributes by which r and r' can be joined. Let a or a' represent a value of $r.A$ or $r'.A'$, respectively.

The *joint entropy* of two variables $r.A$ and $r'.A'$ is defined as follows:

$$H(r.A, r'.A') = -\sum_{\forall (a,a') \in (r.A, r'.A')} p(a, a') \times \log p(a, a') \tag{5.9}$$

where $p(a, a')$ denotes the joint probability $p(r.A = a, r'.A' = a')$. The *marginal probability* on $r.A$ in the joint distribution of $r.A$ and $r'.A'$, denoted as $p_{r.A}(a)$, is defined as follows:

$$p_{r.A}(a) = \sum_{\forall a' \in r'.A'} p(a, a') \tag{5.10}$$

In other words, $p_{r.A}(a)$ is the probability that a occurs as the value of $r.A$ in the instantiated join edges between $r.A$ and $r'.A'$. The marginal probability $p_{r'.A'}(a')$ is defined in the same way and is the probability that a' occurs as the value of $r'.A'$ in the instantiated join edges between $r.A$ and $r'.A'$.

The *pointwise mutual information* of a and a' (belonging to $r.A$ and $r'.A'$, respectively) is defined as follows:

$$i(a, a') = \log \frac{p(a, a')}{p_{r.A}(a) \times p_{r'.A'}(a')} \tag{5.11}$$

The *mutual information* of $r.A$ and $r'.A'$ is defined as the expected value of the pointwise mutual information.

$$I(r.A, r'.A') = \sum_{\forall (a,a') \in (r.A, r'.A')} p(a, a') \times i(a, a') \tag{5.12}$$

In [68], the following distance function is used to measure the dissimilarity between two database columns:

$$D(r.A, r'.A') = 1 - \frac{I(r.A, r'.A')}{H(r.A, r'.A')} \tag{5.13}$$

The smaller the D value, the closer the columns are. It can be proved that $I(X, Y) \leq H(X, Y)$, and $D(X, Y)$ is a metric with $D(X, Y) = 0$ and $D(X, Y) \in [0, 1]$.

Using this distance function, the strength of the edge between the two database columns can be defined as:

$$ST_{MI}(r.A, r'.A') = \frac{I(r.A, r'.A')}{H(r.A, r'.A')} \tag{5.14}$$

where the name ST_{MI} comes from *STrengh by Mutual Information.*

Figure 5.4 shows two instances of foreign connections between tables r and r' through columns $r.A$ and $r'.A'$. In the left-side instance, all of the primary key values are instantiated by the foreign key column, while in the right-side instance, only half of the primary key values appear in the foreign key column. Obviously, the connections in the left instance is stronger than the one on the right. Using Equation 5.14, the strengths between columns $r.A$ and $r'.A'$ are 1.0 for the left instance and 0.88 for the right instance. Note that in calculating these values, the *virtual* output of *full outer join* between $R.A$ and $R'.A'$ is used as the join distribution of the two variables. This is to be consistent with the method in [68]. The reason is that the outer join contains values that do no match. For example, in the right instance of Figure 5.4, there will be $NULL$ values in column A' that match with values c and d in column A. That is, $(c, NULL)$ and $(d, NULL)$ each occur once in the joint distribution of $(r.A, r'.A')$. The reason is to penalize those joins with excessive numbers of unmatched values [68].

Note also that we do not need to perform the real joins in order to compute the strength score of an edge. The score can be computed by using the distribution of each involved column.

5.3.2.2 Edge Strength by Instantiation Fraction

Intuitively, edge strength can also be measured by the fraction of the join key values being instantiated. As mentioned in [67], the more fraction of primary key values are instantiated, the more important the edge is. However, in [67], the authors also assumed that by increasing the number of connections between two tables, the importance of the edge decreases. As we will show in the experiments, this assumption does not work for finding meaningful MJNSs. We propose the following measure, called *instantiation fraction* (IF), to quantify the importance of an edge based on the fractions of instantiated key values.

$$ST_{IF}(r.A, r'.A') = \frac{N_{r.A}^{inst}}{N_r} \times \frac{N_{r'.A'}^{inst}}{N_{r'}} \tag{5.15}$$

where $N_{r.A}^{inst}$ is the number of tuples in r that instantiates the edge between $r.A$ and $r'.A'$, and N_r is the total number of tuples of table r. $ST_{IF}(r.A, r'.A')$ of the the left and right instances in Figure 5.4 is equal to 1 and 0.5, respectively.

A modification of the above model is to consider the entropy of each column. By adding the entropies, the information content of each column is taken into account. This version of the IF measure, denoted as IF_Ent, is defined as

$$ST_{IF\_Ent}(r.A, r'.A') = \frac{N^{inst}_{r.A}}{N_r} \times \frac{r^{inst}_{r'.A'}}{N_{r'}} \times H_{norm}(r.A) \times H_{norm}(r'.A') \qquad (5.16)$$

where $H_{norm}(r.A)$ and $H_{norm}(r'.A')$ are normalized entropies of $r.A$ and $r'.A'$, respectively. The values of $\frac{N^{inst}_{r.A}}{N_r}$ and $\frac{N^{inst}_{r'.A'}}{N_{r'}}$ lie between zero and one. Thus, to make instantiation fractions and entropies equally important to the strength of the edge, normalized entropies in range $[0, 1]$ are used.

5.3.2.3 Ranking MJNSs

The same as the table importance scores, edge importance scores defined above can be computed offline. During a keyword search, to rank the MJNSs by edge importance, we compute a score for each MJNS using its average edge importance score, defined below:

$$Score_{edge}(M) = \frac{\sum_{\forall (r.A, r'.A') \in M} EdgeScore(r.A, r'.A')}{m} \qquad (5.17)$$

where M is an MJNS, $EdgeScore(r.A, r'.A')$ is an edge strength function and can be either ST_{MI}, ST_{IF} or ST_{IF_Ent}, and m is the number of edges in the MJNS M. The same as in ranking MJNSs based on the node importance, if one edge appears more than once, it is counted more than once. Since the edge strength scores are pre-computed, computing $Score_{edge}$ is fast and efficient.

5.3.3 The Hybrid Ranking Model

Our last approach to ranking MJNSs is to rank them based on the importance of both nodes and edges. In this *hybrid model,* we consider node importance when measuring the edge strength. The new edge strength is computed as follow:

$$EdgeScore(r.A, r'.A') \times \frac{TableScore(r) + TableScore(r')}{2} \tag{5.18}$$

If we use ST_{IF} for edge strength and the KE method for computing node importance, the hybrid formula becomes:

$$ST_{IF\_KE}(R.A, R'.A') = ST_{IF}(R.A, R'.A') \times \frac{KE(R) + KE(R')}{2} \tag{5.19}$$

Since the value of $ST_{IF}(R.A, R'.A')$ lies between zero and one, $KE(R)$ and $KE(R')$ should be normalized into range $[0, 1]$.

To rank the MJNSs, the score of an MJNS is computed using Equation 5.17 with ST_{IF_KE} as the $EdgeScore$ function.

5.3.4 Penalizing MJNSs

In all the MJNS ranking methods we described above, the average of node/edge importance scores is used to compute a score for an MJNS. Thus, the size of the MJNS (i.e., the number of nodes) is ignored in these ranking methods. Although we have shown that ranking MJNSs purely based on their size does not return satisfactory results in databses with large and complex schema, completely ignoring the size may not be a good strategy either. Generally, interpreting and understanding larger MJNSs is harder than interpreting smaller ones. Thus, for two MJNSs with similar average node or edge scores, the one with the smaller size should be ranked ahead of the larger one. To achieve this purpose, the final score of an MJNS M can be adjusted as[28]:

$$Score_{new}(M) = Score_{old}(M) \times \frac{1}{\log(treeSize)} \tag{5.20}$$

where $treeSize$ is the number of nodes in M, and $Score_{old}$ can be either $Score_{table}$ or $Score_{edge}$.

In the following section, we evaluate all the scoring methods with and without the penalization factor.

5.4 Experimental Evaluation

We evaluate the proposed ranking methods for finding the most meaningful/relevant MJNSs. All of the evaluated methods are implemented in Java. The experiments were performed on a performance test machine with an Intel(R) Core(TM) i7 2.80 GHz processor and 4GB of RAM.

[28]This formula was used in [52] to penalize large XML trees.

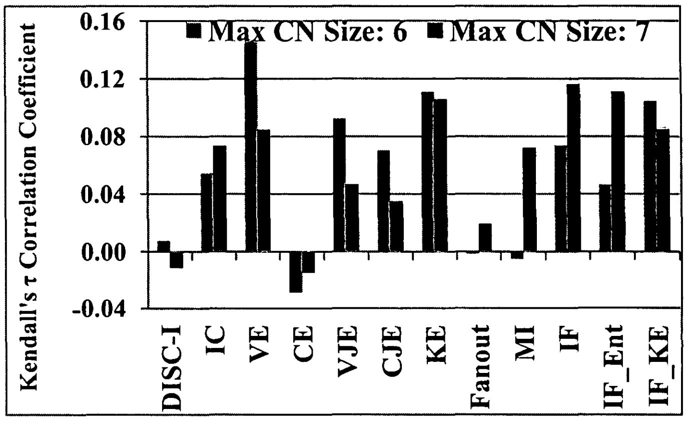


Figure 5.5: Results in Kendall's τ coefficient without penalizing larger MJNSs.

5.4.1 Dataset and Experimental Setup

The experiments are conducted over the TPC-E database. TPC provides a transaction log which we use to generate a gold standard for ranking MJNSs (Section 5.4.2). Since no active transaction is performed, table *Trade_Request* is not loaded with any data. Therefore, in our experiments, table *Trade_Request* is removed from the schema along with all of its foreign key connections. *EGen*, a package from TPC, is used for generating an instance of the database. The parameters of *EGen* are set to the same as those in [67, 68]. The number of customers, initial trade days and scale factor are set to 1000, 10 and 1000, respectively.

The focus of this work is ranking the MJNSs. As described in Section 5.2, a major input to our MJNS generator is the keyword roles . That is, the main input to our ranking methods is a set of relation names (i.e., query keyword roles). Table 5.1 lists ten sets of

Table 5.1: List of 10 sets of query keyword roles.

No.	Keyword Roles (Query)	Size
1	Customer, Company	2
2	Company, Broker	2
3	Customer, Broker	2
4	Customer, Customer	2
5	Customer, Company, Industry	3
6	Customer, Company, Trade_Type	3
7	Customer, Company, Broker	3
8	Customer, Company, Exchange	3
9	Customer, Company, Broker, Security	4
10	Customer, Company, Broker, Customer_Account	4

query keyword roles that we use as the input "queries" in our evaluation. For example, the first set of keyword roles specifies the names of two relations: $Customer$ and $Company$, which may result from keyword query $\{Joseph,\ Andersen\}$. As another example, the fourth "queries" is meant to find the relationships between two customers.

The MJNS generator also receives the maximum size of the MJNS as an input. Since TPC-E has a large schema with some dimensional tables, setting the maximum MJNS size to a value less than 6 results in generating MJNSs that are mostly connected through dimensional tables. On the other hand, setting the maximum value larger than 7 results in generating many MJNSs, whose results are hard to interpret by the users of the system. Thus, the maximum size (i.e., D_{max}) of the MJNSs is set to 6 and 7 in our experiments.

5.4.2 Gold Standard and Performance Measures

TPC provides 12 transactions along with the TPC-E benchmark. The set of transactions represents the usage of the database. Thus, they could be considered as a query log for the TPC-E benchmark [68]. We parsed the pseudo-codes of transaction and recorded the number of times a join between a pair of attributes $r.A$ and $r'.A'$ is performed. Let us denote this number by $n(r.A, r'.A')$. The importance of the connection between $r.A$ and $r'.A'$ is calculated as $\frac{n(R.A,R'.A')}{N_{total}}$, where N_{total} represents the total number of joins in all the pseudo-codes of all the transactions. Given an MJNS, its average edge importance score is calculated as its *gold standard* importance score. Given a set of MJNSs, their gold standard importance scores are used to rank the MJNSs to generate a *gold stardard ranking*.

Given a query, we use our MJNS generator to produce all the MJNSs for the query and then use each of the ranking methods (listed in Table 5.2) to rank the MJNSs. The ranked list from a method is compared with the gold standard ranking of these MJNSs. To see how close a ranked list produced by a ranking method is to the gold standard

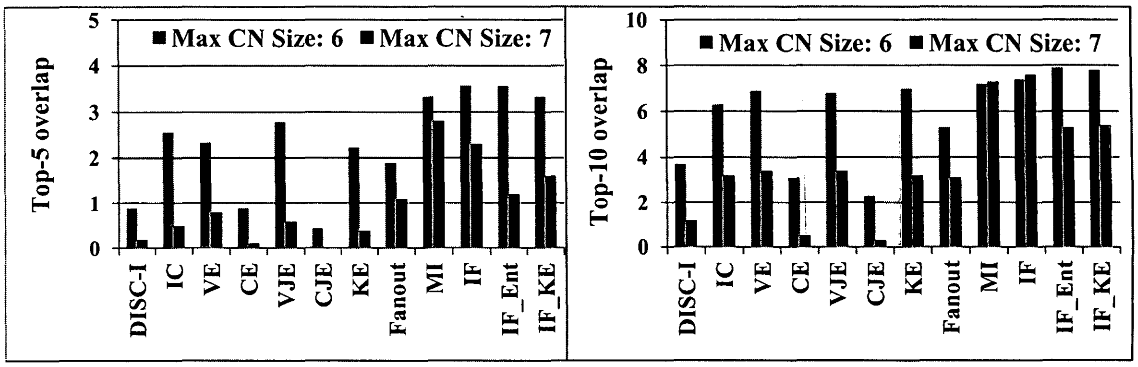


Figure 5.6: Results of the top-5 and top-10 overlaps without penalizing larger MJNSs.

	KE	DISC-I	MI	IF	IF_Ent
IF_KE	**0.018**	**0.004**	**0.003**	**0.025**	0.399
	0.000	**0.001**	**0.032**	**0.000**	0.903
	0.001	**0.000**	**0.037**	0.052	0.070
	0.000	**0.0001**	**0.012**	**0.016**	0.656
IF_Ent	0.153	**0.023**	**0.002**	**0.040**	
	0.021	**0.002**	0.071	**0.016**	
	0.193	**0.017**	1.000	**0.004**	
	0.0000	**0.000**	**0.001**	0.207	
IF	**0.001**	**0.000**	0.096		
	0.000	**0.000**	0.541		
	0.000	**0.000**	**0.002**		
	0.000	**0.000**	**0.002**		
MI	**0.000**	**0.000**			
	0.000	**0.000**			
	0.081	**0.012**			
	0.462	**0.009**			
DISC-I	0.443				
	0.021				
	0.096				
	0.033				

top-5 overlap without
top-10 overlap without
top-5 overlap with penalization
top-10 overlap with penalization

Figure 5.7: p-values of t-tests on the top-5 and top-10 overlap results. MNJS maximum size is 7.

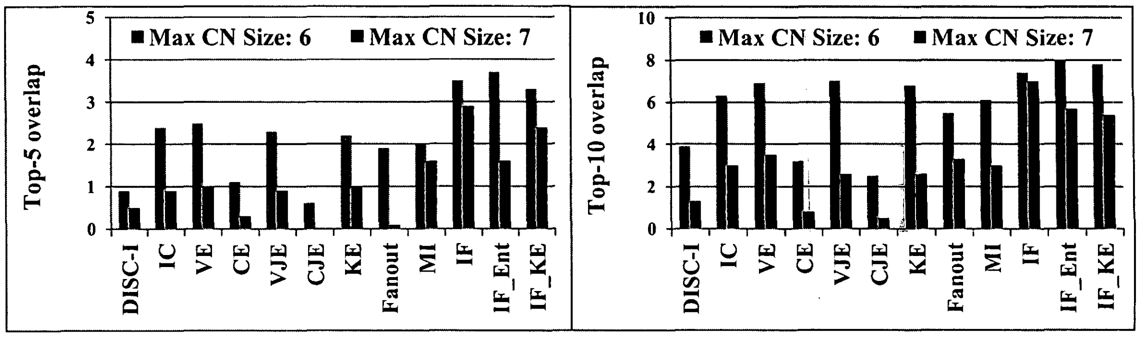


Figure 5.8: Results of the top-5 and top-10 with penalizing larger MJNSs.

ranking, we employ two measures used in the IR community [6]. The first one is a statistical measure for comparing two ranked lists, called Kendall's τ coefficeint [44]. It returns a value between +1 and -1, measuring the correlation between the two lists. If two lists are identical, it returns +1. If they are reversely ordered, it returns -1. Generally, a positive value means that two lists are related, and a negative value means that they are reversely related. The statistical tests are performed using SPSS 15.0 for Windows. The second measure is the size of the overlap between the top-k items in the two lists. We call this measure *top-k overlap*. We use 5 and 10 as the values of k. In our evaluation, the performance score of a method is measured by the average score over the 10 "queries" listed in Table 5.1.

5.4.3 Results of Ranking Methods without Size Penalization

We evaluated a total of 12 MJNS ranking methods. They are described in Table 5.2. In this section, we present the results of these ranking methods without penalizing larger

Table 5.2: List of methods for ranking MJNSs.

Method	Description
DISC-I	Size of MJNSs [27] (i.e. DISCOVER I)
IC	Node's importance, information content, Equation 5.6
VE	Node's importance, variable entropy [67]
CE	Node's importance, constant entropy [67]
VJE	Node's importance, variable joinable entropy [67]
CJE	Node's importance, constant joinable entropy [67]
KE	Node's importance, key entropy, Section 5.3.1.2
Fanout	Edge's importance, Definition 6 in [67]
MI	Edge's importance, mutual information [68]
IF	Edge's importance, instantiation fraction, Equation 5.15
IF_Ent	Edge's importance, IF & entropy, Equation 5.16
IF_KE	Hybrid method, IF & KE, Equation 5.19

MJNSs. The results are not compared with the IR based ranking methods (such as DISCOVER II [28]) since those methods are suitable for ranking the final answers but not the set of MJNSs.

The results of the 12 methods in terms of Kendall's τ rank correlation coefficient are presented in Figure 5.5. The results suggest that for the maximum MJNS size of 6, the ranking closest to the gold standard is achieved by the VE method. The best results for the maximum MJNS size of 7 are produced by IF, KE and IF_Ent. However, the correlation between VE and gold standard is not high when the maximum size of the MJNS is set to 7. This is not the case for IF and IF_Ent when the maximum size of the MJNS is set to 6. On the other hand, the correlation between KE and the gold standard is high for both of the maximum MJNS sizes. It achieves the best average and most stable result in terms of Kendall's τ coefficient.

We also observe that ranking MJNSs by their size does not work well (indicated by the result for *DISC-I*). Its Kendall's τ coefficient is close to zero when the maximum MJNS size is 6, and it is negative when the size is 7. Therefore, ranking the results solely based on the number of nodes in the MJNS does not produce satisfactory results. Since only the size has been used to rank joining networks in previous methods, our proposed ranking methods outperform previous ones.

The results also show that some measures for node/edge importance (such as CE and $Fanout$) produce poor results as well. The measures that we present in this chapter are much better than those measures.

The results of the top-5 and top-10 evaluations are presented in Figure 5.6. By increasing the maximum size of the MJNS from 6 to 7, the top-k overlap with the gold standard decreases. This result is expected since by increasing the maximum size, the number of generated MJNSs increases. As the value of k is a constant (i.e. it is set to 5 or 10) and there are more items (MJNSs) in the list, the chance for overlap in top-k lists decreases.

Generally, the methods that rank MJNSs based on the edge importance work better than the ones based on the node importance. The difference between the two types of methods becomes greater when the maximum MJNS size is 7. In order to see whether different methods are significantly different from each other, we run the t-test. The p-values of the t-tests are presented in Figure 5.7 for the maximum MJNS size of 7. The results for the maximum size of 6 follows the same trend. Since the edge-based ranking methods work better than the node-based ranking methods, the the t-test results for the node-based ranking methods are not presented.

Among the edge based methods, IF is the best in most of the cases. MI is the second best in general according to Figure 5.6. The t-test results show that their performances are not significantly different (with p-values of 0.096 and 0.541) The other edge based method IF_Ent performs the best when the maximum MJNS size is 6, but not well when the maximum size is 7. Similar observation is found for the hybrid method IF_KE. Thus, these two methods are not stable compared to IF and MI.

Looking at the results for the node based methods in Figure 5.6, we observed that the results of VE and KE are very similar. Thus, using KE is better than using VE due to its faster computation of information content, as discussed in Section 5.3.1.2.

5.4.4 The Effect of Penalizing Larger MJNSs

The results of the top-5 and top-10 evaluations with penalizing larger MJNSs are presented in Figure 5.8 and some of its t-test results are shown in Figure 5.7. Note that the penalization technique (Equation 5.20) is applied to the computation of gold standard as well.

Comparing the results in Figures 5.6 and 5.8, we observe that the MI method is significantly negatively affected by the use of the size penalization technique, while the performance of other methods remain pretty much the same.

This set of results also suggests that the edge importance based methods are better than the node based methods, and that the edge based method IF is a stable method with the best overall performance. Again, the edge based IF_Ent method and hybrid method IF_KE have the best or close to best performance for the maximum MJNS size of 6, but their performance descreases significantly for the maximum MJNS size of 7. In addition, both Figures 5.6 and 5.8 suggest that the size based method (*DISC-I*) has the worst performance. The t-test results show that it is significantly worse than other methods.

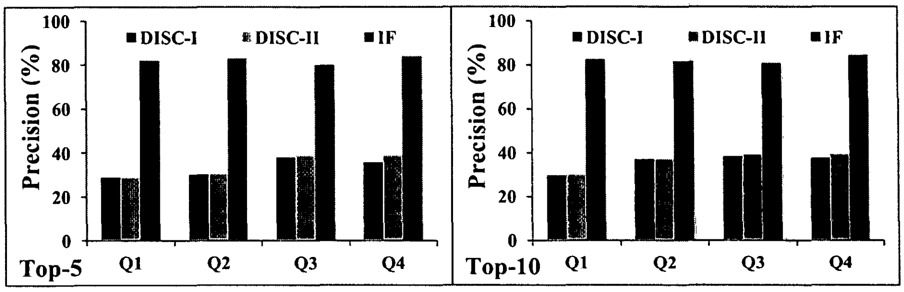


Figure 5.9: Top-5 and Top-10 precision of answers.

5.4.5 Relevance Evaluation of Final Answers by a User Study

To see how effective ranking of MJNSs impacts the *final answers* of the keyword search, we compare the top-k final answers from our keyword search method that uses the IF method for ranking MJNSs with the final answers produced by the method that ranks the final answers based on the number of joins (Discover I [27]) and the one based on the IR techniques (Discover II [28]) in terms of how relevant their answers are to the query. A common metric of relevance used in information retrieval is top-k precision, defined as the percentage of the answers in the top-k answers that are relevant to the query. To evaluate the top-k precision of the methods, we conduct a user study. We use 4 sets of keyword related to the first, second, fifth and sixth queries in Table 5.1 to evaluate the search results by human user. For example, the first query is "Jacob Insurance" in which "Jacob" is associated with the *customer* table and "Insurance" is associated with the *company* table (i.e., their roles are *customer* and *company*, respectively). In the experiment, top-5 and top-10 answers are produced for each query by each search method.

We ask 8 graduate students in computer science, information technology, and mathematics to judge the relevancy of the answers. A user assigns a score between 0 and 1 to each final answer, where 1 means completely relevant and 0 means completely irrelevant to the query. This score may vary among the users. Thus, the average of the relevancy scores from the 8 users is used as the final relevancy score for an answer. The top-k precision is computed as the average relevancy score of the top-k answers.

The top-5 and top-10 precisions for each query are presented in Figure 5.9. Clearly, the IF method which ranks the answers based on the edge strength between the associated entities achieves much better precisions than DISC-I and DISC-II in all the queries for both k values. The reason for DISC-I and DISC-II to have a lower precision is that in most of the cases, the tuples of the final answers are connected together by the dimension tables (e.g. *status_type*) and fact tables are not involved. Thus, most of the users find the answers not so meaningful and assign them lower scores.

5.5 Conclusion and Future Work

Our goal was to improve relevance scoring of answers. Based on their networks (join trees) in light of larger and more complex database schema. To this end, we propose a series of measures, and algorithms to compute them, using different approaches to capture the intended semantic of queries based on the importance of the connections involved in the networks. As the information for ranking the MJNSs can be pre-built as an index, the ranking model works fast. Extensive experiments and a user study on a large and complex TPC-E schema establish that the proposed methods are able to capture well the intended semantics behind queries.

While our methods prove to be effective, there is much room for further research and improvement. Keyword search evaluation can be optimized in a number of ways. We can apply previous techniques as pipelining in DISCOVER I [27] to our approach to improve the efficiency. Multi-query optimization [62] over the SQL queries generated for the MJNSs could exploit commonalities among them to speed up greatly evaluation. Furthermore, while in this work we sought to demonstrate how effective deriving relevance of the "nodes" and "edges" of the database schema could be based on just the schema and data, by no means are we advocating that auxiliary information cannot improve it. We would like to explore the use of Linked Data[29] and WordNet[30].

[29]http://linkeddata.org

[30]http://wordnet.princeton.edu

6 Finding Affordable and Collaborative Teams from a Network of Experts

Given an expert network, in which a node represents an expert that has a cost for using the expert service and an edge represents the communication cost between the two corresponding experts, we tackle the problem of finding a team of experts that covers a set of required skills and also minimizes the communication cost as well as the personnel cost of the team. Since two costs need to be minimized, this is a bicriteria optimization problem. We show that the problem of minimizing these objectives is NP-complete. We use two approaches to solve this bicriteria optimization problem. In the first approach, we propose several (α, β)-approximation algorithms that receive a budget on one objective and minimizes the other objective within the budget with guaranteed performance bounds. In the second approach, an approximation algorithm is proposed to find a set of Pareto-optimal teams, in which each team is not dominated by other feasible teams in terms of the personnel and communication costs. The proposed approximation algorithms have provable performance bounds. Extensive experiments on real datasets demonstrate the effectiveness and scalability of the proposed algorithms.

6.1 Introduction

An expert network contains a group of professionals who can provide specialized information and service. With the widespread use of the Internet, online expert networks have become popular where more and more businesses seek subject matter experts to complete a task or project. There are many expert network providers, such as Gerson Lehrman Group[31] and the Network of Experts[32]. In such networks, an expert is described by their areas of expertise, education background, location, etc. In addition, an expert can specify his/her consulting rate.

We consider the problem of finding a team of experts from such a network to complete a project. A team must possess a set of required skills in order to complete the tasks of the project. In addition, a project is usually constrained by the budgeted amount of money available for the project. Different experts may incur different fees for conducting the activities of the project. It is desirable to find a team of experts whose total cost is minimized. Furthermore, the success of a project greatly depends on how well the team members of the project communicate and collaborate with each other. Experts located in different countries may not communicate as easily as the ones living in the same city

[31]http://www.glgresearch.com/

[32]http://www.networkofexperts.com/

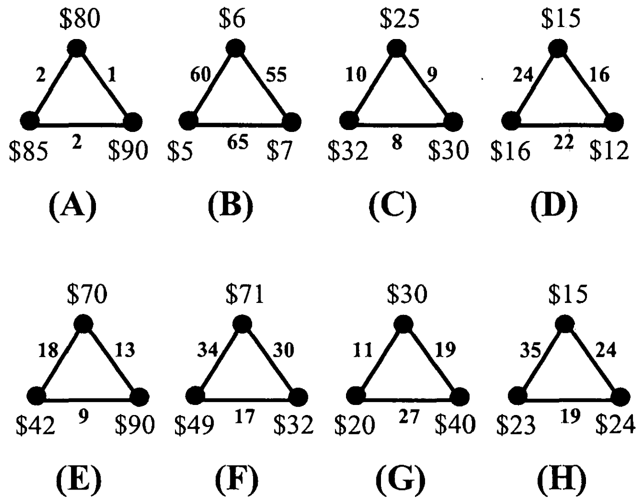


Figure 6.1: An example of all feasible teams.

when face-to-face meetings are required. Thus, it is important to minimize the communication cost among the experts. This turns the problem into a bicriteria optimization problem.

The problem of finding a team of experts from a network which minimizes the communication cost has been tackled in [34, 48]. However, previous works in this domain did not consider the budget of the project nor the fees that may be associated with the experts. In the real world, an expert needs to be paid for his/her service, and it is preferred that the personnel cost of a project is minimized or under a budget. Only minimizing the communication cost may result in a team with high personnel cost. For example, assume that all the feasible teams of experts for a project are shown in Figure 6.1. Each team has three experts that together cover all of the required skills. Assume that the communication cost of a team is calculated using the sum of distances between experts in the team. The communication costs of teams A, B, C, D, E, F, G and H are 5, 180, 27, 62, 40, 81, 57 and 78, respectively. The personnel costs of these teams are \$255, \$18, \$87, \$43, \$202, \$152, \$90 and \$62, respectively. Figure 6.2 shows these eight teams on a diagram. If one wants to minimize only the communication cost, team A is the best. However, its personnel cost is the highest. On the other hand, if one wants to minimize the personnel cost, team B is the best choice but has the highest communication cost. If one wants to have a team in which the members collaborate most effectively and at the same time the personnel cost is the lowest or reasonable, there is not an obvious best choice.

Clearly, there is a trade-off between the personnel cost and the communication cost. A good method should either allow the user to provide a tolerance limit on one of the objectives and produce the best answer on the other objective, or provide a set of best trade-off solutions for the user to choose from. For example, in the above example, if a budget is given on the personnel cost as \$300, the best team is A. However, for budgets of \$100 or \$50, the best team is C or D respectively. Alternatively, if the budget is not available, we can provide users with a set of solutions that are not worse than any other solutions on both objectives. These solutions are called *Pareto-optimal solutions* [33]. Teams A, B, C and D in Figures 6.1 and 6.2 are Pareto-optimal solutions since none of them is worse than other teams on both costs. However, the remaining teams (E, F, G and H) are worse than at least one Pareto solution.

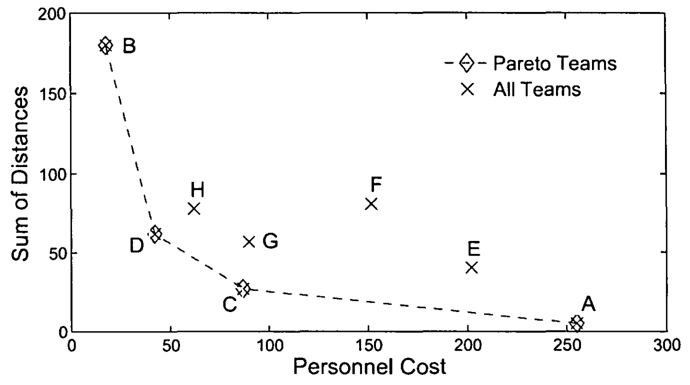


Figure 6.2: Feasible and Pareto optimal solutions.

The contributions of this chapter are summarized as follows.

1. We define the problem of finding an affordable and collaborative team in an expert network. We use two functions to measure the communication cost and one function to measure the personnel cost of a team.
2. We show the problem we tackle is NP-complete and propose a series of new (α, β)-approximation algorithms (to be defined later) to solve the bi-objective team formation problem, which optimizes one objective given a budget on the other objective with proved performance bounds.
3. For finding a set of Pareto-optimal solutions, a new approximation algorithm is proposed that can find solutions with guaranteed performance bounds.
4. The effectiveness and efficiency of the proposed algorithms are evaluated extensively on two large real datasets.

The chapter is organized as follows. In Section 6.2, we present problem statement. In

Section 6.3, we present a series of algorithms for finding a team of experts with bounded budget. An algorithm for finding Pareto-optimal teams is presented in Section 6.4. In Section 6.5, experiments on real datasets are presented. In Section 6.6, we conclude.

6.2 Problem Statement

Let $C = \{c_1, c_2, \ldots, c_m\}$ denote a set of m experts, and $S = \{s_1, s_2, \ldots, s_r\}$ denote a set of r skills. Each expert c_i has a set of skills, denoted as $Q(c_i)$, and $Q(c_i) \subseteq S$. If $s_j \in Q(c_i)$, expert c_i has skill s_j. In addition, a subset of experts $C' \subseteq C$ have skill s_j if at least one of them has s_j. For each skill s_j, the set of all experts having skill s_j is denoted as $C(s_j) = \{c_i | s_j \in Q(c_i)\}$. A project $P \subseteq S$ is defined as a set of skills required to complete the project. A subset of experts $C' \subseteq C$ is said to *cover* a project P if $\forall s_j \in P \, \exists \, c_i \in C', s_j \in Q(c_i)$.

The experts are connected together in a network, modeled as an undirected and weighted graph (G). Each node in G represents an expert in C. Below, terms node and expert are used interchangeably. Each node in the graph is associated with a cost representing the amount of money he/she is paid for completing a project. The cost of an expert c_i is denoted as $t(c_i)$. Two experts may be connected by an edge in the graph. The weight on an edge represents the communication cost between the two experts. The lower the weight, the more easily the two experts can collaborate or communicate, and the lower the communication cost between them. The communication cost between two experts can be defined according to the application need. For example, it can be defined as the geometric distance between two experts, which is a good communication cost measure when face-to-face meetings are needed in the project. The communication cost can also be defined by the collaboration ability or familiarity between the two experts. In this case, two nodes are connected by an edge if the experts have communicated or collaborated before, and the weight of the edge represents the strength of the relationships between the two experts. Such relationships can be obtained from social networks (such as LinkedIn), scientific collaboration networks (such as DBLP), or other sources.

Definition 10 ***(Team of Experts)*** *Given a set C of experts and a project P that requires skills $s_1, s_2, \ldots,$ and s_n, a* team of experts *for P is a set of n skill-expert pairs: $\{\langle s_1, c_{s_1}\rangle, \langle s_2, c_{s_2}\rangle, \ldots, \langle s_n, c_{s_n}\rangle\}$, where c_{s_j} is an expert in C having skill s_j for $j = 1, \ldots, n$. A skill-expert pair $\langle s_i, c_{s_i}\rangle$ means that expert c_{s_i} is responsible for skill s_i in the project.*

Note that an expert in a team may be responsible for more than one required skill, that is, c_{s_i} can be the same as c_{s_j} for $i \neq j$. To evaluate the **communication cost of a team**, we define the *sum of distances* or *diameter* of a team, which has been used in [34] and [48] respectively.

Definition 11 ***(Sum of Distances)*** *Given a team T of experts from a graph G for a* *project:* $\{\langle s_1, c_{s_1}\rangle, \langle s_2, c_{s_2}\rangle, \ldots, \langle s_n, c_{s_n}\rangle\}$, *the* sum of distances *of T is defined as*

$$sumDistance = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d(c_{s_i}, c_{s_j})$$

where $d(c_{s_i}, c_{s_j})$ is the sum of weights on the shortest path between c_{s_i} and c_{s_j} in G, i.e., the shortest distance between c_{s_i} and c_{s_j}.

The use of the shortest distance in the above definition implies that the communication cost between two experts can be estimated by using their communication costs with a third expert, especially when the two experts are not directly connected. This can be easily justified when, say, travel distances are used as edge weights in the graph. In case familiarity is used to weigh an edge, the use of the shortest distance implies that two people who have not collaborated before can collaborate if they have collaborated with a third person. This can be justified by Newman's finding on scientific networks [56]: two people are much more likely to collaborate if they have both worked with a third person.

Definition 12 ***(Diameter)*** *Given a graph G and a team of experts T consisting of some experts in G, the diameter of team T is the largest shortest distance between any two experts of T in G.*

Definition 13 ***(Personnel Cost)*** *Let the set of experts in a team T be $\{c_1, c_2, \ldots, c_q\}$. The* personnel cost *of T is defined as:*

$$PCost(T) = \sum_{i=1}^{q} t(c_i)$$

Problem 5 *(Affordable and Collaborative Team Formation) Given a project P and a graph G representing a network of experts, the problem of affordable and collaborative team formation is to find a team of experts T for P from G so that the communication cost of T, defined as either the sum of distances or diameter of T, and the personnel cost of T, defined as $PCost$, are minimized.*

Clearly, Problem 5 is a bi-criteria optimization problem. It has been proved that finding a team T of experts in a graph while minimizing the sum of distances or diameter of T is an NP-complete problem [34, 48]. Below we show that minimizing $PCost$ is also NP-complete.

Theorem 10 *Finding a team of experts in a graph G to cover a set of skills while minimizing $PCost$ is NP-complete.*

Proof

Given a set of m elements (called universe) and n sets whose union is the universe, the *set cover* problem is to identify the *smallest* number of sets whose union still contains all elements in the universe. *Weighted set cover* is a variant of *set cover* in which each set has a cost associated with it and the objective is to pick sets that cover the universe with the minimum total cost. Consider the set of required skills in our problem as the universe and the experts in G as the sets whose union is the universe. Each expert is associated with a cost. Then, finding a team of experts while minimizing $PCost$ is equivalent to solving *weighted set cover* problem. Since the *weighted set cover* problem is NP-complete [64], finding a team of experts while minimizing $PCost$ is NP-complete. □

Since minimizing the *sum of distances*, *diameter* or *personnel cost* is NP-complete, solving Problem 5 is NP-complete. Thus, we have to rely on approximation algorithms for solving this problem. Many (if not most) methods for solving bi-criteria optimization problems combine two objectives into a single one by using a weighted sum of two functions [39]. If the weight value is not chosen correctly, the result may not be reliable. Also, such methods are usually very sensitive to small changes in weight values [25]. In this chapter we use two other approaches to solve this bicriteria problem. In the first approach, a budget value (bound) is specified on one objective and the other objective is optimized under this budget. In the second approach, the set of Pareto optimal answers [58] are found, which represent optimal trade-offs between the two objectives. Below in Section 6.3 we propose several (α, β)-approximation algorithms for our problem, which take the first approach, and then in Section 6.4 we propose an algorithm for finding a set of Pareto-optimal solutions.

6.3 Finding a Team of Experts with Bounded Budget

We first define the concept of (α, β)-approximation algorithm, and then propose a few (α, β)-approximation algorithms for solving our problem.

Definition 14 *An (α, β)-approximation algorithm for an (A, B)-bicriteria problem is defined as a polynomial time algorithm that produces an answer in which the value of the first objective (A) is at most α times a budget, and the value of the second objective (B), is at most β times the minimum for any answer that is within the budget on A.*

6.3.1 Finding a Team of Experts with a Budget on the Communication Cost

In this subsection, we propose two algorithms for solving Problem 5. Both algorithms receive a budget on the communication cost of the team and minimize the personnel cost. The first algorithm uses the *diameter* and the second algorithm uses the *sum of distances* as the communication cost function.

6.3.1.1 Budget on the Diameter

The algorithm takes a budget on the diameter and minimizes the $PCost$ function. It is a $(2, \log n)$-approximation algorithm where n is the number of required skills of the project. The *diameter* budget is specified as D. The $(2, \log n)$-approximation means that the answer produced by the algorithm has a diameter at most twice the budget (D) and its $PCost$ value is at most $\log n$ times the cost of the minimum $PCost$ for any answer within the D diameter.

The idea of the first algorithm is as follows. It first collects the experts with the rarest required skill s_{rare} (i.e., the required skill with the least number of experts). Then, for each expert cr_i that possesses s_{rare}, all of the experts having other required skills than s_{rare} and within D distance from cr_i are collected into a set V. A candidate team based on cr_i is then formed by including cr_i and selecting experts from V to cover all the required skills. The expert selection is a greedy procedure that iteratively selects an expert c_k^v that maximizes the ratio of the number of currently uncovered required skills covered by c_k^v to the cost of c_k^v until all the required skills are covered by the team. That is, the quality of an expert is evaluated using the number of uncovered skills per unit cost. The algorithm outputs the team that has the smallest personnel cost among all the candidate teams built around the experts with s_{rare}. If more than one team has the least cost, the one with the lowest diameter is chosen. The reason for starting a team with an expert with s_{rare} is to keep the number of candidate teams as small as possible.

The pseudo code of this approximation algorithm for solving the $(diameter, PCost)$ problem is presented in Algorithm 13. The algorithm first obtains the set $C(s_i)$ of experts having required skill s_i for each i. This can be done quickly by using a pre-built inverted index that maps a skill to its experts. In the code, $d(cr_i, c_j)$ is the shortest distance between experts cr_i and c_j, which can be efficiently obtained by consulting a pre-built index. Using a pre-built index to obtain the shortest distance between nodes has been used in other graph search methods such as the ones in [34, 51, 60]. The time complexity of Algorithm 13 is $O(|C(s_{rare})| \times (|C| + |V| \times n))$ where $|C(s_{rare})|$ is the number of experts with the rarest required skill, $|C|$ is the number of experts with other required

Algorithm 13 $(2, \log n)$-approximation algorithm for solving $(diameter, PCost)$ problem

Input: graph G, project $P = \{s_1, s_2, \ldots, s_n\}$, and budget D on the diameter.
Output: the best team and its personnel cost

```
1:  for $i \leftarrow 1$ to $n$ do
2:      $C(s_i) \leftarrow$ the set of experts with $s_i$
3:  $s_{rare} \leftarrow \arg\min |C(s_i)|, 1 \leq i \leq n$
4:  $C \leftarrow \bigcup_{i=1\&i\neq rare}^{n} C(s_i)$
5:  $bestTeam \leftarrow \emptyset$
6:  $leastCost \leftarrow \infty$
7:  for each expert $cr_i$ in $C(s_{rare})$ do
8:      $requiredSkill \leftarrow P - Q(cr_i)$
9:      $V \leftarrow \emptyset$
10:     for each expert $c_j$ in $C$ do
11:         if $\langle d(cr_i, c_j) \leq D\rangle$ & $\langle Q(c_j) \cap requiredSkill \neq \emptyset\rangle$ then
12:             add $c_j$ to $V$
13:     $\{c_1^v, \ldots, c_q^v\} \leftarrow V$
14:     $skillV \leftarrow \bigcup_{i=1}^{q} Q(c_i^v)$
15:     if $requiredSkill \subseteq skillV$ then
16:         $team \leftarrow \{\langle q_1, cr_i\rangle, \langle q_2, cr_i\rangle, ..., \langle q_k, cr_i\rangle\}$ where $q_1, q_2, \ldots q_k$ are the required
            skills that $cr_i$ has, i.e., $\{q_1, q_2, \ldots q_k\} = P \cap Q(cr_i)$
17:         $cost \leftarrow t(cr_i)$
18:         while $requiredSkill \neq \emptyset$ do
19:             Select $k$ s.t. $\frac{|requiredSkill \cap Q(c_k^v)|}{t(c_k^v)}$ is maximized
20:             $team \leftarrow team \cup \{\langle q_1, c_k^v\rangle, \langle q_2, c_k^v\rangle, ..., \langle q_k, c_k^v\rangle\}$ where $\{q_1, q_2, \ldots q_k\} =$
                $requiredSkill \cap Q(c_k^v)$
21:             $cost \leftarrow cost + t(c_k^v)$
22:             $requiredSkill \leftarrow requiredSkill - Q(c_k^v)$
23:         if $cost < leastCost$ then
24:             $bestTeam \leftarrow team$
25:             $leastCost \leftarrow cost$
26:         else
27:             if $cost = leastCost$ and $team.diameter < bestTeam.diameter$ then
28:                 $bestTeam \leftarrow team$
29: return $bestTeam, leastCost$
```

skills, $|V|$ is the number of experts within D distance to a member of $C(s_{rare})$ and n is the number of required skills. Since the number of experts with the required skills is at most the number of all experts in G, i.e. m, the run time of the algorithm in the worst case is $O(m^2 \times n)$. However, in practice, $|C(s_{rare})|$, $|C|$ and $|V|$ are much less than m.

Theorem 11 *Algorithm 13 is a* $(2, \log n)$ *approximation algorithm for solving* $(diameter$*,* $PCost)$ *problem where* n *is the number of required skills.*

Proof

The algorithm forms a team by using an expert cr_i with the rarest required skill and the experts within D distance from cr_i, which means that all the experts in the team are within D distance from cr_i. Assume that the two nodes that have the largest shortest distance among all pairs of nodes are n_1 and n_2. Since the shortest distance satisfies the triangle inequality, we have:

$$dist(cr_i, n_1) + dist(cr_i, n_2) \geq dist(n_1, n_2)$$

Since $dist(cr_i, n_1) \leq D$ and $dist(cr_i, n_2) \leq D$, then $dist(n_1, n_2) \leq 2 \times D$. Thus, the diameter of the team is at most $2 \times D$.

Every team should have a member cr_i with s_{rare}. Around cr_i, all of the experts with other required skills and within D distance are considered to be added into a team . Thus, all of the teams containing s_{rare} and with diameter less than or equal to D are considered by the greedy procedure in Lines 18-22. The greedy procedure tries to find the team with the minimum personnel cost by selecting experts using a heuristic that is the same as the one in the greedy algorithm for solving the weighted set cover problem [64]. It has been shown that the algorithm has a $\log n$ approximation ratio [64]. Thus, the personnel cost of the team produced by Algorithm 13 is at most $\log n$ times the cost of the optimal team with diameter at most D. □

6.3.1.2 Budget on the Sum of Distances

The algorithm takes a budget on the sum of distances and minimizes the $PCost$ function. It is a $(n, \log n)$-approximation algorithm where n is the number of required skills of the project. The pseudo code of this approximation algorithm for solving the $(sumDistance,$ $PCost)$ problem is presented in Algorithm 14. The algorithm is similar to Algorithm 13, but has two major differences. First, instead of using only the rarest skill holders, this algorithm uses all the required skill holders as the seed of a candidate team. Second, for each seed (cr_i), this algorithm only considers adding its neighbors within the radius of $\frac{SD}{n-1}$ into the team, where SD is the *sumDistance* budget. The time complexity of this algorithm is $O(m^2 \times n^2)$ where n is the number of required skills and m is the number of experts in G.

Theorem 12 *Algorithm 14 is a $(n, \log n)$-approximation algorithm for solving the problem of $(sumDistance, PCost)$ where n is the number of required skills.*

Proof

The algorithm forms a team by using an expert cr_i which has at least one required skill and the experts within $\frac{SD}{n-1}$ distance from cr_i[33]. It means that all the experts in the team are within $\frac{SD}{n-1}$ distance from cr_i. Assume that the two nodes that have the largest shortest distance among all pairs of nodes are n_1 and n_2. Since the shortest distance satisfies the triangle inequality, we have:

$$dist(cr_i, n_1) + dist(cr_i, n_2) \geq dist(n_1, n_2)$$

Since $dist(cr_i, n_1) \leq \frac{SD}{n-1}$ and $dist(cr_i, n_2) \leq \frac{SD}{n-1}$, then $dist(n_1, n_2) \leq \frac{2 \times SD}{n-1}$. Thus, in the worst case, the maximum distance between each pair of nodes is at most $2 \times \frac{SD}{n-1}$. Since the sum of distances function sums up $\frac{(n) \times (n-1)}{2}$ pairwise distances, the maximum possible value for the sum of distances is as follows:

$$sumDistance \leq \frac{(n) \times (n-1)}{2} \times \frac{2 \times SD}{(n-1)} = n \times SD$$

Therefore, the sum of distances is at most $n \times SD$. The rest of the proof is similar to Theorem 11. $\square$

Note that the two approximation ratios (n and $\log n$) only occur in the worst case. We will show in the experiments that the real ratios are much smaller than n and $\log n$ in our experiments on real data sets.

[33]Note that if the distance is set to a value smaller than $\frac{SD}{n-1}$, some of the teams with $sumDistance$ less than or equal to SD might be missed. On the other hand, if it is set to a value larger than $\frac{SD}{n-1}$, the approximation ratio is increased.

Algorithm 14 $(n, \log n)$-approximation algorithm for solving $(sumDistance, PCost)$ problem

Input: graph G, project $P = \{s_1, s_2, \ldots, s_n\}$, and budget SD on the sum of distances.
Output: the best team and its personnel cost

```
1: for $i \leftarrow 1$ to $n$ do
2:     $C(s_i) \leftarrow$ the set of experts with $s_i$
3: $C \leftarrow \bigcup_{i=1}^{n} C(s_i)$
4: $bestTeam \leftarrow \emptyset$
5: $leastCost \leftarrow \infty$
6: for $h \leftarrow 1$ to $n$ do
7:     for each expert $cr_i$ in $C(s_h)$ do
8:         $requiredSkill \leftarrow P - Q(cr_i)$
9:         $V \leftarrow \emptyset$
10:        for each expert $c_j$ in $C$ do
11:            if $\langle d(cr_i, c_j) \leq \frac{SD}{n-1} \rangle$ & $\langle Q(c_j) \cap requiredSkill \neq \emptyset \rangle$ then
12:                $V \leftarrow V \cup \{c_j\}$
13:        $\{c_1^v, \ldots, c_q^v\} \leftarrow V$
14:        $skillV \leftarrow \bigcup_{i=1}^{q} Q(c_i^v)$
15:        if $requiredSkill \subseteq skillV$ then
16:            $team \leftarrow \{\langle q_1, cr_i \rangle, \langle q_2, cr_i \rangle, \ldots, \langle q_k, cr_i \rangle\}$ where $q_1, q_2, \ldots q_k$ are the required
               skills that $cr_i$ has, i.e., $\{q_1, q_2, \ldots q_k\} = P \cap Q(cr_i)$
17:            $cost \leftarrow t(cr_i)$
18:            while $requiredSkill \neq \emptyset$ do
19:                Select $k$ s.t. $\frac{|requiredSkill \cap Q(c_k^v)|}{t(c_k^v)}$ is maximized
20:                $team \leftarrow team \cup \{\langle q_1, c_k^v \rangle, \langle q_2, c_k^v \rangle, \ldots, \langle q_k, c_k^v \rangle\}$ where $\{q_1, q_2, \ldots q_k\}$ =
                   $requiredSkill \cap Q(c_k^v)$
21:                $cost \leftarrow cost + t(c_k^v)$
22:                $requiredSkill \leftarrow requiredSkill - Q(c_k^v)$
23:            if $cost < leastCost$ then
24:                $bestTeam \leftarrow team$
25:                $leastCost \leftarrow cost$
26:            else
27:                if $cost = leastCost$ & $team.sumDistance < bestTeam.sumDistance$
                   then
28:                    $bestTeam \leftarrow team$
29: return $bestTeam$, $leastCost$
```

6.3.2 Finding a Team of Experts with a Budget on the Personnel Cost

In practice, there is often a budget on the personnel cost and the goal is to minimize the communication cost within the personnel budget. Below we propose approximation algorithms that minimize the communication cost under a personnel budget for solving the $(PCost, diameter)$ and $(PCost, sumDistance)$ problems.

According to [54], bicriteria problems are generally hard when the two criteria are *hostile* with respect to each other, meaning that the optimization of one criterion conflicts with the optimization of the other criterion. Two minimization objectives in our problem are hostile because the minimum value of one objective is monotonically non-decreasing as the bound (budget) on the value of the other objective is decreased. This can be proved as follows. By decreasing the budget on the communication cost, the set of possible teams under the new budget becomes a subset of possible teams before the budget is decreased. Since the optimal team in a subset cannot be better than the optimal team in the superset, the personnel cost of the optimal team with the lower budget on the communication cost cannot be lower than the personnel cost of the optimal team with a higher budget on the communication cost.

In [54], a generic procedure was proposed that uses an (α, β)-approximation algorithm for the (A, B) problem to solve the (B, A) problem in polynomial time and with the approximation ratio of (β, α). The procedure applies to only hostile bicriteria problems. Since the two criteria in our problem are hostile and the algorithms we proposed in the last subsection for the ($diameter$, $PCost$) and ($sumDistance$, $PCost$) problems are (α, β)-approximation algorithms, we can adapt the generic procedure in [54] to derive (β, α)-approximation algorithms to solve the $(PCost, diameter)$ and $(PCost, sumDistance)$, respectively. Note that this is the first time that this generic procedure is adapted to find teams of experts.

The $(\log n, 2)$-algorithm for solving the $(PCost, diameter)$ problem is presented in Algorithm 15. The basic idea of the algorithm is to conduct a **binary search** over the range of diameter values for a diameter value that is as small as possible and at the same time the $PCost$ value of the team is not over the budget B. The algorithm starts with the diameter of the input graph G, and stores it in D_{prev}. It calls Algorithm 13 with D_{prev} as the diameter budget to find the best (approximate) team that minimizes $PCost$. If the $PCost$ value of the found team is greater than the input budget B on $PCost$, no solution exists because if the diameter is lowered, the minimal $PCost$ value will not decrease (due to the hostile relationship between the two objectives). But if the $PCost$ value of the team found by Algorithm 13 is less than B, then there may exist teams with lower diameters and also under the $PCost$ budget. Thus, the algorithm continues and checks the diameter which is half of the previous value in D_{prev} by calling Algorithm 13 with this new diameter value (stored in D_{new}) as the diameter budget. If the $PCost$ value of the answer returned by Algorithm 13 is more than B, there is no solution that

Algorithm 15 $(\log n, 2)$-approximation algorithm for solving $(PCost, diameter)$ problem

Input: graph G, project $P = \{s_1, s_2, \ldots, s_n\}$, budget B on $PCost$, and precision threshold ϵ.

Output: the best team and its diameter

```
1: $MaxDiameter \leftarrow \max dist(c_i, c_j), 1 \leq i, j \leq m$ and $m$ is the number of nodes in $G$
2: $D_{prev} \leftarrow MaxDiameter$
3: $\langle team_{prev}, PCost_{prev} \rangle \leftarrow$ **Algorithm13**($G$, $P$, $D_{prev}$)
4: **if** $PCost_{prev} > B$ **then**
5:     **return** $\emptyset, \infty$
6: $D_{lower} \leftarrow 0$
7: **while** $(D_{prev} - D_{lower}) > \epsilon$ **do**
8:     $D_{new} \leftarrow \frac{D_{prev}+D_{lower}}{2}$
9:     $\langle team_{new}, PCost_{new} \rangle \leftarrow$ **Algorithm13**($G$, $P$, $D_{new}$)
10:    **if** $team_{new} \neq \emptyset$ and $PCost_{new} \leq B$ **then**
11:        $D_{prev} \leftarrow D_{new}$
12:        $\langle team_{prev}, PCost_{prev} \rangle \leftarrow \langle team_{new}, PCost_{new} \rangle$
13:    **else**
14:        $D_{lower} \leftarrow D_{new}$
15: **return** $team_{prev}, D_{prev}$
```

has a diameter under or equal to D_{new} (due to the hostile relationship between diameter and $PCost$). The algorithm then increases the value in D_{new} to $(D_{prev} + D_{new})/2$ and calls Algorithm 13 again with the new value in D_{new} to continue the search. However, if the $PCost$ value of the answer returned by Algorithm 13 is less than B, there may exist solutions with lower diameter values and thus the algorithm decreases the value in D_{new} by half and continues the binary search. In each iteration, the optimal diameter lies between D_{prev} and D_{lower}, which are the upper and lower boundaries of the current search range, and D_{new} is the middle value between D_{prev} and D_{lower}. The boundaries are adjusted according to whether the $PCost$ value of the team returned by Algorithm 13 is greater than B or not. Thus, when the search range gets smaller, we get closer to the team with the minimum diameter under the $PCost$ budget. The process stops when the difference between D_{prev} and D_{lower} is smaller than an input precision threshold, and it outputs the last valid team returned by Algorithm 13, stored in $team_{prev}$.

The maximum number of iterations of Algorithm 15 is $\log_2 \frac{MaxDiameter}{\epsilon} + 1$. Thus, the time complexity of Algorithm 15 in the worst case is $O(m^2 \times n \times (\log_2 \frac{MaxDiameter}{\epsilon} + 1))$, where $O(m^2 \times n)$ is the worst case complexity of Algorithm 13. Since $MaxDiameter$ is the largest shortest distance between any two nodes in the input graph G, which is at most m times the maximum edge weight on the shortest path (where m is the number of nodes in G), the algorithm is polynomial in terms of input data.

Theorem 13 *Algorithm 15 is a* $(\log n, 2)$*-approximation algorithm for solving the* $(PCost,$

$diameter$) problem where n is the number of required skills in the project.

Proof

If G does not contain a B-bounded team, the algorithm returns $(\emptyset, \infty)$ and terminates. Now, assume that G contains a B-bounded team. Let the minimum (optimal) diameter of a B-bounded team in G be Opt_D. Let $Approx_D$ and $Approx_B$ specify the $diameter$ and $PCost$ values returned by Algorithm 15 respectively. Due to the hostility of the objectives, during any iteration of the binary search, we have $D_{lower} \leq Opt_D \leq D_{prev}$. Let D_{lower}^{final} and D_{prev}^{final} denote the final values of D_{lower} and D_{prev} respectively. We have $D_{prev}^{final} \leq D_{lower}^{final} + \epsilon$. Thus, $D_{prev}^{final} \leq Opt_D + \epsilon$. It is not difficult to see that the team returned by the algorithm is generated by Algorithm 13 with D_{prev}^{final} as the diameter budget. Thus, according to the approximation ratio of Algorithm 13, $Approx_D \leq 2D_{prev}^{final} \leq 2(Opt_D + \epsilon)$. Since ϵ is a small precision threshold, it can be ignored. Thus, Algorithm 15 returns a team from G whose diameter is at most 2 times that of the optimal B-bounded team. Let Opt_B denote the optimal $PCost$ value under the diameter budget D_{prep}^{final}, according to the approximation ratio of Algorithm 13, $Approx_B \leq \log n Opt_B$. Since $Opt_B \leq B$, we have $Approx_B \leq \log n B$. □

Since the general structure of Algorithm 15 is generic, it can be changed to solve $(PCost, sumDistance)$ problem by calling the appropriate algorithm at the places where Algorithm 13 is called.

6.4 Finding Pareto-optimal Teams

The algorithms above allow the user to provide a budget on one objective and finds the best solution on the other objective under the budget. Sometimes, the user may not want to specify budgets, but prefer to see all the optimal choices in the two-objective space so that he/she can select a solution that best fits his/her preferences. To this end, in this section we propose an algorithm that produces a set of optimal solutions that are not dominated by others. Below we define the relevant concepts , present the algorithm and prove the bounds of the solutions produced by the algorithm.

Definition 15 ***(Dominance)*** *A team T dominates a team T' (denoted by $T \prec T'$) with respect to the communication and personnel costs if T is better than T' in one objective and not worse than T' in the other objective.*

Definition 16 ***(Pareto-optimal team)*** *Given a project P, a team T is a Pareto-optimal team for project P if there does not exist a team T' that contains all the skills required by P such that $T' \prec T$.*

The set of all Pareto-optimal teams for project P is called the **Pareto set** of P. The teams in a Pareto set usually forms a convex curve (called **Pareto curve**) in the two-objective space.

A popular approach for finding Pareto-optimal solutions for multi-objective problems in the literature is to use an evolutionary algorithm, which is a heuristic method that

Algorithm 16 An approximation algorithm for finding Pareto Set of Team of Experts minimizing *diameter* and *PCost*.

Input: graph G, project $P = \{s_1, s_2, \ldots, s_n\}$, and precision threshold ϵ.

Output: *ParetoSet*

```
1: $MaxDiameter \leftarrow \max dist(c_i, c_j), 1 \leq i, j \leq m$ and $m$ is the number of nodes in $G$
2: $PT \leftarrow \emptyset$ /* for storing generated teams */
3: $Diameter \leftarrow MaxDiameter$
4: while $Diameter \geq 0$ do
5:   $\langle team, cost \rangle \leftarrow$ Algorithm13($G$, $P$, $Diameter$)
6:   $flag = 0$ /* for indicating whether $t$ is dominated */
7:   if $team \neq \emptyset$ then
8:     if Algorithm 13 is an approximation algorithm then
9:       for each $t$ in $PT$ do
10:        if $t \prec team$ then
11:          $flag = 1$
12:          break the for loop
13:        else
14:          if $team \prec t$ then
15:            remove $t$ from $PT$
16:    if $flag = 0$ then
17:      insert $team$ into $PT$
18:  else
19:    return $PT$
20:  $Diameter \leftarrow Diameter - \epsilon$
21: return $PT$
```

mimics the process of natural evolution. A problem with such a method is that there is no provable bound for the approximation ratio. Here we propose a new general procedure that makes use of the (α, β) approximation algorithms that we proposed in the last section to find a set of (approximate) Pareto-optimal solutions with performance bounds.

The algorithm for producing (approximate) Pareto-optimal answers based on $diameter$ and $PCost$ is presented in Algorithm 16. It repeatedly calls Algorithm 13 with a set of diameter budgets, starting from the diameter value of the input graph and decrementally changing the budget value by ϵ, which is an input precision threshold. In this way, a set of teams is generated each of which minimizes the personnel cost ($PCost$) under a diameter budget. If Algorithm 13 is an exact algorithm, the generated teams are guaranteed to be Pareto-optimal (See the proof of Theorem 14 below). If Algorithm 13 is an approximation algorithm (such as the Algorithm 13 proposed in Section 3), Algorithm 16 checks whether a newly-generated team is dominated by (or dominates) a previously-generated team . If it is dominated by a generated team, it is ignored. If it dominates a generated team, the generated team is removed and the new team is added to the set of Pareto teams. The worst case time complexity of Algorithm 16 is $O(\frac{MaxDiameter}{\epsilon} \times (m^2 \times n + \frac{MaxDiameter}{\epsilon}))$ where $m^2 \times n$ is the worst time taken by Algorithm 13.

Theorem 14 *Algorithm 16 produces Pareto-optimal teams if Algorithm 13 in line 5 returns an exact answer.*

Proof

Assume that T is a team returned by Algorithm 16 and that there exists a team Q that dominates T. Q dominates T if and only if one of the three conditions holds. We show none of these conditions can occur.

(1) $PCost(Q) < PCost(T)$ and $diameter(Q) < diameter(T)$. Since Q is better than T in both objectives, if Algorithm 13 is an exact algorithm, T would not be chosen under any diameter budget.

(2) $PCost(Q) = PCost(T)$ and $diameter(Q) < diameter(T)$. This is not possible since for teams with the same personnel cost, Algorithm 13 chooses the one with the least diameter.

(3) $PCost(Q) < PCost(T)$ and $diameter(Q) = diameter(T)$ T would not be chosen by Algorithm 13 if Algorithm 13 is an exact algorithm since Q is better than T in the personnel cost under any diameter budget no less than diameter(T).

Since team Q does not exist, team T is Pareto-optimal. □

The following theorem states how close a team generated by Algorithm 16 is to a Pareto-optimal team in the worse case if Algorithm 13 is the approximation algorithm as presented in Section 3.

Theorem 15 *For each team s' produced by Algorithm 16, there exists a team s in the Pareto set such that $diameter(s') \leq 2 \times diameter(s)$ and $PCost(s') \leq \log n \times PCost(s)$.*

Proof

This can be easily derived from Theorems 11 and 14. □

The following theorem states how well the teams in the Pareto Set are represented by the teams produced by Algorithm 16.

Theorem 16 *For each team s in the Pareto set, there exists a team s' produced by Algorithm 16 such that $diameter(s') \leq 2 \times (\epsilon + diameter(s))$ and $PCost(s') \leq \log n \times PCost(s)$, where ϵ is the input precision threshold of Algorithm 16.*

Proof

For any team s in the Pareto set, there is a team r in the Pareto set that is generated by Algorithm 16 if Algorithm 13 were an exact algorithm such that $diameter(r) \leq \epsilon + diameter(s)$, because the interval of the diameter budgets used in Algorithm 16 to call Algorithm 13 is ϵ. Let s' be the team generated by Algorithm 16 when Algorithm 13 is the approximation algorithm under the diameter budget that r were generated with the exact Algorithm 13. According to Theorem 11, we have $diameter(s') \leq 2 \times diameter(r)$. Thus, $diameter(s') \leq 2 \times (\epsilon + diameter(s))$.

For $PCost$, we have $PCost(s') \leq \log n \times PCost(r)$. Also, since r is on the Pareto curve, $PCost(r) \leq PCost(s)$. Thus, we have $PCost(s') \leq \log n \times PCost(s)$. □

To find the Pareto optimal solutions for minimizing $sumDistance$ and $PCost$, the appropriate algorithms can be used in Algorithm 16 at the places where Algorithm 13 is called. The corresponding approximation bounds can be derived similarly.

6.5 Experimental Evaluation

6.5.1 The Datasets and Experimental Setup

The DBLP and IMDb data sets are used in the experiments. The DBLP graph is produced from the DBLP XML data[34] taken on April 25, 2011. The dataset contains information about a collection of papers and their authors. The set of experts and skills are generated in the same way as in [34, 48] as follows. For each paper, the name(s) of the author(s), the conference where it was published and the title of the paper are specified. We only keep the papers of some major conferences in computer science: SIGMOD, VLDB, ICDE, ICDT, EDBT, PODS KDD, WWW , SDM, PKDD, ICDM, ICML, ECML, COLT, UAI, SODA, FOCS, STOC, and STACS. The set of experts consists of authors with at least 3 papers in the DBLP. The skills of an expert is the set of terms that appear in the titles of at least two papers of the expert. Two experts are connected together if they have at least two papers together. The weight of the edge between two nodes n_i and n_j is equal to $1 - |\frac{p_{n_i} \cap p_{n_j}}{p_{n_i} \cup p_{n_j}}|$ where p_{n_i} is the set of papers of author n_i. The cost of an expert in the DBLP is set to be the number of publications of the expert, assuming that the more publications an expert has, the more expensive he/she is. The final graph has 6,229 nodes and 9,400 edges.

The part of the IMDb dataset[35] used in our experiments contains information about the actors and the list of movies that each actor played in. It is processed in the same way

[34]http://dblp.uni-trier.de/xml/

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

as described in [34, 48]. The expert cost in IMDb is defined as the number of movies the actor plays in. The IMDb graph has 6,784 nodes and 35,875 edges. In our experiments, the DBLP and IMDb datasets show similar trends. Thus, most of the results presented below are from DBLP.

The projects used in the experiments are generated as follows. Each project is determined by a set of skills. The number of skills in a project varies from 4 to 10. For each number of skills, 100 projects are generated randomly. The average result over 100 projects is computed and used as the result of each algorithm. Skills have different frequencies. The frequency of a skill is the percentage of experts in the expert network that possess the skill. The frequency of the skills in our experiments varies from 0.1 to 5.0 percent. Other values of skill frequency show similar trends. All the algorithms are implemented in Java. The experiments are conducted on an Intel(R) Core(TM) i7 2.80 GHz computer with 4 GB of RAM.

6.5.2 Single Objective Methods

In this section, we show that single objective methods fail to find an affordable team of experts with acceptable communication cost. We implemented 3 single objective algorithms for such a purpose. The first algorithm is called *Rarest First*. It was introduced in [48] for minimizing the diameter. Note that this algorithm does not minimize the personnel cost. The second algorithm is called *Alg-PCost*. It minimizes the personnel

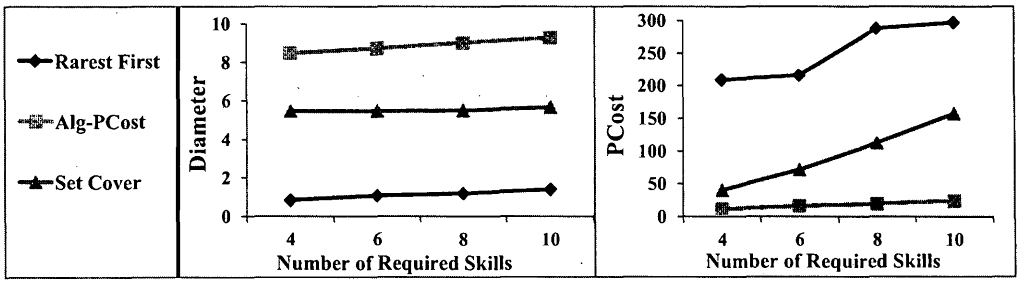


Figure 6.3: The diameter and *PCost* values of single objective methods for different number of required skills on DBLP dataset.

cost and do not minimize the communication cost. Note that *Alg-PCost* is the *weighted* greedy set cover algorithm [64]. In addition, we also implemented the greedy set cover algorithm (*Set Cover*) [64]. The objective of the greedy set cover problem is to minimize the number of experts and it does not minimize the communication cost nor the personnel cost. Figure 6.3 shows the diameter and *PCost* values of these methods for different numbers of required skills. As expected, the diameter of the *Rarest First* algorithm is much smaller than other methods because it minimizes the diameter of the answer. However, in terms of *PCost*, its values are the highest. *Alg-PCost* achieves the lowest personnel cost, but its teams have high diameter values. The results of the *Set Cover* method lie in the middle of other methods on both measures.

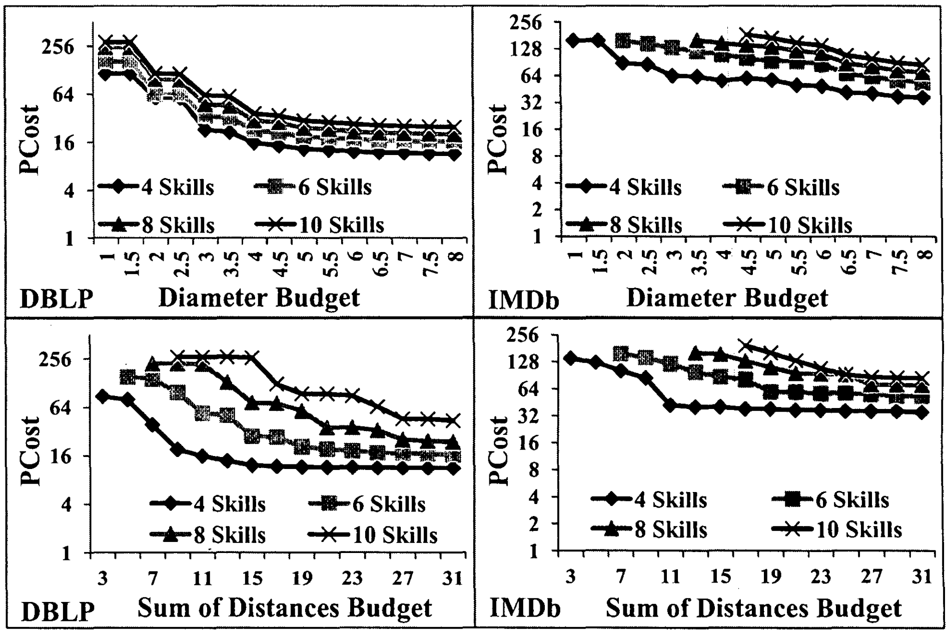


Figure 6.4: The personnel cost (shown in logarithmic scale) produced by our algorithms that receive a budget on the communication cost on DBLP and IMDb datasets. For some budget values, no team exists in the graph.

6.5.3 Results of Algorithms with Given Budget

6.5.3.1 Hostility between two objectives

Figure 6.4 shows the *PCost* values of the teams produced by our algorithms that receives budget on the communication cost for different budgets on *diameter* or *sumDistance*. Since the two objectives are hostile, by increasing the communication budget, the personnel cost decreases. For teams with the same budget, the more the required skills, the higher personnel costs. The results also show that our algorithms are able to find teams with both small personnel cost and small communication cost. For example, for 4 required skills, our Algorithm 13 is able to find a team with a *PCost* value of 16 and within a diameter budget of 4. Such a team cannot be found by the single objective methods that minimize either communication cost or personnel cost

6.5.3.2 Quality of Approximation Algorithms

We compare our approximation algorithms with the exact algorithms in terms of the quality of the answers. The answers of the exact algorithms are obtained using exhaustive search. Figure 6.5 shows the communication and personnel costs of the teams produced by the exact algorithms and Algorithms 13 and 14 for different budget values on diameter or sum of distances for projects with four skills. Due to the poor performance and long run time of the exhaustive search, the results of higher number of skills and higher communication cost budgets are not presented. The results show that the costs of the teams produced by our approximation algorithms are very close to those produced by the exact algorithms. The ratio of the diameter from Algorithm 13 or sum of distances from Algorithm 14 to the one from the exact algorithm is at most 1.29 or 1.68 respectively, although the theoretical bound for the approximation ratio is 2 (as shown in Theorem 11) or 4 (which is the number of required skills as shown in Theorem 12). This means that our approximation algorithms perform very well in practice, much better than the worse case scenario. The results also show that the *PCost* values of the teams produced by Algorithms 13 and 14 are sometimes slightly smaller than the ones from the exact algorithm. This seems a surprise. However, the reason is that some of the teams returned by the approximation algorithms have larger diameter/sum of distances than the budget. These teams are not considered by the exact algorithm. Therefore, they might have smaller personnel cost than the teams that actually lie within the communication budget. Note that these results do not violate the $(2, \log n)$ approximation ratio of Algorithm 13. The personnel cost of the approximation algorithm is at most $\log n$ times of the personnel cost of the exact answer. In this case, it is even smaller than the cost of the exact answer. For brevity, only the results of Algorithms 13 and 14 are presented. Other approximation algorithms have similar performance.

6.5.3.3 Precision vs. Run Time

As discussed before, the value of ϵ in Algorithm 15 determines the precision of the output teams. However, by increasing the precision (i.e., decreasing the value of ϵ), the run time increases. Figure 6.6 shows how the run time of Algorithm 15 changes with the ϵ value for different numbers of required skills on the DBLP dataset. As expected, by decreasing the value of ϵ, the run time increases close to linearly. It is because the run time is logarithmically related to the ratio of the diameter of the graph G to ϵ.

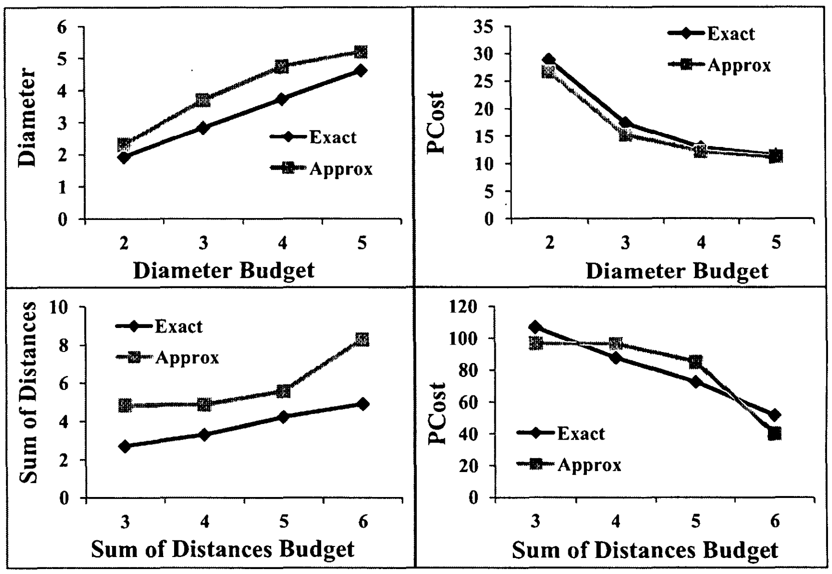


Figure 6.5: The costs of the teams from exact algorithms and Algorithms 13 and 14 on DBLP for projects with 4 skills.

6.5.4 Results of the Pareto Set Algorithm

In this section the effectiveness and efficiency of the proposed method for finding Pareto solutions are evaluated. The proposed method (Algorithm 16) is referred to as *Approx-Pareto*. To the best of our knowledge, there does not exist a Pareto optimization method for team formation. However, we implemented the following methods to compare with *Approx-Pareto*: (1) ***Exact-Pareto***: The exact Pareto set is found using exhaustive search. (2) ***Random-Pareto***: This method randomly selects a set of connected teams (1% of total teams), and then removes the teams that dominated by other generated teams. (3) ***GA-Pareto*** [25]: We apply a genetic algorithm for finding Pareto solutions proposed in [25]

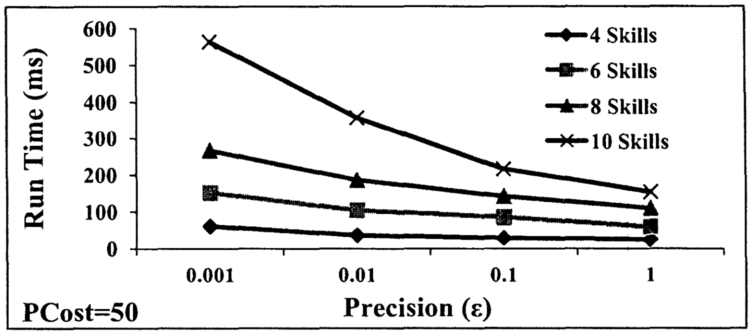


Figure 6.6: The run time of Algorithm 15 for different values of ϵ on DBLP dataset.

to our team formation problem. All the parameters are set in the same way as in [25].

We use the following performance measures: (1) ***Hypervolume*** (HV) [73]: It measures (in percentage) the volume of the dominated space by a generated Pareto set within search space composed by bounds of objective values. The higher the value, the better the Pareto set. (2) ***Average Distance*** (D_{avg}) and ***Maximal Distance*** (D_{max}) [13]: Given a true Pareto set R and a set S of approximate Pareto teams, D_{avg} is the average distance from each $y \in R$ to the closest team in S and D_{max} is the maximum distance between them. For both measures, lower values are preferred. (3) ***Precision*** and ***Recall***: $Precision = \frac{|True\ ParetoSet \bigcap Retrieved\ ParetoSet|}{|Retrieved\ ParetoSet|}$, $Recall = \frac{|True\ ParetoSet \bigcap Retrieved\ ParetoSet|}{|True\ ParetoSet|}$, where $|\cdot|$ denotes set cardinality. (4) ***Run time***. The result of the Exact-Pareto method is used as the true Pareto set for calculating H, D_{avg} and D_{max} indicators.

Table 6.1 shows the results of the algorithms for different numbers of required skills. The overall best results and best results among non-exact methods are highlighted in

Table 6.1: Results of algorithms for finding Pareto set (For *Approx-Pareto*, ϵ is set to 0.1).

# Skill	**Method**	**HV(%)**	D_{max}	D_{avg}	**Precision(%)**	**Recall (%)**	**Time (ms)**
3	Exact-Pareto	**47.6**	**0**	**0**	**100**	**100**	10,563
3	Approx-Pareto	**42.6**	**4.31**	**0.89**	**64**	**34.5**	498
3	GA-Pareto	29.8	150.81	39.60	3	1.1	353
3	Random-Pareto	30	102.83	18.23	7.1	3.6	**104**
4	Exact-Pareto	**45.8**	**0**	**0**	**100**	**100**	42,376
4	Approx-Pareto	**40.2**	**12.5**	**1.44**	**70.4**	**28.23**	647
4	GA-Pareto	34	58	10.06	4.4	2.9	921
4	Random-Pareto	37	90.8	15.87	2	0.87	**242**
5	Exact-Pareto	**55.3**	**0**	**0**	**100**	**100**	92,235
5	Approx-Pareto	**50.87**	**17**	**3.47**	**21.43**	**13.64**	968
5	GA-Pareto	50.14	28.01	8.37	0	0	2030
5	Random-Pareto	49.35	140	13.22	6.67	4.55	**496**

bold. Not surprisingly, *Exact-Pareto* gives the best or perfect results on all the quality measures (HV, D_{avg}, D_{max}. $Precision$ and $Recall$). However, its run time is orders of magnitude longer than those of the three non-exact methods. This indicates the need for non-exact algorithms. The results also indicate that *Approx-Pareto* significantly outperforms the other non-exact methods in terms of all the quality measures (HV, D_{avg}, D_{max}. $Precision$ and $Recall$). Its HV values are close to those of the exact method. In run time, the random method is the fastest as expected. Compared to the GA method, *Approx-Pareto* is slower than *GA-Pareto* when the number of required skills is 3, but is much faster than GA when the number of skills becomes a bit bigger. It is because by increasing the number of required skills from 3 to 4 or 5, the search space expansion of *GA-Pareto* is much more than *Approx-Pareto*.

6.6 Conclusion and Future Work

We studied the problem of finding an affordable and collaborative team from an expert network that minimizes two objectives: the communication cost among team members and the personnel cost. We proved that the problem we tackle is NP-complete. Two functions are used to measure the communication cost of a team and another function is proposed to evaluate the personnel cost of the team. A suite of algorithms classified into two approaches are proposed to solve this bicriteria problem. In the first approach, a budget is given on one objective and the purpose is to minimize the other objective under the budget. The budget could be either on the communication cost or the personnel cost. In the second approach, a set of approximate Pareto-optimal solutions are generated in which there exists no other team that dominates the solution in both of the costs. All of the proposed algorithms have provable approximation bounds. We evaluated the proposed algorithms on the DBLP and IMDb datasets and showed that our proposed algorithms are effective and efficient.

As future work, we consider adding another objective, i.e., maximizing the level of expertise in the team, into the problem, which leads to a challenging three objective optimization problem. Other communication and personnel cost functions might be applied and the approximation ratio of the proposed algorithms could be improved as well.

7 Conclusions

7.1 Review of Contributions

Original research has been accomplished in pursuit of this degree, and the results have been published in top-tier venues as TKDE [38], VLDB'11 [35], ICDE'12 [37] , ICDM'11 [36], SDM'13 [40], CIKM'11 [34] and ECML-PKDD'12 [39]. Some other parts of this dissertation are still under review in top-tier conferences/journals. In the course of this research, several topics in databases, data mining and theoretical computer science have been studied and extended.

In terms of keyword search over graph data, we propose to find r-cliques. The benefit of finding r-cliques are in two folds. First, the content nodes within an answer are guaranteed to be close to each other. Second, in the search process, only content nodes are explored rather than the whole graph. This improves the run time of the search process.

Previous work in graph keyword search might return duplicate or very similar answers which is obviously not desirable to the users. We propose a procedure which produces non-duplicate answers in polynomial delay. We further show that this procedure is faster than finding non-duplicate answers by post processing of the answers produced by previous methods. We define minimal answers as the one in which every content node covers an input keyword uniquely and illustrate some applications in which minimal answers are desirable. We propose approximation algorithms for finding duplication free and minimal answers efficiently.

For keyword search over relational databases which uses the schema based approach, the following ideas are proposed and implemented. We redefine answers via roles that capture important answers missed by previous techniques. In terms of ranking the networks of interconnected tuples, we devise importance measures for nodes, importance measures for edges, and a hybrid measure of the two. Then, we devise relevance measures for join trees derived from the schema relevance and study the effect of penalizing larger trees. A gold standard is constructed for relevance of nodes and edges from an extensive workload of real SQL queries. This is used to evaluate the effectiveness of our measures which do not require such external information. A comprehensive evaluation is performed based on the TPC-E schema to demonstrate the viability of our methods and to compare against existing methods.

We propose a suite of algorithms for finding a team of experts in a social network that minimizes both the communication cost and the personnel cost of the team. They are classified into two approaches as follows. In the first approach, a budget is given on one objective and the purpose is to minimize the other objective under the budget.

The budget could be either on the communication cost or on the personnel cost. In the second approach, a set of approximate Pareto-optimal solutions is generated in which no solution is dominated by any feasible solution in terms of the two costs. All of the proposed algorithms have provable approximation bounds and their viability are showed by extensive experiments on two real datasets.

7.2 Future Directions

We have provided novel and effective solutions to improve the state of the art in keyword search over graphs and relational databases, and in team formation from expert networks. However, there is still room for further research and improvement. The following directions can be explored, most of which have been mentioned in previous chapters.

- The approximation ratio of the proposed approximation algorithms for finding r-cliques might be improved or alternatively the ratio might be proved to be tight.
- The indexing method can be improved by efficiently updating the index in case of updating the input graph. The current indexing method does not support updates.
- Building keyword search engines over graph data using the *MapReduce* programming model can be another extension to this dissertation. In this case, the search engine is able to handle graphs with billions of nodes/edges in a distributed environment.
- The ratio of the proposed approximation algorithms for finding minimal answers

might be improved.

- Finding more applications for minimal answers in other domains (e.g., bioinformatics and computation biology networks) and applying the procedure proposed in this dissertation could be an interesting item of future work.
- Our work on finding meaningful answers for keyword search over relational databases could be improved by applying previous techniques such as pipelining in DISCOVER I [27]. Multi-query optimization [62] over the SQL queries generated for the MJNSs could be utilized to speed up evaluation greatly by exploiting the commonalities among queries.
- In finding meaningful answers for keyword search over relational databases, we seek to demonstrate how effective deriving relevance of the "nodes" and "edges" of the database schema could be based on just the schema and data. However, the results could be further improved by applying auxiliary information such as Linked Data[36] and WordNet[37]..
- In the team formation problem, other communication and personnel cost functions might be applied. The approximation ratio of the proposed algorithms might be improved as well.
- Another extension to the team formation problem could be adding more criteria to

[36]http://linkeddata.org

[37]http://wordnet.princeton.edu

the problem, such as the level of expertise or the availability of the experts.

Bibliography

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE'02*, 2002.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: Forming teams in large-scale community systems. In *Proc. of CIKM'10*, 2010.

[3] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *Proc. of the WWW'12*, 2012.

[4] E. M. Arkin and R. Hassin. Minimum-diameter covering problems. *Networks*, 36, 2000.

[5] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proc. of KDD'06*, 2006.

[6] J. Bar-Ilan, M. Mat-Hassan, and M. Levene. Methods for comparing rankings of search engine results. *Computer Networks*, 50:1448–1463, 2006.

[7] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis. Keyword search over relational databases: A metadata approach. In *Proc. of SIGMOD'11*, 2011.

[8] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE'02*, 2002.

[9] L. Blunschi, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger. Soda: Generating sql for business users. In *Proc. of VLDB'12*, 2012.

[10] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE'01*, 2001.

[11] F. Bourgeois and J. C. Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of ACM*, 14:802–804, 1971.

[12] C.Dorn and S.Dustdar. Composing near-optimal expert teams: A trade-off between skills and connectivity. In *Proc. of the International Conference on Cooperative Information Systems*, 2010.

[13] P. Czyzzak and A. Jaszkiewicz. Pareto simulated annealing a metaheuristic technique for multiple objective combinatorial optimization. *Journal of Multi Criteria Decision Analysis*, 7:34–47, 1998.

[14] B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. In *Proc. of VLDB'08*, 2008.

[15] S. Datta, A. Majumder, and K. Naidu. Capacitated team formation problem on social networks. In *Proc. of KDD'12*, 2012.

[16] B. Ding, J. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *Proc. of ICDE'07*, 2007.

[17] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[18] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. In *Proc. of VLDB'10*, 2010.

[19] E. Fitzpatrick and R. Askin. Forming effective worker teams with multi functional skill requirements. *Comput. Ind. Eng.*, 48(3):593–608, 2005.

[20] A. Gajewar and A. D. Sarma. Multi skill collaborative teams based on densest subgraphs. In *Proc. of the SDM'12*, 2012.

[21] M. Gaston, J. Simmons, and M. desJardins. Adapting network structures for efficient team formation. In *Proc. of the AAAI Fall Symposium on Artificial Multi-agent Learning*, 2004.

[22] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *Proc. of SIGMOD'08*, 2008.

[23] Z. Guan, G. Miao, R. McLoughlin, X. Yan, and D. Cai. Co-occurrence based diffusion for expert search on the web. *Transactions on Knowledge and Data Engineering*, pre-print, 2012.

[24] H. He, H. Wang, J. Yang, and P. Yu. Blinks: ranked keyword searches on graphs. In *Proc. of SIGMOD'07*, 2007.

[25] J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, 1994.

[26] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *Proc. of VLDB'03*, 2003.

[27] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. of VLDB'02*, 2002.

[28] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on xml graphs. In *Proc. of ICDE'03*, 2003.

[29] M. Jackson. *Network formation.* The New Palgrave Dictionary of Economics and the Law, 2008.

[30] M. Jayapandian and H. V. Jagadish. Automated creation of a forms based database query interface. In *Proc. of VLDB'08*, 2008.

[31] D. Johnson, M. Yannakakis, and C. Papadimitriou. On generating all maximal independent sets. *Info. Proc. Lett.*, 27, 1998.

[32] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB'05*, 2005.

[33] D. Kalyanmoy. Multi-objective optimization. In *Search Methodologies*, pages 273–316. Springer US, 2005.

[34] M. Kargar and A. An. Discovering top-k teams of experts with/without a leader in social networks. In *Proc. of CIKM'11*, 2011.

[35] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. In *Proc. of VLDB'11*, 2011.

[36] M. Kargar and A. An. Teamexp: Top-k team formation in social networks. In *Proc. of ICDM'11*, 2011.

[37] M. Kargar and A. An. Efficient top-k keyword search in graphs with polynomial delay. In *Proc. of ICDE'12*, 2012.

[38] M. Kargar, A. An, and X. Yu. Efficient duplication free and minimal keyword search in graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25, 2013.

[39] M. Kargar, A. An, and M. Zihayat. Efficient bi-objective team formation in social networks. In *Proc. of ECML-PKDD'12*, 2012.

[40] M. Kargar, M. Zihayat, and A. An. Finding affordable and collaborative teams from a network of experts. In *Proc. of SIAM International Conference on Data Mining (SDM'13)*, 2013.

[41] R. Karp. Reducibility among combinatorial problems. *Compexity of computer computations*, 1972.

[42] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. *Supercomputing*, 1995.

[43] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum. Star: Steiner-tree approximation in relationship graphs. In *Proc. of ICDE'09*, 2009.

[44] M. Kendall. A new measure of rank correlations. *Biometrika*, 30:91–93, 1983.

[45] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *Proc. of SIGMOD'06*, 2006.

[46] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *Proc. of KDD'06*, page 245255, 2006.

[47] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15, 1981.

[48] T. Lappas, L. Liu, and E. Terzi. Finding a team of experts in social networks. In *Proc. of KDD'09*, 2009.

[49] E. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18, 1972.

[50] C. Li and M. Shan. Team formation for generalized tasks in expertise social networks. In *Proc. of IEEE International Conference on Social Computing*, 2010.

[51] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: Efficient and adaptive keyword search on unstructured, semi-structured and structured data. In *Proc. of SIGMOD'08*, 2008.

[52] Z. Liu and Y. Chen. Processing keyword search on xml: a survey. *World Wide Web*, 14:671–707, 2011.

[53] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: Top-k keyword query in relational databases. In *Proc. of SIGMOD'07*, 2007.

[54] M. V. Marathe, R. Ravi, R. Sundaram, S. Ravi, D. J. Rosenkrantz, and H. B. Hunt. Bicriteria network design problems. *Journal of Algorithms*, 28(5):142–171, 1998.

[55] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge Univ. Press, 1995.

[56] M. Newman. The structure of scientific collaboration networks. In *Proc. of the National Academy of Sciences*, 2001.

[57] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30:41–82, 2005.

[58] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. of FOCS'00*, 2000.

[59] J. Park and S. Lee. Keyword search in relational databases. *Knowledge and Information Systems*, 26:175–193, 2011.

[60] L. Qin, J. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *Proc. of ICDE'09*, 2009.

[61] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: The power of rdbms. In *Proc. of SIGMOD'09*, 2009.

[62] T. K. Sellis and S. Ghosh. On the multiple-query optimization problem. *IEEE Trans. Knowl. Data Eng.*, 2:262–266, 1990.

[63] D. Srivastava and S. Venkatasubramanian. Information theory for data management. In *Proc. of SIGMOD'10*, 2010.

[64] V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[65] S. V. N. Vishwanathan and A. Smola. Fast kernels on strings and trees. In *Proc. of NIPS'02*, 2002.

[66] H. Wi, S. Oh, J. Mun, and M. Jung. A team formation model based on knowledge and collaboration. *Expert Syst. Appl.*, 36(5):9121–9134, 2009.

[67] X. Yang, C. M. Procopiuc, and D. Srivastava. Summarizing relational databases. In *Proc. of VLDB'09*, 2009.

[68] X. Yang, C. M. Procopiuc, and D. Srivastava. Summary graphs for relational database schemas. In *Proc. of VLDB'11*, 2011.

[69] J. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17, 1971.

[70] J. Yu, L. Qin, and L. Chang. *Keyword Search in Databases*. Morgan and Claypool Publisher, 2010.

[71] X. Yu and H. Shi. Ci-rank: Ranking keyword search results based on collective importance. In *Proc. of ICDE'12*, 2012.

[72] F. Zhao, X. Zhang, A. K. H. Tung, and G. Chen. Broad: Diversified keyword search in databases. In *Proc. of VLDB'11*, 2011.

[73] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, ETH Zurich, Switzerland, 1999.