Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

SCHEMA-AWARE KEYWORD SEARCH ON LINKED DATA

by
Ananya Dass

Keyword search is a popular technique for querying the ever growing repositories of RDF graph data on the Web. This is due to the fact that the users do not need to master complex query languages (e.g., SQL, SPARQL) and they do not need to know the underlying structure of the data on the Web to compose their queries. Keyword search is simple and flexible. However, it is at the same time ambiguous since a keyword query can be interpreted in different ways. This feature of keyword search poses at least two challenges: (a) identifying relevant results among a multitude of candidate results, and (b) dealing with the performance scalability issue of the query evaluation algorithms.

In the literature, multiple schema-unaware approaches are proposed to cope with the above challenges. Some of them identify as relevant results only those candidate results which maintain the keyword instances in close proximity. Other approaches filter out irrelevant results using their structural characteristics or rank and top-k process the retrieved results based on statistical information about the data. In any case, these approaches cannot disambiguate the query to identify the intent of the user and they cannot scale satisfactorily when the size of the data and the number of the query keywords grow. In recent years, different approaches tried to exploit the schema (structural summary) of the RDF (Resource Description Framework) data graph to address the problems above. In this context, an original hierarchical clustering technique is introduced in this dissertation. This approach clusters the results based on a semantic interpretation of the keyword instances and takes advantage of relevance feedback from the user. The clustering hierarchy uses pattern graphs which are structured queries and clustering together result graphs with the same structure. Pattern graphs represent possible interpretations for the keyword query. By navigating though the hierarchy the user can select the pattern graph which is relevant to

her intent.

Nevertheless, structural summaries are approximate representations of the data and, therefore, might return empty answers or miss results which are relevant to the user intent. To address this issue, a novel approach is presented which combines the use of the structural summary and the user feedback with a relaxation technique for pattern graphs to extract additional results potentially of interest to the user. Query caching and multi-query optimization techniques are leveraged for the efficient evaluation of relaxed pattern graphs. Although the approaches which consider the structural summary of the data graph are promising, they require interaction with the user.

It is claimed in this dissertation that without additional information from the user, it is not possible to produce results of high quality from keyword search on RDF data with the existing techniques. In this regard, an original keyword query language on RDF data is introduced which allows the user to convey his intention flexibly and effortlessly by specifying cohesive keyword groups. A cohesive group of keywords in a query indicates that its keywords should form a cohesive unit in the query results. It is experimentally demonstrated that cohesive keyword queries improve the result quality effectively and prune the search space of the pattern graphs efficiently compared to traditional keyword queries. Most importantly, these benefits are achieved while retaining the simplicity and the convenience of traditional keyword search.

The last issue addressed in this dissertation is the diversification problem for keyword search on RDF data. The goal of diversification is to trade off relevance and diversity in the results set of a keyword query in order to minimize the dissatisfaction of the average user. Novel metrics are developed for assessing relevance and diversity along with techniques for the generation of a relevant and diversified set of query interpretations for a keyword query on an RDF data graph. Experimental results show the effectiveness of the metrics and the efficiency of the approach.

SCHEMA-AWARE KEYWORD SEARCH ON LINKED DATA

by
Ananya Dass

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology and
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2016

APPROVAL PAGE

SCHEMA-AWARE KEYWORD SEARCH ON LINKED DATA

Ananya Dass

Dr. Dimitri Theodoratos, Dissertation Advisor — Date
Associate Professor, Department of Computer Science, NJIT

Dr. Yi Chen, Committee Member — Date
Associate Professor and Henry J. Leir Chair, School of Management, NJIT

Dr. James Geller, Committee Member — Date
Professor and Associate Dean of Research, Department of Computer Science, NJIT

Dr. Michael Halper, Committee Member — Date
Professor and Director, Information Technology Program, NJIT

Dr. Vincent Oria, Committee Member — Date
Associate Professor and Associate Chair, Department of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Ananya Dass

Degree: Doctor of Philosophy

Date: May 2016

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, New Jersey, 2016
- Bachelor of Technology in Computer Engineering,
  West Bengal University of Technology, Kolkata, India, 2011

Major: Computer Science

Presentations and Publications:

Dass, A., Aksoy, C., Dimitriou, A., & Theodoratos, D. *Diversifying the Results of Keyword Queries on Linked Data*, submitted to International Conference on Web Information System Engineering (WISE'16), 15 pages.

Dass, A., Aksoy, C., Dimitriou, A., & Theodoratos, D. *Relaxation for Keyword Pattern Graphs on Linked Data*, submitted to Journal on Web Engineering (JWE), Rinton Press, 36 pages.

Aksoy, C., Dimitriou, A., Dass, A., & Theodoratos, D. *Clustering Query Result Patterns for Effective Keyword Search on Tree Data*, submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE), under 2nd review, 14 pages.

Aksoy, C., Dass, A., Theodoratos, D., & Wu. X. *Diversification of Keyword Query Result Patterns*, Proceedings of the 17th International Conference on Web-Age Information Management (WAIM'16), Nanjing, China, Springer, LNCS, 12 pages.

Dimitriou, A., Dass, A., Theodoratos, D.,& Vasiliou, Y. *Cohesive Keyword search on Tree Data*, Proceedings of the 19th International Conference on Extending Database Technology (EDBT'16), Bordeaux, France, March 2016, pages 149-160.

Dass, A., Dimitriou, A., Aksoy, C., & Theodoratos, D. *Incorporating Cohesiveness into Keyword Search on Linked Data*, Proceedings of the 19th International Conference on Web information System Engineering (WISE'15), Miami, Florida, Oct. 2015, Springer, LNCS, pages 47-62.

Dass, A., Aksoy, C., Dimitriou, A., & Theodoratos, D. *Keyword Pattern Graph Relaxation for Selective Result Space Expansion on Linked Data*, Proceedings of the 15th International Conference on Web Engineering (ICWE'15), Rotterdam, the Netherlands, 2015, Springer, LNCS, pages 287-306.

Dass, A., Aksoy, C., Dimitriou, A., & Theodoratos, D. *Exploiting Semantic Result Clustering to Support Keyword Search on Linked Data*, Proceedings of the 18th International Conference on Web information System Engineering (WISE'14), 2014, Thessaloniki, Greece, Springer, LNCS, pages 448-463.

Aksoy, C., Dass, A., Theodoratos, D., & Wu, X. *Clustering Query Results to Support Keyword Search on Tree Data*, Proceedings of the 15th International Conference on Web-Age Information Management (WAIM'14), Macau, China, April 2014, Springer, LNCS, pages 213-224.

To my beloved mom, Mrs Subhra Dass
for her unconditional love and support,
for her silent sacrifices andcountless blessings,
without her I am incomplete, my success is without glory.

To my best teacher, motivator and friend, Dr. Debasis Barman
for his confidence in me, when I hadnonefor myself,
for encouraging me to dream high and chase them,
for supporting me when I failed and uplifting my spirit to strive harder again,
without him I would not have been where I am today.

To my family,
for being there with me to share my sorrows, to celebrate my joy
for making my life enjoyable with their presence, rich with their experience,
without them life would not have been worth living.

ACKNOWLEDGMENT

Firstly, I would like to express my unfeigned gratitude to my dissertation advisor Dr. Dimitri Theodoratos for being the torchbearer of research during my Ph.D. studies. I deeply appreciate his efforts and patience in supervising me, who had no or little experience in research prior to joining the Ph.D program. He illuminated the spirit of research in me and nurtured it with his intellectual guidance whenever needed. Furthermore, in every life front, whether academics, career, or personal, he compassionately lent his support and enabled me to combat with any challenges I confronted in this alien land.

I would also like to convey my sincere thanks to Dr. Yi Chen, Dr. James Geller, Dr. Michael Halper and Dr. Vincent Oria for being a part of my dissertation committee. They put the time and efforts to scrutinize my work and provided me with their valuable comments for the improvement of my work and the successful accomplishment of my Ph.D. Additionally, Dr. James Geller and Dr. Vincent Oria readily extended their support by offering to serve as a reference for me, and recommended me so that I can chase and attain success in my future endeavors.

My colleague Dr. Cem Aksoy, deserves a special acknowledgement for his contributions to my studies. His thoughtful advice aided my work time to time and his friendly support always uplifted my spirit.

I am also grateful to my colleagues and co-authors in my publications Aggeliki Dimitriou and Xiaoying Wu for their help.

I would like to deeply thank the Department of Computer Science at New Jersey Institute of Technology for offering financial support during the course of my Ph.D studies. My genuine thanks also go to my fellow office mates Xiguo Ma, Sheetal Rajgure, Jichao Sun, Arwa Wali, Xiangqian Yu, Souvik Sinha and Animesh Dwivedi for being amazing companions and supports throughout my studies.

A very special thanks to Dr. Debasis Barman, a professor from my undergrad

university, to whom I am absolutely indebted. He not only imparted knowledge on different subjects of science, he also taught me valuable lessons of life. He exemplifies an ideal teacher in my life and inspires me to be a good teacher. My initial motivation for research had its source from assisting Dr. Barman in his research. It is all his vision, and immense confidence in me that lead me through this Ph.D program. He bestowed me with profound parental affection and support in both academic and personal life. I dedicate this dissertation to him.

Last, but not the least, I would like to express my heartfelt gratitude to my family for being so loving and for supporting me spiritually and emotionally through thick and thin during my PhD and my life in general. They always encouraged me and had faith in all the choices I made. A special mention, my mom, to whom I cannot be grateful enough, for all the sacrifices she made, all the hardships she went through to ensure for me a blooming future.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION 1

1.1 Challenges of Keyword Search on Semi-structured Data 3

1.2 Exploiting Semantic Result Clustering to Support Keyword Search 5

1.3 Keyword Pattern Graph Relaxation to Retrieve Additional Relevant Results 6

1.3.1 Benefits of the Structural Summary 8

1.3.2 The Missing Relevant Result Problem 9

1.3.3 Our Solution: Keyword Pattern Graph Relaxation 10

1.4 Cohesive Keyword Queries: As Simple As Traditional Keyword Queries . . 11

1.5 Diversification of Keyword Search Results 15

1.6 Organization 18

2 STATE OF THE ART 19

2.1 Search on RDF Data 19

2.1.1 Resource Description Framework 19

2.1.2 Structured Queries on RDF Data 20

2.1.3 Keyword Queries on RDF Data 20

2.2 Keyword Search Approaches on Structured and Semi-structured Databases . 24

2.2.1 Form of the Answer 24

2.2.2 Schema-agnostic Approaches 25

2.2.3 Schema-aware Approaches 25

2.2.4 Keyword Query Evaluation Algorithms 26

2.2.5 Relevance Assessment of the Results 27

3 RDF DATA MODEL AND QUERY LANGUAGE SEMANTICS 29

3.1 RDF Data Model 29

TABLE OF CONTENTS
(Continued)

Chapter **Page**

3.2 Queries and Answers 30
3.3 The Structural Summary: Summarized RDF Data 35
3.4 Pattern Graphs: Structured Queries on Structural Summary 37

4 EXPLOITATION OF SEMANTIC RESULT CLUSTERING 38
4.1 An Algorithm for Computing Query Pattern Graphs 38
4.2 Semantic Hierarchical Clustering and Ranking 40
4.3 Experimental Evaluation 42
4.3.1 Dataset and Queries 42
4.3.2 Effectiveness of Hierarchical Clustering 43
4.3.3 Efficiency of the System 44
4.4 Conclusion 45

5 KEYWORD PATTERN GRAPH RELAXATION 47
5.1 Computing and Selecting Pattern Graphs 47
5.2 Computing Relaxed Pattern Graphs 48
5.2.1 Relaxed Pattern Graphs 49
5.2.2 Vertex Splitting 51
5.2.3 Measuring Pattern Graph Relaxation 52
5.2.4 Relaxation Order 56
5.3 Computing Relaxed Pattern Graphs 57
5.3.1 Identifying Empty Vertices for Relaxation 58
5.3.2 An Algorithm for Computing Relaxed Pattern Graphs 59
5.3.3 Optimization Techniques to Support Query Relaxation and Evaluation 63
5.4 Experimental Evaluation 67

TABLE OF CONTENTS
(Continued)

Chapter **Page**

5.4.1 Datasets and Queries 67
5.4.2 Effectiveness in Ranking Relaxed Pattern Graphs 69
5.4.3 Efficiency of the System in Producing Relaxed Results 72
5.5 Conclusion 73

6 INCORPORATING COHESIVENESS INTO KEYWORD SEARCH 75
6.1 Data Model and Flat Keyword Queries 75
6.2 Keyword Queries with Cohesive Keyword Groups 77
6.2.1 Syntax 77
6.2.2 Semantics 78
6.3 An Algorithm for Evaluating Cohesive Queries 80
6.3.1 Structural Summary and Pattern Graphs 80
6.3.2 The Basic Components of the Algorithm 81
6.3.3 Algorithm Description 83
6.4 Experimental Evaluation 85
6.4.1 Datasets and Queries 85
6.4.2 Effectiveness of the Cohesive Queries 86
6.4.3 Efficiency of Algorithm CohesivePGGen 88
6.5 Conclusion 88

7 INTRODUCING DIVERSITY IN THE RESULT SETS OF KEYWORD QUERIES 90
7.1 Data Model and Pattern Graph Computation 90
7.2 Balancing Relevance and Diversity 92
7.2.1 Problem Statement 92
7.2.2 Assessing the Relevance of a Pattern Graph Set 93

TABLE OF CONTENTS
(Continued)

Chapter **Page**

7.2.3 Assessing the Diversity of a Pattern Graph Set 95
7.3 Algorithm 98
7.4 Experimental Results 100
7.4.1 Datasets and Queries 100
7.4.2 Effectiveness Results 101
7.4.3 Efficiency Results 103
7.5 Conclusion 105

8 CONCLUSION AND RESEARCH DIRECTIONS 106

REFERENCES 110

LIST OF TABLES

Table **Page**

4.1 Queries used in the Experiments and their Statistics 43

5.1 The Keyword Queries in the Two Datasets 68

6.1 Queries on Jamendo and DBLP Datasets 86

7.1 Keyword Queries on Jamendo and DBLP Datasets 100

LIST OF FIGURES

Figure **Page**

1.1 (a) An RDF data graph D, (b) The structural summary S_D of D. 7

1.2 (a), (b), (c), and (d) Pattern graphs. 7

1.3 (a) Pattern graph (b) Result graph (c) Relaxed pattern graph. 10

1.4 Result graphs of the query $Q =$ `(Andrew Job Gordon Network)` 12

1.5 (a) An RDF graph D, (b) The Structural Summary of S. 16

1.6 (a), (b), (c) and (d) are different patterns graphs of the keyword query $Q =$`{ Johns, Hopkins, Computer}`. 16

3.1 An RDF graph. 30

3.2 Matching constructs (a) class (b) value (c) relationship (d) property. 32

3.3 Inter-construct connection. 32

3.4 (a) Invalid result graph (b) Valid result graph (c) and (d) result graphs with overlapping matching constructs. 34

3.5 (a) Structural Summary, (b), (c), (d) and (e) are value, class, property, and relationship *MCs*, respectively. 36

3.6 Query Pattern Graph. 37

4.1 Reach times of the two clustering approaches. 44

4.2 (a) Time to compute the MCs for the query keywords (b) Evaluation time for the selected pattern graphs. 45

5.1 (a) An RDF graph, (b), (c), (d) and (e) class, relationship, value and property matching constructs, respectively, (f) inter-construct connection and result graph. 48

5.2 An original pattern graph P_1 and relaxed pattern graphs P_2, P_3, P_4. 50

5.3 (a) Original pattern graph, (b), (c), (d), (e) and (f) relaxed pattern graphs. 54

5.4 (a) Original pattern graph (b) and (c) relaxed pattern graphs. 56

LIST OF FIGURES
(Continued)

Figure **Page**

5.5 Star-join view of entity variable vertex X. 58

5.6 A global evaluation plan for relaxed queries. 65

5.7 $nDCG_{max}$, $nDCG_{min}$ and $nDCG_{avg}$ for the queries on the Jamendo and YAGO datasets. 71

5.8 Kendall tau-b coefficient for the queries on the Jamendo and YAGO datasets. . 72

5.9 Efficiency improvement achieved by Multiquery optimization on two datasets. . 73

6.1 (a) An RDF Graph, (b), (c), (d) and (e) class, relationship, value and property matching constructs respectively, (f) inter-construct connection and query instance. 76

6.2 (a) result graph, (b) and (c) query instances which are not result graphs for the query `(Publication (Project (Semantics 2015)) (author (Grace Hopper)))`. 79

6.3 (a) Structural Summary (b) Query Pattern Graph. 81

6.4 (a) Parse tree (b) Generation of pattern graphs of the query by the algorithm. . . 84

6.5 % reduction on the number of pattern graphs for the queries on the two datasets. 87

6.6 Average *precision@k* for cohesive queries and their corresponding flat queries varying k on the two datasets. 87

6.7 Execution time of CohesivePGGen on cohesive and flat queries. 88

7.1 (a) An RDF graph, (b), (c), (d) and (e) class, relationship, value and property matching constructs, respectively, (f) inter-construct connection and result graph. 91

7.2 (a) Structural Summary G', (b), (c), (d), and (e) are matching constructs for keywords in the keyword query `Q1={Tom, author, Project, title}` on G', (f) Pattern Graph of Q on G'. 91

7.3 Five Pattern graphs of `Q={Tom, semantics,publication, hopper, project}` 96

LIST OF FIGURES
(Continued)

Figure **Page**

7.4 nDCG$_{max}$, nDCG$_{min}$ and nDCG$_{avg}$ for the queries of Table 7.1 on the two datasets. 102

7.5 Prec@k for $k = 3$, 5 and 10, for the queries of Table 7.1 on the Jamendo and DBLP datasets based solely on the relevance metric. 103

7.6 Prec@k for $k = 3$, 5 and 10, for the queries of Table 7.1 on the Jamendo and DBLP datasets based on the relevance and diversity metrics. 104

7.7 Processing time of *PGDiversification* algorithm for the queries on the Jamendo and DBLP datasets. 104

CHAPTER 1

INTRODUCTION

Nowadays, a vast amount of information is available on the World Wide Web accommodating all aspects of human activities, knowledge and experiences. The original vision of Tim Berners-Lee [10], the inventor of World Wide Web, was to enable the publication and interconnection of documents through the Internet. Over the years the number of documents kept increasing on the Web. Search engines were invented to aid users to discover information from the Web that is relevant to their needs. Apart from the constant increase in the number of documents on the Web, there is a current trend to publish data and to make it easily discoverable, accessible, and available to all the Web users without any restriction [49]. In recent days, due to the Open Data movement a plethora of data sources became available on the Web from various domains, from government data to scientific datasets. As a consequence, the Web evolved from a Web of documents to a Web of data. The current need in this Web of data infrastructure is to have the data interlinked and integrated so as to enable the users to combine information from different data sources and extract composite knowledge. Currently, the Web users by themselves are performing this Web data integration. An automatic process for the integration of all available online data would simplify our everyday lives. The vision of the Semantic Web has been expressed to offer solutions to the above problems [11]:

> *"The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."*

The Semantic Web extends the current Web by introducing machine-readable data and metadata of the documents and information of how they are interconnected. With the inclusion of intelligent and automatic processes that perform tasks on behalf of the users,

the Web is evolving into the Semantic Web. The advocates of the Semantic Web envisioned a standardized means for representing structured and meaningful information on the Web in order to realize the goals of Semantic Web. Several data models have been proposed for representing such information. The most prominent one is the Resource Description Framework (RDF) [57]. RDF allows and provides a simple and abstract knowledge representation for resources on the Web which are uniquely identified by Universal Resource Identifiers (URIs). Each statement in RDF can be encoded as an RDF triple. Additionally, ontologies are used to give meaning to resources by grouping them into classes and also to identify the relationships between these classes. RDFS and OWL are the two most commonly used languages for encoding ontologies. The vocabulary language of RDF is RDF Schema (RDFS) [36]. RDFS defines and specifies meanings to the terms that will be used in RDF statements. Web Ontology Language (OWL) [41] can also be used for conceptualization and provides further expressivity in stating relationships among the resources. A query language is necessary for exploring and querying the structured information expressed in RDF. During the past years, many query languages have been proposed for querying and retrieving information from the RDF data model. Since January 2008, SPARQL has been the official W3C recommendation language for querying RDF data [76].

Today, the Semantic Web is more than just a vision. This is manifested by the publication of large datasets according to the principles of the Linked Data initiative[1]. The Linked Data initiative aims at offering a new way of data integration and interoperability by connecting data sources on the Web and exposing real life data using semantic technologies. A Web of Data is the outcome of the Linked Data initiative and efforts. In this Web of Data infrastructure, URIs identify real life things. RDF information about those things can be obtained by following those URIs. Furthermore, this RDF information contains related URIs which are links to other resources enabling further exploration. The Linked Data community has established a set of best practices for collaboratively publishing and

[1] http://linkeddata.org/ (accessed on April 24, 2016)

interlinking structured data on the Web [14, 40]. Ranging from community-driven efforts to governmental bodies or scientific groups, there are numerous sources that publish their data on the Web in the form of Linked Data. DBpedia[2][26], BBC music information [82], open government data[3] are only a few examples of the constantly increasing Linked Data cloud[4].

With the proliferation of RDF graph data repositories, keyword search is by far the most popular technique for querying linked data on the Web. Keyword search offers a straightforward, intuitive, and flexible method for retrieving information. In this context, users anticipate that with the sole use of some keywords they will be able to satisfy their information needs. The success of keyword search is due to the following facts: (a) it allows the user to retrieve information without knowing any formal query language (e.g., SPARQL), (b) it allows the user to retrieve information without being aware of the structure/schema of the data source against which the keyword query is issued, and (c) the same keyword query can be used to extract data from multiple data sources with different structures.

1.1 Challenges of Keyword Search on Semi-structured Data

The advantages of keyword search on semi-structured data on the Web come with a number of disadvantages. Keyword queries are ambiguous in determining both: the user intent and the form of the results. For this reason, keyword search on tree and graph data faces three major challenges:

(a) **Determining the form of the results:** In contrast to the IR domain where the answer of a keyword query is a set of flat documents, in the domain of tree and graph databases,

[2]http://dbpedia.org (accessed on April 24, 2016)

[3]http://www.data.gov/, http://www.data.gov.uk/ (accessed on April 24, 2016)

[4]http://www4.wiwiss.fu-berlin.de/lodcloud/state/ (accessed on April 24, 2016)

each result in the keyword query answer is a database substructure (e.g., node, subtree, subgraph). This not only multiplies enormously the number of candidate results and, therefore, makes the evaluation more complex, but it also raises the issue of determining an appropriate form for the results. The goal is to return results (substructures) that are meaningful to the user. Different approaches on tree data define the results as LCA nodes [93, 26], minimum connecting trees [44], instance trees [5], etc. In the context of graph data, multiple approaches return trees [32, 47, 90] usually constrained by the adopted search algorithms. Indeed, traditional keyword search algorithms on graphs associate keywords only to vertices and, by construction, compute and return minimum spanning trees [12, 34, 39, 48]. However, tree structures do not appropriately capture the semantics of queries on graph data which should naturally return graph structures. Further, in RDF data, semantics are assigned to graph elements through their association to schema elements. This information should be taken into account in the search process and integrated in the query results in order to help disambiguating the queries and their results. In this direction, some approaches attempt to exploit predicates [31, 47, 89].

(b) **Identifying the relevant results:** Because of the ambiguity of keyword queries there is usually an overwhelming number of results matching the query keywords (candidate results) of which only a tiny portion is relevant to the user intent. Multiple approaches assign semantics to keyword queries by exploiting structural and semantic features of the data in order to automatically filter out irrelevant results [68]. Although filtering approaches are intuitively reasonable, they are sufficiently ad-hoc and they are frequently violated in practice resulting in low precision and/or recall. A better technique followed by some other approaches ranks the candidate results in descending order of their estimated relevance [37, 68]. Given that users are typically interested in a small number of query results, some of these approaches combine ranking with top-k algorithms for keyword search [32, 89, 90]. Keyword search over graph data returns a multitude of candidate results and of extended diversity. Therefore, current algorithms compute candidate results in an approximate way

by considering only those which maintain the keyword instances in close proximity [12, 28, 34, 39, 48, 50, 63, 78]. The ranking and top-k processing of the filtered results usually employ IR-style metrics for flat documents (e.g., tf*idf or PageRank) [31, 37, 68, 90, 89] adapted to the structural characteristics of the data. However, the occurrence statistics alone can neither capture effectively the diversity of the results represented in a large graph dataset nor identify the intent of the user. As a consequence the produced rankings are, in general, of low quality.

(c) **Coping with the performance scalability issue:** As mentioned above, the number of candidate results can be very large. Computing all the results of a certain form is intractable. For instance, the problem of finding the Steiner trees for a set of keywords in a data graph is NP-complete [50]. The current algorithms, which compute all the results of a certain restricted form so that their size is below a certain threshold, are still of high complexity. Consequently, these algorithms do not scale satisfactorily when the size of the data graph and the number of query keywords increase.

1.2 Exploiting Semantic Result Clustering to Support Keyword Search

In this dissertation, we present a novel approach for keyword search on RDF graph data. Our approach utilizes a semantic two-level hierarchical clustering of the keyword query results. We define keyword query results as meaningful subgraphs of the data graph that appropriately connect together keyword *matching constructs* (elementary subgraphs representing semantic interpretations of the keyword instances). This definition addresses the problem of determining the form of query result. In our two-level hierarchical clustering, the first—fine-grained—level of clustering, partitions the results that share the same structure and cluster them together as pattern graphs. These pattern graphs involve all the matching constructs of the query keywords on the structural summary and express the possible interpretations of the keyword query on the graph data. The pattern graphs, when

used as queries on the RDF graph, compute the results of the corresponding cluster. The second—coarser—level of clustering, partitions the pattern graphs (and their results) based on the different types of construct (e.g., class, property, value) each query keyword matches. However, our approach does not exhaustively generate and cluster all the results and the hierarchy components. Instead, it benefits from relevance feedback at different levels of granularity to identify the pattern graphs which are relevant to the user intent. Our clustering hierarchy allows the user to disambiguate the query and compute the relevant results while examining only a small portion of the hierarchy components. To shorten the user interaction we devise ranking techniques for the hierarchy components that take into account structural and semantic information and occurrence frequency statistics. We design an algorithm which takes as input the matching constructs of the query keywords on the structural summary and compute pattern graphs forming r-radius Steiner graphs that involve these matching constructs. Our algorithm computes the pattern graphs on the structural summary without accessing the data graph. Only the selected pattern graphs are evaluated on the data, returning all and only the relevant results. This way, our system addresses efficiently the problems of relevant result identification and performance scalability. We implemented and experimentally evaluated our approach. Our results on measuring reach time show that the user can find the relevant pattern graphs in short time supported effectively by our ranking of the hierarchy components. They also showed that the system efficiently computes the required hierarchy components on the structural summary and evaluates the relevant pattern graph on the data.

1.3 Keyword Pattern Graph Relaxation to Retrieve Additional Relevant Results

In order to address the three major challenges of keyword search on graph data, recent approaches to keyword search developed techniques which exploit a structural summary of the data graph [22, 89, 90]. This is a concept similar to the 1-index [52] or data guide [33] in tree databases. The structural summary summarizes the structure of an RDF graph

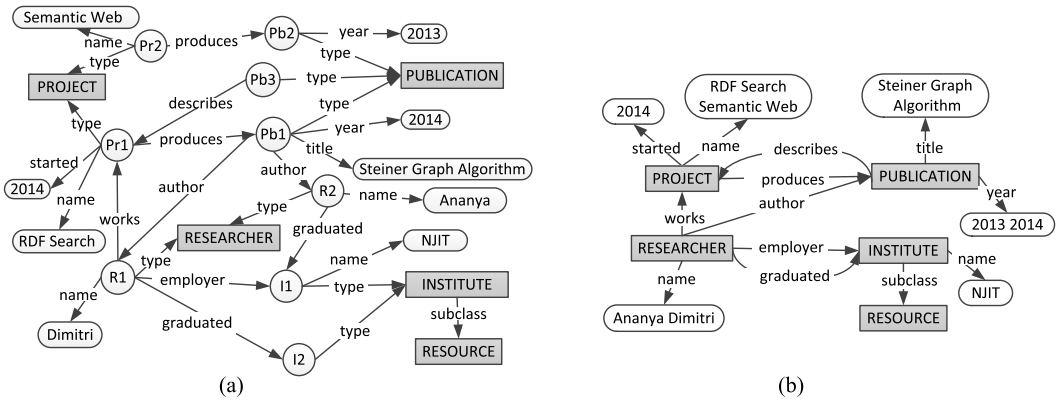


Figure 1.1 (a) An RDF data graph D, (b) The structural summary S_D of D.

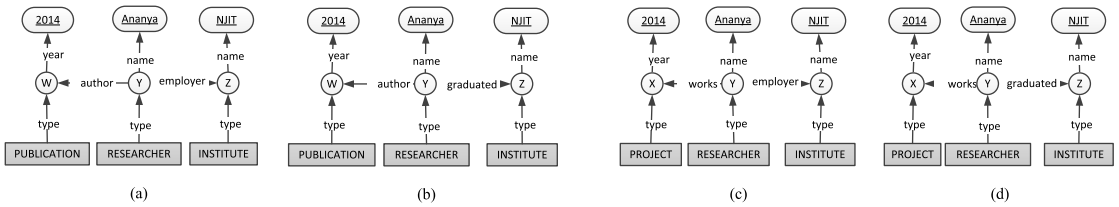


Figure 1.2 (a), (b), (c), and (d) Pattern graphs.

and associates inverted lists of keyword instances (extensions) with nodes. A structural summary is typically much smaller than its data graph. These aforementioned techniques use the structural summary to produce pattern graphs for a given keyword query. The pattern graphs are structured queries corresponding to alternate interpretations of an imprecise keyword query. By evaluating the pattern graphs on the data graph, the candidate results for the keyword query can be produced. Interestingly, a pattern graph can be expressed as a SPARQL query, and all the machinery of query engines and optimization techniques developed for SPARQL can be leveraged to efficiently evaluate the pattern graphs.

Example 1.1. *Consider the RDF graph D shown in Figure 1.1(a). This is a database about publications, projects, researchers and universities. Let us assume that the query* $Q = \{$`Ananya, NJIT, 2014`$\}$ *is issued against the database D. The user is looking for publications by* `Ananya` *published in* `2014` *which have an author working for* `NJIT`*. Figure 1.1(b) depicts the structural summary* S_D *of D. Finding the instances of the keywords* `Ananya, NJIT` *and* `2014` *on* S_D*, the system will construct pattern graphs. Dif-*

ferent algorithms can be employed for this task which aim at constructing pattern graphs by connecting the instances of the keywords in the structural summary in some minimal way [22, 32, 89, 90, 92]. All these algorithms will basically construct the pattern graphs shown in Figure 1.2. These pattern graphs are queries that can be matched against the data graph to produce answers for the queries (X,Y,Z are variable nodes to be matched against entity nodes in the data graph). Notice that these pattern graphs represent different interpretations of the given keyword query. For instance, the first pattern graph looks for publications by `Ananya`*, who works at* `NJIT`*, published in* `2014` *while the last one is looking for a project initiated in* `2014` *which employs a researcher named* `Ananya` *who graduated from* `NJIT`*. Evaluating these queries on D will return all the results for the initial keyword query Q on D.*

1.3.1 Benefits of the Structural Summary

A structural summary-based approach can resolve the challenges mentioned above of: (a) effectively identifying relevant results and (b) coping with the performance scalability issue. Indeed, the pattern graphs (structured queries) can be ranked using a scoring function and the top-k of them are presented to the user. As these structured queries represent different interpretations of the keyword query on the data graph, the user can choose one that meets his intention, and only the corresponding structured query is evaluated against the data graph [89, 90]. Our approach described in Chapter 4 exploits a hierarchical clustering of the pattern graphs. In order to select a relevant pattern graph the user chooses semantic interpretations for the query keywords and only the pattern graphs that correspond to these interpretations are generated and presented to the user [22]. Effectiveness studies show that the approaches based on the structural summary display good precision. Further, computing, ranking and identifying top-k subgraphs (query results) for a keyword query directly on the data graph is very expensive even when answers are computed in an approximate way [12, 48]. In contrast, since the structural summaries are much smaller than the actual data,

generating and manipulating relevant pattern graphs can be done efficiently. Therefore, the structural summary-based approaches scale satisfactorily and compute answers of keyword queries efficiently even on large RDF graphs stored in the external memory [21, 22, 89].

1.3.2 The Missing Relevant Result Problem

Despite its advantages, the structural summary-based approach for keyword search on RDF data has a drawback. The problem is that the pattern graph selected by the user might return no result when evaluated against the RDF graph even though results that match the user intent exist in the RDF graph. It is also possible that the pattern graph returns a non-empty answer but misses relevant results. This might happen even if a pattern graph is correctly selected by the user, that is, even if the selected pattern matches the user intent.

Example 1.2. *In our running example the pattern graph of Figure 1.2(a) is relevant and is selected by the user. One can see that this pattern graph does not have a match on the RDF graph D shown in Figure 1.1(a). Indeed,* `Ananya` *has authored a paper in 2014 but she does not work for* `NJIT`*. However, there is a result in the data graph D which matches the user intent since there is a publication authored by* `Ananya` *in* `2014` *which has an author (the co-author named Dimitri) working for* `NJIT`*. This relevant result cannot be directly obtained from any of the pattern graphs of Figure 1.2. It is missed by the structural summary-based approach.*

As another example consider the keyword query $Q =$*{*`publication, describes, project, produces, Steiner`*} on the RDF graph D. The user is looking for a* `publication` *which* `describes` *a* `project` *that* `produces` *a paper titled* `Steiner` *graph algorithm. The structural summary-based approach will generate one pattern graph shown in Figure 1.3(a). As one can easily see, this pattern graph does not have a match on D. However, there is a result in D which matches the user intent. Figure 1.3(b) shows a subgraph of D which reflects this result. This relevant result is again missed by the structural summary-based approach. The reason for this discrepancy is that*

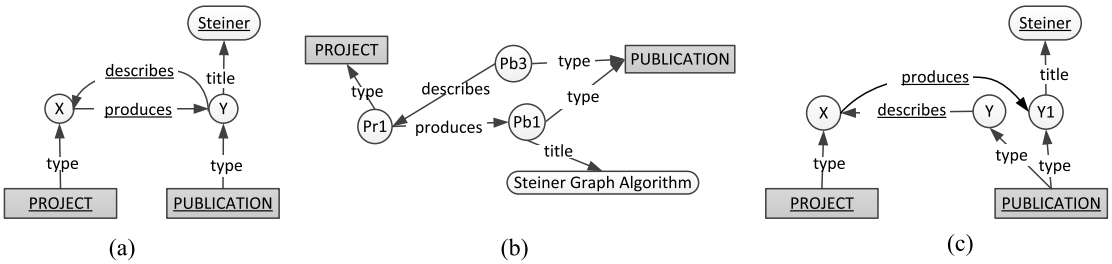


Figure 1.3 (a) Pattern graph (b) Result graph (c) Relaxed pattern graph.

the structural summary merges entity vertices of the same type of the RDF data graph into one vertex and this coarse representation loses information as to how an entity vertex is related to other entity vertices or is assigned to properties and values.

1.3.3 Our Solution: Keyword Pattern Graph Relaxation

In this dissertation, we provide an approach for keyword search over RDF graph data which addresses the weakness of the structural summary-based approach while maintaining its advantages. Our system allows the user to navigate through a clustering hierarchy to select a relevant pattern graph. It then enables the gradual relaxation of this pattern graph so that additional results of interest to the user are retrieved from the RDF graph, if needed (for example, if the original pattern graph returns no result or if the user wants to extract more semantically similar results). For instance, in the example of Figure 1.3, our system will produce the relaxed pattern graph of Figure 1.3(c) from the pattern graph of Figure 1.3(a) which can retrieve the relevant result shown in Figure 1.3(b) missed by the original pattern graph of Figure 1.3(a).

In order to define relaxed pattern graphs we leverage pattern graph homomorphisms. Relaxed pattern graphs can expand the result space of an original pattern graph. They can produce additional results of possible interest to the user based on her choice of an original pattern graph. We define an operation on pattern graphs in order to allow the construction of relaxed pattern graphs. A vertex split operation creates a split image of an entity variable vertex in a pattern graph and partitions its incident edges between the two

vertices in order to increase the chances for the relaxed pattern graph to have embeddings to the RDF graph. We show that this operation is complete, that is it can produce all the relaxed pattern graphs. Since we want to relax a pattern graph so that it is as close to the initial pattern graph as possible, we introduce three metrics of decreasing importance to measure the degree of relaxation of a pattern graph. All three metrics take into account structural and semantic characteristics of the relaxed pattern graph and depend on the vertex split operations applied to the original pattern graph. If an original pattern graph has an empty answer on an RDF graph, we would like to identify its vertices which contribute to this condition. These are vertices which if not split, the relaxed pattern graph will keep producing an empty answer. We call these vertices empty vertices and we provide necessary and sufficient conditions for characterizing them in a pattern graph. Empty vertices are used to guide the relaxation process so that relaxed pattern graphs with non-empty answers are produced. We design an algorithm which takes a pattern graph as input and gradually generates relaxed pattern graphs having non-empty answers. The algorithm returns the relaxed pattern graphs (and computes their answer on the RDF graph) in ascending order of relaxation as this is defined by the three relaxation metrics mentioned above. We run experiments to measure the effectiveness of our ranking of relaxed pattern graphs and the efficiency of our system in computing relaxed pattern graphs and their answers.

1.4 Cohesive Keyword Queries: As Simple As Traditional Keyword Queries

The structural summary-based approaches described above, though promising, require interaction with the user who has to select clusters of results and possibly navigate in a clustering hierarchy in order to disambiguate the keyword query and eventually retrieve the answer relevant to her query intent. We claim that existing techniques to keyword search on RDF data are not sufficient to produce results of high quality without additional information from the user. Current RDF graphs are very large and integrate data from various application domains. Query keywords on RDF graphs can have not only numerous in-

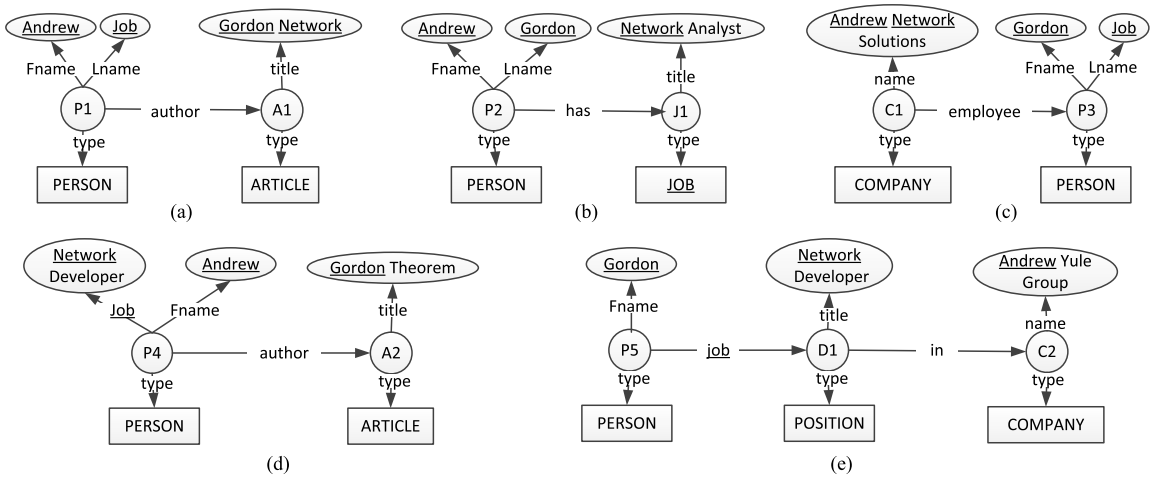


Figure 1.4 Result graphs of the query $Q =$ `(Andrew Job Gordon Network)`.

stances of the same type, but also numerous instances of different types. For example, the keyword `job` can be the name of different people, the name of an employment agency, a relationship between a person and his occupation, the name of a property of an entity person, the name of a RDF class etc. These multiple instances generate a multitude of result graphs corresponding to different interpretations of the user query.

Example 1.3. *Consider the keyword query* $Q =$ `(Andrew Job Gordon Network)`. *With this query the user is looking for an article authored by* `Andrew Job` *on* `Gordon Network`. *Figure 1.4(a) shows a result graph which corresponds to the user intent. Underlined words in a result graph indicate the instances of the keywords of Q. However, existing algorithms [22, 32, 60, 89, 90, 92] will also compute other result graphs like those shown in Figure 1.4(b), (c), (d) and (e) which correspond to different undesirable interpretations of the query. These result graphs associate the keywords in a way which is not the one intended by the user. For instance, the result graph of Figure 1.4(b) represents a person named* `Andrew Gordon` *who has a* `job` *of a* `Network` *analyst.*

Using scoring functions based on statistical information to rank the results might help but certainly cannot by itself disambiguate the different interpretations and rank on the top position the result graphs that correspond to the user intent. Indeed, the result graph

that matches the user intent might be one with low probability which loosely correlates the keyword instances.

To address this problem, we introduce the concept of *cohesive group* of keywords in a query. The keywords of a cohesive group are to be interpreted as forming a cohesive whole. That is, the instances of the keywords outside the group should not "penetrate" the subgraph defined by the instances of the keywords of the group in the result graph. Cohesive groups of keywords are specified naturally and effortlessly by the user while formulating the query. For instance, the previous example query Q can be written as `((Andrew Job) (Gordon Network))`, where parentheses are used to delimit the cohesive groups `(Andrew Job)` and `(Gordon Network)`. This cohesive query excludes the graphs of Figures 1.4(b), (c), (d) and (e) from the set of legal result graphs since, as we explain in more detail later, they breach the cohesiveness of the specified cohesive groups. As an example, the keyword instance `Job` penetrates the subgraph defined by the instances of the keywords `Gordon` and `Network` in the result graph of Figure 1.4(e). Since result graphs not intended by the user are excluded, the precision of the answer improves. Further, the evaluation time of the query is reduced since the system does not waste time computing unwanted results. These benefits are obtained thanks to the grouping of the keywords provided effortlessly by the user.

Cohesive groups can be nested within other cohesive groups in a query. An example is the keyword query `(((Person (Andrew Job)) (Gordon Network))`. Further, in contrast to flat keyword queries, cohesive keyword queries can contain repeated keywords provided they occur in different cohesive groups. For instance, the keyword query `((Andrew Job) job (Network Analyst))`. These features of cohesive queries increase their expressive power and their capacity to narrow down the search to relevant results by excluding irrelevant ones. It is important to note that the user can naturally specify cohesive groups. In fact, cohesive queries offer more flexibility to the users and allow them to express queries with a clearer meaning. For instance, the user who is

looking in a bibliographic data source for a paper authored by Johns Smith and edited by s Brown can naturally formulate the query `((author (Johns Smith)) (editor (Johns Brown)))`. In summary, cohesive queries are intuitive and as simple as flat keyword queries while retaining both advantages of flat keyword search: the user does not need to know any query language and he does not need to have knowledge of the structure of the data sources.

Note that in IR, flat, document-based search engines like Google allow the user to search for whole phrases (sequence of keywords) by enclosing them between quotes to improve the accuracy of the search. Cohesiveness queries also aim at improving the accuracy and execution time but they are different and more flexible than phrase matching over flat documents since they are designed for data with some form of structure and they do not impose any order on the keywords. The user naturally groups the keywords in a cohesive query based on the associations she wants to express on them and she is not required to know how these keywords are sequenced in the dataset.

In this dissertation, we formally define cohesive keyword queries which involve cohesive groups of keywords and allow keyword group nesting and keyword replication. Cohesive queries can better express the user intent. They are as simple as flat keyword queries and they can be formulated naturally and effortlessly by the user. We provide semantics for cohesive queries on RDF graphs which interprets cohesive keyword groups in a query as cohesive units. This means that the instances of the query keyword occurrences which are not in the cohesive group cannot penetrate and be part of the subgraph of a query result graph representing the cohesive group. We design an algorithm to efficiently evaluate cohesive queries. Our algorithm exploits the structural summary of the RDF graph to compute pattern graphs which are r-radius Steiner graphs with a minimal r. The pattern graphs are structured queries representing alternative interpretations of the cohesive queries and can be evaluated against the RDF data graph to produce the query results. The algorithm constructs pattern graphs incrementally excluding early on graphs under con-

struction which violate the cohesiveness of keyword groups. We ran experiments to assess the effectiveness of cohesive queries and the efficiency of our algorithm. Our results show that the pattern graphs of cohesive queries can be computed by our algorithm much faster than the pattern graphs of flat keyword queries. Cohesive keyword queries considerably improve the quality of the results compared to flat keyword queries, importantly reducing the number of pattern graphs returned to the user.

1.5 Diversification of Keyword Search Results

As we have already discussed, keyword queries are ambiguous and allow for multiple interpretations. For instance, a user issuing the query “apple” could be interested in searching about the fruit “apple,” the American multinational technnology company selling consumer goods and computer products, or the second largest chartered savings bank in New York State. If results are returned to the user based on the most plausible interpretation of the query (in this case “apple” as technology company) then there is an inherent risk of leaving the user who is interested in “apple” bank or in “apple" fruit unsatisfied. This problem is known as the over-specialization problem [80]. Diversifying the results retrieved for a keyword query could be a meaningful solution to this problem. By introducing diversity in the result set, the search mechanism can maximize the user’s chance of finding at least one of the retrieved results relevant to her intent [19]. Additionally, even if a keyword query has a single, clearly defined interpretation, it can still be under-specified to some extent. For example, a user searching for “apple electronics” may be interested in laptops, desktops, or the best selling apple electronics, or sale on apple electronics, or service centers for apple electronics. Therefore, another motive for diversifying search results is to cover different aspects of the entire result space and enable the user to explore and find desired results [29].

Search result diversification is a well-studied problem in both Information Retrieval and Recommendation Systems [29, 35]. However, there is a limited amount of work on diversification of keyword search on RDF graph data. There is a proliferation of RDF

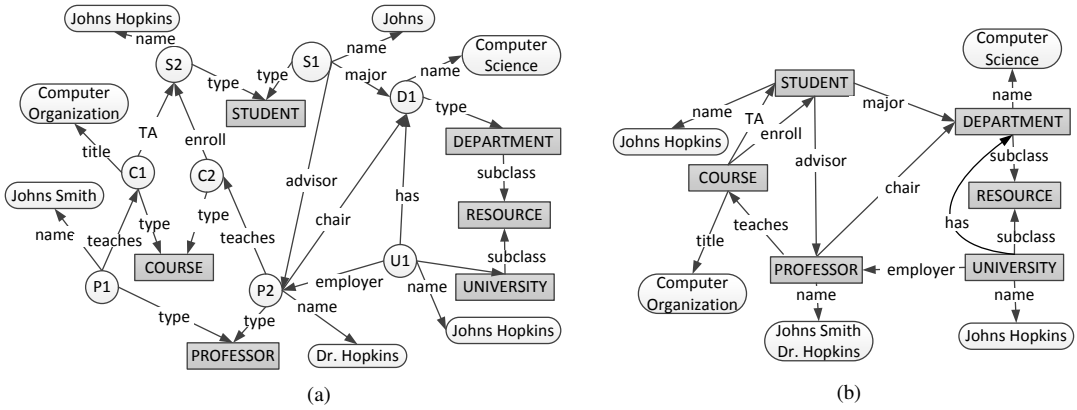


Figure 1.5 (a) An RDF graph D, (b) The Structural Summary of S.

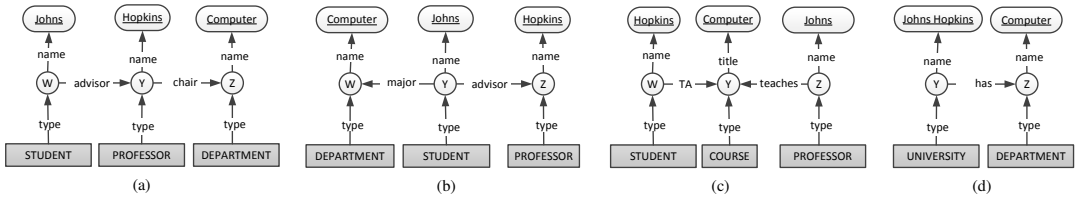


Figure 1.6 (a), (b), (c) and (d) are different patterns graphs of the keyword query $Q = \{$ `Johns, Hopkins, Computer` $\}$.

repositories in recent years, and keyword search is the most popular technique for querying linked data on the Web. While ranking ensures the most relevant results are ranked on top for a given keyword query, it is often the case that the top results tend to be homogeneous, making it difficult for users interested in less popular aspects to find relevant results. Thus, result diversity can play a big role in ensuring that the users get a broad view of the different aspects of the results and in guarantying that most users can find relevant results to their queries in the top ranks.

As an example, consider the RDF data graph D in Figure 1.5(a) and its structural summary S in Figure 1.5(b). Consider also the keyword query $Q = \{$`Johns, Hopkins, Computer`$\}$. Figures 1.6(a)-(d) show the pattern graphs for Q on S, corresponding to alternate interpretations of Q. For instance, the pattern graph of Figure 1.6(a) interprets "Johns" as a student advised by Professor "Hopkins," a chair of a department whose name involves "Computer." The pattern graph of Figure 1.6(d) views "Johns Hopkins" as the name of a University which has a department whose name involves "Computer." Every

pattern graph in Figure 1.6 interprets the query Q differently. However, if one has to select out of those pattern graphs in Figure 1.6 a subset that balances both diversity and relevance, it is important to characterize the diversity of a set of pattern graphs. This can be done by quantifying the semantic dissimilarity of pairs of pattern graphs. In this case, comparing the pattern graphs of Figures 1.6(a) and 1.6(b), one can see that the keywords in both patterns graphs have the same semantics, that is , "Johns" is a student, "Hopkins" is a professor and "Computer" is a department. On the other hand, by comparing the pattern graphs of Figures 1.6(a) and 1.6(c), one can see that none of the keywords in the two pattern graphs have the same meaning. Therefore, the set of pattern graphs of Figures 1.6(a) and (c) is more diverse than the set of the pattern graphs of Figures 1.6(a) and (b).

In this dissertation, we propose a novel technique for diversifying keyword search results on RDF graph data. We formulate the diversification problem as an optimization problem over pattern graphs (structured queries) representing alternate interpretations of a keyword query. This is because, the computation of pattern graphs is cost efficient in comparison to the computation of all the relevant result graphs. Our diversification approach aims at selecting a set of pattern graphs that balances relevance and diversity. Returning to the users pattern graphs instead of plain result graphs already secures a certain degree of diversification of the query results since the pattern graphs are clusters of results with the same structure and semantics. In order to measure the relevance of a pattern graph to a keyword query, we devise a metric based on the popularity score of individual elements of the pattern graph. We also present a technique for assessing the semantic distance between pattern graphs. We design an algorithm that employs greedy heuristics for computing top-k pattern graphs trading off relevance and diversity. Finally, we implement and experimentally evaluate the efficiency of our algorithm and the effectiveness of our proposed measures for estimating the relevance and the diversity of a set of pattern graphs.

1.6 Organization

This dissertation is organized as follows: Chapter 2, reviews the state-of-the-art for keyword search on RDF data and the various categories of keyword search techniques on both structured and semi-structured data. Chapter 3, provides the definition of the data model we adopt and formally introduces different concepts that we extensively use in the dissertation. Chapter 4 presents a semantic hierarchical result clustering technique to support keyword search on RDF data graphs. In Chapter 5, we introduce our approach for keyword pattern graph relaxation. In Chapter 6, we propose our cohesive keyword query language. In Chapter 7, the problem of diversification of keyword search results on RDF graph data is addressed. Finally, Chapter 8 concludes the dissertation and provides direction for future work.

CHAPTER 2

STATE OF THE ART

This chapter provides a background on keyword search on RDF graph data. It also reviews the keyword-search based approaches proposed for both structured and semi-structured data which are most relevant to our work.

2.1 Search on RDF Data

This section briefly discusses the RDF data model and the two major types of search practices on RDF data.

2.1.1 Resource Description Framework

The Resource Description Framework (RDF) is a data model for representing information about various resources in the World Wide Web. Resources are identified using *URIs (Uniform Resource Identifiers)*[1]. RDF consists of W3C[2] recommendations that enable the encoding, exchange and reuse of structured data, providing means for publishing both human-readable and machine-processable vocabularies. Two possible representations of RDF data are labeled graphs and triple sets. *A triple* consists of three elements: the *subject*, the *predicate* and the *object*. A set of RDF triples forms an *RDF graph*. The RDF Schema (RDFS) is a language for defining vocabularies for modeling RDF graphs. The RDFS vocabulary includes concepts (e.g., *classes, properties, entities, relationships*) which are used to describe groups of related resources in a domain modeled by an RDF graph. The RDF graph model adopted in this dissertation is described in Chapter 3 in detail.

[1] http://labs.apache.org/webarch/uri/rfc/rfc3986.html[RFC3986] (accessed on April 24, 2016)

[2] http://www.w3.org/ (accessed on April 24, 2016)

2.1.2 Structured Queries on RDF Data

During the past years, many structured query languages have been proposed for retrieving information from the RDF data model. They include RQL [51], RDQL [83], SeRQL [16], and TRIPLE [87]. The most noteworthy among all the RDF query languages is SPARQL [76] which became the official W3C recommendation language for querying RDF data in 2008. Using the SPARQL query language for RDF graphs, it is possible to extract information about both the data and the schema. The basic task in SPARQL query language is to match graph patterns against the RDF data graph. The simplest graph patterns are triple patterns, which are like RDF triples except that any of the subject, predicate or object position of the triples can hold variables. A query that contains a conjunction of triple patterns is called a basic graph pattern. A basic graph pattern matches a subgraph of the RDF data graph when the node or edge labels from the subgraph may be substituted with the variables of the graph pattern. The syntax of SPARQL follows an SQL-like select-from-where paradigm.

2.1.3 Keyword Queries on RDF Data

In recent years, a number of papers address keyword search on graph data. However, the approaches that are proposed for generic graphs cannot be used directly for keyword search over RDF graph data. This is because the edges of an RDF graph represent predicates, which can also be matched by the keywords of a keyword query. The approaches proposed for keyword search on RDF data can be classified into two categories: (a) data-based approaches [30, 31] and (b) schema-based approaches [22, 32, 60, 74, 89, 90, 92]. Data-based approaches rely on the data graph to produce answers. Although these approaches generate precise answers, they fail to scale well when the size of the data increases. In contrast, summary-based approaches rely on a reduced size structural summary extracted from the data. In order to compute answers, these approaches focus on capturing the interpretations of a keyword query by mapping the keywords to elements of the structural

summary and constructing pattern graphs. Given that keyword search is ambiguous, these approaches often exploit relevance feedback from the users in order to identify the users' intent [22, 47, 89]. A hierarchical clustering mechanism and user interaction at multiple levels of the hierarchy can be used to facilitate disambiguation of the keyword query and to support the computation of the relevant results. Such a mechanism is suggested in [6, 67] in the context of tree data. In this dissertation, a semantic clustering mechanism is proposed in the context of RDF data [22]. Although summary based approaches proved to have better performance scalability compared to data-based approaches, they provide an approximate solution and they might miss relevant results for a given keyword query. As RDF data graphs are practically schema free, a summary graph extracted from an RDF graph cannot capture completely all the information in the RDF graph.

In Chapter 5, we provide a pattern graph relaxation technique [23] to address this issue. Relaxation techniques are studied in [7, 15, 54] in a context different than ours since they are applied to queries over tree (XML) data. Further, their goals and processes are different: [7] relaxes weighted tree pattern queries with descendant edges in order to permit approximate matching on XML data. [15] provides a framework for generating similar satisfiable queries, when the user tree pattern query is unsatisfiable. [54] relaxes the MaxMatch semantics [69] of keyword queries on XML data so that they also return LCA (Lowest Common Ancestor) nodes which are not SLCA (Smallest Lowest Common Ancestor) nodes. In contrast, we deal with pattern graphs and RDF graph data and we relax pattern graphs by splitting vertices in order to produce and rank relaxed patterns graphs that are semantically close to the original pattern graph. Though the above approaches are promising, they require interaction with the user who has to select clusters of results and possibly navigate in a clustering hierarchy.

In this dissertation, we claim that existing techniques to keyword search on RDF data are not sufficient for producing results of high quality without additional information from the user. This is because, unlike structured queries, traditional keyword queries do

not provide a way to the user to express his intent. To address this issue, we introduced in Chapter 6 cohesive keyword queries on RDF data [24]. Previous research work tries to retain the expressive power of structured queries while incorporating the flexibility of flat keyword queries [65, 75]. In [65], an entity relationship query language over unstructured document data is introduced. Unlike our cohesive keyword query language which does not follow a strict structure and allows a user to group keywords and terms, the query language in [65] follows a strict structure and allows keywords only to express entity properties and relationships. In [75], a keyword-based structured query language is presented, to be used over structured knowledge bases extracted from the Web. The main goal of this work is to extract entities from the knowledge base (possibly in conjunction with the relevant text documents). Although the desire of trading off flexibility and convenience for expressivity is common with our work, the structured queries in [75] are schema dependent: the user needs to characterize some keywords as relations in order to build nested structured queries beyond flat keyword queries. In contrast, in our query language, nesting is incorporated in queries based on the desire of the user to form cohesive groups irrespectively of any schema information. Cohesive queries were also introduced in [27]. However, those queries are for tree-structured data. Therefore, their semantics is different than ours as it does not involve any semantic information.

All the proposed techniques for keyword search on RDF graphs mentioned above return the most relevant results, in the form of graphs or trees. The relevance of the results in these cases are weighed in terms of: (a) content similarity between the elements that comprise a result and the query terms, and (b) result compactness (smaller trees or graphs are considered more relevant). As a consequence, the result sets that are returned to the users are often characterized by a high degree of redundancy. Furthermore, significant information might be compromised since graph paths connecting two entities that denote a significant relation between them might be omitted to satisfy the compactness requirement. Moreover, general graph-based approaches do not consider the rich structure and seman-

tics provided by the RDF data model. An effective RDF keyword search method should also give equal importance to the RDF properties (edges) as they might provide significant information about the relations between the entities being searched. A way of addressing these drawbacks is to introduce the step of diversification into the result retrieval and ranking process in order to return to the users a result set which is more meaningful and informative.

The diversification problem has been extensively studied both in Information Retrieval [80] and recommendation systems [94, 95, 96]. The goal is to solve the over-specialization problem [29] where a highly homogenous set of results is returned to the user due to relevance-based ranking and/or personalization. Result diversification is a way to minimize user dissatisfaction by providing a diverse set of results. In general, the diversification problem is defined as selecting a subset of the retrieved result set with k results such that the diversity among these k results is maximized. This is achieved in different ways. In [17], the concept of maximal marginal relevance (MMR) is used to tradeoff between relevance and novelty. In [35] an axiomatic approach for result diversification is adopted. The concept of query expansion is employed in [71] for search result diversification. Reference [29] is a review of different definitions of diversity, and the algorithms and evaluation metrics for diversification. It categorizes diversity definitions as content-based [96], novelty-based [20, 95] and coverage-based [2]. Most of these approaches perform the diversification as a post-processing or re-ranking step of candidate result retrieval. Unfortunately, in the context of databases, the post-retrieval diversification process can incur a huge computation cost since the number of candidate results can be extremely large for a keyword query. In contrast, our diversification process is part of a query disambiguation phase which takes place before extracting any search results. We compute pattern graphs corresponding to alternate interpretations of a given keyword query since they offer clear semantics and quality information for diversification. Additionally, this way we avoid, the computation overhead of computing all relevant results.

There are a few contributions on search result diversification on structured and semi-structured databases which include [13, 25, 38, 64, 70]. However, diversifying search results on RDF data is an open problem. Previous techniques cannot be directly applied in this context as the semantic information in RDF data graphs requires different criteria and methods. In this dissertation, we exploit structural and semantic characteristics of pattern graphs for capturing the relevance of a pattern graph to a query, and also the similarity and dissimilarity between pattern graphs. We use these measures to generate a set of pattern graphs for a keyword query which trades off relevance and diversity.

2.2 Keyword Search Approaches on Structured and Semi-structured Databases

This section provides a brief survey on the literature related to relevant keyword search approaches introduced in the context of both structured and semi-structured databases.

2.2.1 Form of the Answer

When a keyword query is issued to a Web search engine, a ranked set of Web pages containing the keywords in the query are returned to the user as result. In contrast, keyword queries on structured and semi-structured data return results which have structure. The structure of these results reflect semantic information about how the query keywords relate to each other and provide possible interpretations of the results. These structured results are generally a substructure of the underlying database and are in form of a tree [3, 12, 28, 34, 39, 42, 43, 45, 48, 66, 72, 73, 77, 79], or a graph [50, 63, 78, 86] or just a node/entity [1, 9, 56, 61, 91]. Some approaches in the context of RDF graph data compute structured queries corresponding to different interpretation of the keyword query and allow the user to choose among them the one that expresses his query intent [88, 89]. In this dissertation, we defined a form of result that has graph structure and we cluster result graphs into pattern graphs. These pattern graphs are structured queries corresponding to different

possible interpretations of the keyword query [22, 23, 24].

2.2.2 Schema-agnostic Approaches

Schema-agnostic approaches on structured databases model tuples to nodes and edges to primary-foreign key dependencies [12, 9, 39, 45, 48, 63]. In RDF, the data graph is implicitly formed from the RDF triples [56, 63, 91]. These approaches explore the data graph and retrieve subtrees/ subgraphs connectinng all the keywords of the query. It is to be noted that in contrast to structured databases, in a semi-structured database setting there are a lot of work which adopt a schema-agnostic approach. This is expected since unlike semi-structured data, the data stored in relational databases must adhere to a specific schema. The drawbacks of schema-agnostic techniques are that they are prone to produce a plethora of candidate results posing a challenge in identifying relevant ones and they do not scale satisfactorily with a growing size of the data and/or the number of keywords in a query. Considering these shortcomings of schema-agnostic techniques in this dissertation we adopted schema-aware techniques for effective keyword search on RDF graph data.

2.2.3 Schema-aware Approaches

Schema-aware approaches on relational databases model the data as a graph based on their schemas. In the literature of structured databases schema-aware approaches are studies and adopted extensively [3, 42, 43, 72, 66, 73, 77, 79]. In RDF, schema-aware approaches create a structural summary by mapping RDF classes to nodes and the properties between the entities of two classes to edges between these class nodes labeled by the property names [61, 88, 89]. The process of keyword query evaluation with schema-aware approaches involve two phases. The first phase involves finding the schema elements that matche the query keywords and generating the schema for all possible possible results of the query in the form of pattern graphs. These pattern graphs can be expressed as structured queries

(e.g., SPARQL). The second phase consists of evaluating these structured queries over the data graph to retrieve query answers. In this dissertation, we adopted schema-aware approach for keyword search on RDF graph data as the structural summary of an RDF graph is typically much smaller that its corresponding data graph. This approach allows for an efficient exploration of the structural summary and generation of all possible pattern graphs (structured queries). Furthermore, we take advantage of relevance feedback from the user in order to disambiguate imprecise keyword queries and to return high quality relevant results [22, 23].

2.2.4 Keyword Query Evaluation Algorithms

Several algorithms were proposed to explore the data and generate tree or graph structured results connecting the keywords in a query. In [12], a backward search algorithm, called BANK is presented for finding Steiner trees. The Steiner tree problem is superficially similar to the minimum spanning tree problem. For a given set of vertices, a Steiner tree interconnects these vertices by paths of shortest length, where the length is the number of edges. The difference between the Steiner tree problem and the minimum spanning tree problem is that, in the Steiner tree problem, extra intermediate vertices and edges may be added to the tree in order to reduce the length of the spanning tree. These new vertices introduced to decrease the total length of connection are known as Steiner points or Steiner vertices. The problem of finding Steiner trees is NP-complete. Different techniques are used to work around NP-completeness. In [28], a dynamic programming approach applicable to only few keywords and having an exponential time complexity is employed. In [34], a polynomial delay algorithm is introduced. The algorithm in [48] produced trees rooted at distinct vertices. This algorithm was supplemented by BLINK [39] with an efficient indexing structure. Tree-based methods produce succinct answers but answers from graph-based methods are more informative.

A recent graph-based approach [63] computes all possible r-radius Steiner graphs

and indexes them. An $r-$radius Steiner graph is a graph, which has a connecting vertex and the length of the path from the connecting vertex to any of the vertices in the graph is r or less. The method in [63] is prone to produce redundant results since it is possible that a highly ranked r-radius Steiner graph is included in another Steiner graph having a larger radius. The algorithm in [78] finds multi-centered subgraphs called communities containing all the keywords, such that there exists at least one path of distance less than or equal to R_{max} between every keyword instance and a center vertex. Later in [50], r-cliques containing all the keywords are found such that the distance between any two keyword matching vertices is no more than r. Finding r-clique with the minimum weight is an NP-hard problem. Hence, the authors provided an algorithm with polynomial delay to find the top-k r-cliques where r is an input to the algorithm. Predicting an optimal r for producing r-cliques is a challenge because it is possible that there exists no clique with that r or less. Unlike traditional graph-based keyword searh, on RDF graphs, keywords can match both a vertex and an edge label of the graph [89, 30]. In this dissertation we define meaningful subgraphs of the RDF data graph as results, and adopted and extended the r-radius Steiner graph algorithm in the context of RDF graphs to generate structured queries over the structural summary of the data graph [22, 23, 24].

2.2.5 Relevance Assessment of the Results

It is always a challenge to identify among a plethora of candidate keyword search results, the ones those are relevant to the user. This is because keyword search is inherently ambiguous. Various ranking schemes are proposed in the literature, which consider both the properties of the data nodes (e.g., tf/idf and complex measures adopted from IR, node/edge weight, ranking in the style of page rank) and the properties of the whole query result (e.g., path length, number of nodes/edges, weight of nodes/edges, size normalization) [3, 12, 34, 39, 42, 43, 45, 48, 63, 66, 72, 89, 91] in order to sort the list of the retrieved results hoping that only the top results in the sorted list are relevant to the user. In [12, 45],

the results are ranked using the notion of proximity coupled with a notion of prestige of nodes based on incoming links, similar to Web search techniques. Contributions such as [39, 63, 66, 72, 89] employ a number of IR measures to rank the results. IR techniques built-in in RDBMS are exploited by [39, 72], while in [66] they are extended to include novel and more fine-grained scoring measures (e.g., tree normalization, inter-document frequency normalization, different scoring schemes for schema and value matching keyword term).

A novel index which materializes tf/idf-based IR rankings is proposed by [63]. In contrast, [89] employs the Lucene index to textual ranking. The authors of [72] consider results which are relevant to any subset of the query keywords. Some approaches [66, 72] use non-monotonic functions, in which the score of an answer is independent to its components for top-k query processing. In this dissertation, we proposed novel ranking techniques considering both structural characteristics and semantics of RDF data to rank the retrieved results. We adapted the ranking techniques discussed in [89] to our context considering statistical information about the data [22, 24]. We also introduced a new paradigm for ranking the relaxed pattern graphs obtained by relaxing an original pattern graph chosen by the user as the most relevant to his intent [23].

CHAPTER 3

RDF DATA MODEL AND QUERY LANGUAGE SEMANTICS

In this chapter, we formally define the RDF data model that we have considered in this dissertation. We also introduce and provide definitions for different novel concepts (e.g., keyword matching constructs, query signature, inter-construct connections). We define query results as meaningful subgraphs of the data graph. Furthermore, we discuss and define the extraction of structural summary graph from the RDF data graph. Finally, we introduce the concept of pattern graphs which are structured queries on the structural summary and they represent different interpretations of a keyword query.

3.1 RDF Data Model

Resource Description Framework (RDF) provides a framework for representing information about web resources in a graph form. RDF vocabulary includes elements, that can be broadly classified into Classes, Properties, Entities and Relationships. All the elements are resources. RDF has a special class, called *Resource* class and all the resources that are defined in an RDF graph belong to the *Resource* class. Our data model is an RDF graph defined as follows:

Definition 3.1.1 (RDF Graph)**.** An *RDF graph* is a quadruple $G = (V, E, L, l)$ where:

- V is a finite set of vertices, which is the union of three disjoint sets: V_E (representing entities), V_C (representing classes) and V_V (representing values).
- E is a finite set of directed edges, which is the union of four disjoint sets: E_R (inter-entity edges called *Relationship* edges which represent entity relationships), E_P (entity to value edges called *Property* edges which represent property assignments), E_T (entity to class edges called *type* edges which represent entity to class membership)

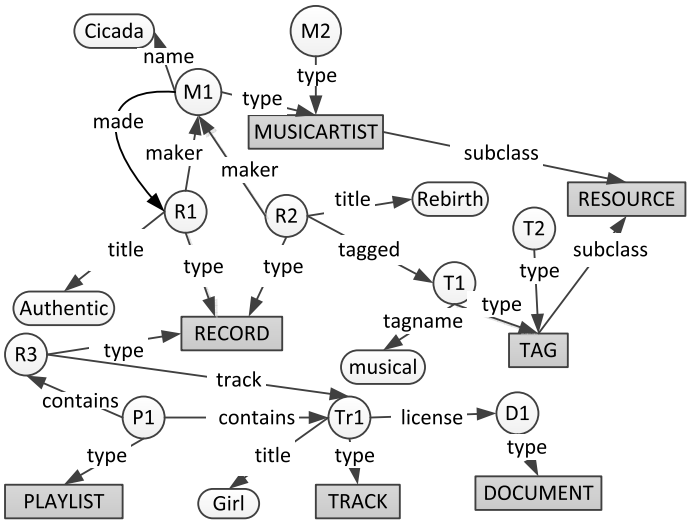


Figure 3.1 An RDF graph.

and E_S (class to class edges called *subclass* edges which represent class-subclass relationship).

L is a finite set of labels that includes the labels "type," "subclass" and "resource."

l is a function from $V_C \cup V_V \cup E_R \cup E_P$ to L. That is, l assigns labels to class and value vertices and to relationship and property edges.

Entity and class vertex and edge labels are Universal Resource Identifiers(URIs). Vertices are identified by IDs which in the case of entities and classes are URIs. Every entity belongs to a class. Figure 3.1 shows an example RDF graph (a subgraph of the Jamendo Dataset [1]).

3.2 Queries and Answers

A *query* is a set of keywords. The *answer* of a query Q on an RDF graph G is a set of subgraphs (*result graphs*) of G, where each result graph involves at least one instance of every keyword in Q. A *keyword instance* of a keyword k in Q is a vertex or edge label containing

[1] http://dbtune.org/jamendo/ (accessed on April 24, 2016)

k. In order to facilitate the interpretation of the semantics of the keyword instances, every instance of keyword in a query is matched against a small subgraph (*matching construct*) of the graph G which involves this keyword instance. Each matching construct provides a deeper insight about the context of a keyword instance in terms of classes, entities and relationship edges. We link one matching construct for every keyword in the query Q through edges (*inter-construct connection*) and common vertices into a connected component to form a *result graph*. A query Q can have multiple signatures, representing different interpretations for the keywords. Each signature can generate multiple result graphs.

Next, we provide definitions for all the important concepts of keyword matching constructs, query signature, an inter-construct connection and result graph in order to formally define a query answer.

Definition 3.2.1 (Matching Construct)**.** Given a keyword k of a query and an RDF graph G, for every instance of k in G, we define a *matching construct* as a small subgraph of G. If the instance i of k in G is:

- the label of a class vertex $v_c \in V_C$, the matching construct of i is the vertex v_c (*class matching construct*).

- the label of a value vertex $v_v \in V_V$, the matching construct of i comprises the value vertex v_v, the corresponding entity vertex, and its class vertices along with the property and type edges between them (*value matching construct*).

- the label of relationship edge $e_r \in E_R$, the matching construct of i comprises the relationship edge e_r, its entity vertices and their class vertices along with the type edges between them (*relationship matching construct*).

- the label of property edge $e_p \in E_P$, the matching construct of i comprises the property edge e_p, its value and the entity vertices, and the class vertices of the entity vertex along with the type edges between them (*property matching construct*).

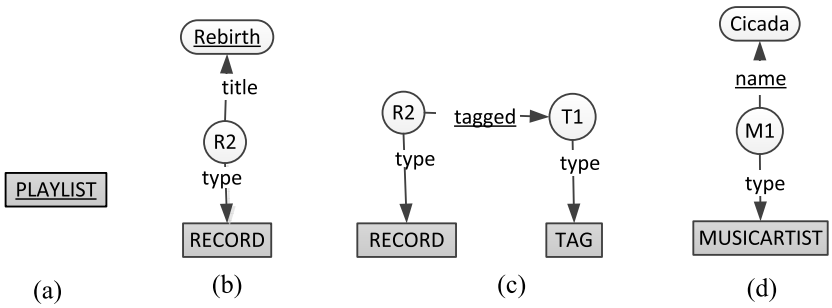


Figure 3.2 Matching constructs (a) class (b) value (c) relationship (d) property.

Figures 3.2(a), (b), (c) and (d) show a class, value, relationship and property matching construct, respectively, for different keyword instances in the RDF graph of Figure 3.1. Underlined labels in a matching construct denote the keyword instances on which the matching construct is defined (called *active keyword instances* of the matching construct).

Definition 3.2.2 (Query Signature)**.** Given a query Q and an RDF graph G, a *signature* of Q is a function from the keywords of Q that matches every keyword k to a matching construct of k in G.

Figures 3.2(a), (b), (c) and (d) show a query signature for the query *{Playlist, Rebirth, tagged, name}*. Note that a signature of a query Q can have less matching constructs than the keywords in Q, since one matching construct can have more than one active keyword instance.

Definition 3.2.3 (Inter-construct Connection)**.** Given a query signature S, an *inter-construct connection* between two distinct matching constructs C_1 and C_2 in S is a simple path augmented with the class vertices of the intermediate entity vertices in the path (if not already

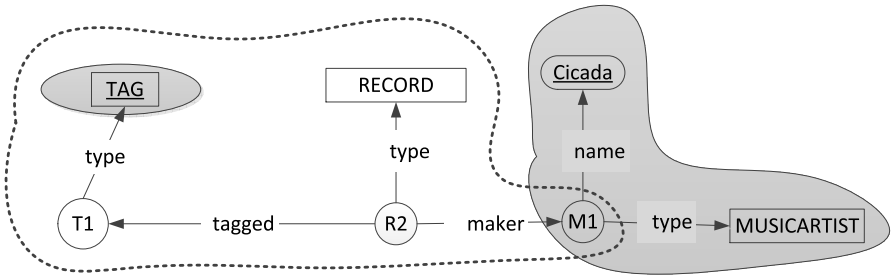


Figure 3.3 Inter-construct connection.

in the path) such that: (a) one of the terminal vertices in the path belongs to C_1 and the other belongs to C_2, and (b) no vertex in the connection except the terminal vertices belong to a construct in S.

Figure 3.3 shows an inter-construct connection between the matching constructs for keywords *Tag* and *Cicada* in the RDF graph of Figure 3.1. The matching constructs are shaded and the active keyword instances are underlined in the figure. In Figure 3.3 the inter-construct connection is circumscribed by a dashed line and it consists of the vertices Tag, $T1$, $R2$ and $M1$ where $R2$ is augmented with its class vertex Record.

In order to define result graph,s we need the concept of acyclic subgraph with respect to a query signature. Let G_s be a subgraph of the RDF graph that comprises all the constructs in the signature of a query. We construct an undirected graph G_c as follows: there is exactly one vertex in G_C for every matching construct and for every vertex not in a matching construct in G_s. Further: (a) if v_1 and v_2 are non-construct vertices in G_c, there is an edge between v_1 and v_2 in G_c iff there is an edge between the corresponding vertices in G_s, (b) if v_1 is a construct vertex and v_2 is a non-construct vertex in G_c, there is an edge between v_1 and v_2 in G_c iff there is an edge between a vertex of the construct corresponding to v_1 and the vertex corresponding to v_2 in G_s, and (c) if v_1 and v_2 are two construct vertices, there is an edge between them in G_c iff there exists in G_s, an edge between a vertex of the construct corresponding to v_1 and a vertex of the construct corresponding to v_2 that edge does not occur in any one of the constructs. Graph G_s is said to be *connection acyclic* if there is no cycle in G_c.

Consider the query Q = *{Cicada, musical, Playlist}* on the RDF graph G of Figure 3.1. Figure 3.4 shows two subgraphs of G which comprise a signature of Q on G. The active keyword instances are underlined and the corresponding matching constructs are shaded. One can see that the subgraph in Figure 3.4(a) is connection cyclic while the other subgraph Figure 3.4(b) is connection acyclic.

Definition 3.2.4 (Result Graph)**.** Given a signature S for a query Q over an RDF graph G, a

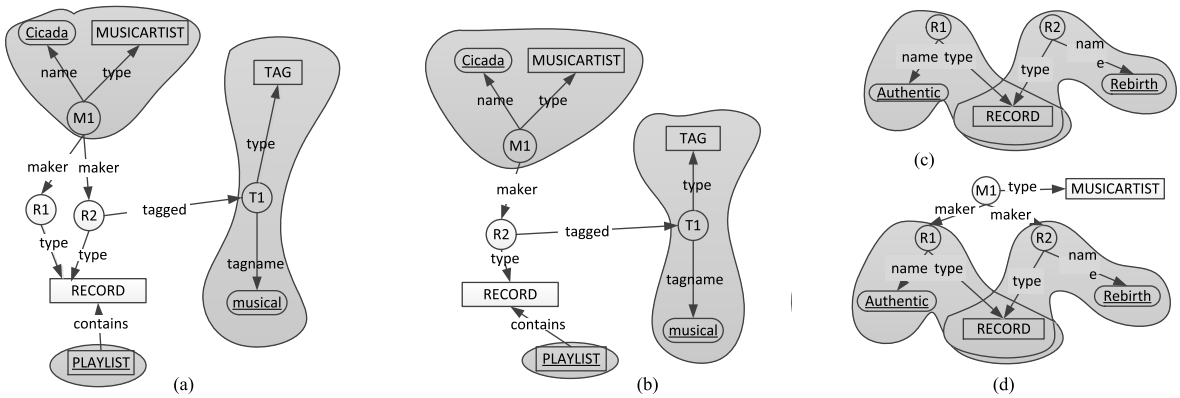


Figure 3.4 (a) Invalid result graph (b) Valid result graph (c) and (d) result graphs with overlapping matching constructs.

result graph of Q for S is a connected connection acyclic subgraph G_R of G which contains only the matching constructs in S and possibly inter-construct connections between them.

Therefore, a result graph of a query contains all the matching constructs of a signature of the query and guarantees that they are linked with inter-construct connections into a connected whole. Note that a result graph might not contain any inter-construct connection (this can happen if every matching construct in the query signature overlaps with some other matching construct). However, if inter-construct connections are used within the result graph, no redundant (cycle creating) inter-construct connections are introduced.

Consider the query Q = *{Authentic, Rebirth}* on the RDF graph G of Figure 3.1. Figure 3.4(c) shows a result graph for Q in G that is formed by overlapping matching constructs without any inter-construct connections. The result graph in Figure 3.4(d) has the same overlapping matching constructs but it also includes an inter-contruct connection between them. This is permissible since this subgraph is connection acyclic.

We now define the *answer* of a query Q.

Definition 3.2.5 (Query Answer)**.** The *answer* of a query Q on an RDF graph G is the set of result graphs of Q on G.

3.3 The Structural Summary: Summarized RDF Data

We formally introduce in this section the structural summary of a data graph. Intuitively, the structural summary of an RDF graph G is a special type of graph which summarizes the data graph showing vertices and edges corresponding to the class vertices and property, relationship and subclass edges in G. A structural summary graph is typically much smaller than its underlying RDF data graph.

Definition 3.3.1 (Structural Summary)**.** The *structural summary* of an RDF graph $G(V,E,L,l)$ is a vertex and edge labeled graph $G'(V',E',L',l')$ where

- $V' = V'_C \cup V'_v$ where:

  (a) V'_C is a set of class vertices which has a one to one mapping f onto V_C,

  (b) V'_v is a set of value vertices which contains a vertex for every distinct pair (c,l_p) such that there exists an entity of class c in the RDF graph G having a property labeled by l_p.

  A class vertex c in V_C is labeled by the label of the corresponding class vertex $f(c)$ in G.

- $E' = E'_p \cup E'_r \cup E'_s$ where:

  (a) E'_p is a set of edges from vertices in V'_c to vertices in V'_v such that there is an edge $(c,v) \in E'_p$ labeled by l_p iff there is an entity of class $f(c)$ in G which has a property edge labeled by l_p,

  (b) E'_r is a set of edges from a vertex in V'_C to another vertex in V'_C such that there is an edge $(c_1,c_2) \in E'_r$ labeled by l_r iff there is an edge from an entity of $f(c_1)$ to an entity of $f(c_2)$ in G labeled by l_r.

  (c) E'_s is a set of edges from a vertex in V'_C to another vertex in V'_C such that there is an edge $(c_1,c_2) \in E'_s$ labeled by *subclass* iff there is an edge from $f(c_1)$ to $f(c_2)$ in G labeled by *subclass*.

- L' is the set of labels of vertices in V' and edges in E'.

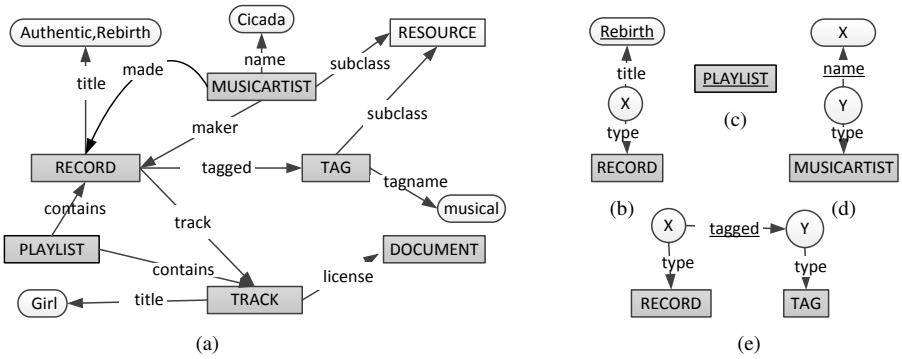


Figure 3.5 (a) Structural Summary, (b), (c), (d) and (e) are value, class, property, and relationship *MCs*, respectively.

- l' is a function assigning labels to vertices and edges in G' as already described above.

Figure 3.5(a) shows the structural summary for the RDF graph G of Figure 3.1. Similarly to matching constructs on the data graph we define matching constructs on the structural summary and we refer to them as *MC*. However, structural summaries do not have entity vertices. Therefore, an *MC* of a keyword on a structural summary possesses one distinct entity variable vertex, for every class vertex labeled by a distinct variable. We formally define *MC* as follows:

Definition 3.3.2 (Matching Construct on Structural Summary)**.** A *matching constructs on structural summary (MC)* for a keyword in a query Q is a graph similar to a matching construct on the data graph for the same keyword, but with the following two exceptions:

(a) the labels of the entity vertices in the property, relationship and value matching constructs over the data are replaced by distinct variables in the *MC*. These variables are called *entity variables* and they range over entity labels.

(b) The labels of the value vertices in the property matching constructs over the data are replaced by distinct variables. These variables are called *value variables* and they range over value labels in the RDF graph.

Figures 3.5(b), (c), (d), and (e) show the value, class, property, and relationship *MCs* for different keyword instances on the structural summary of Figure 3.5(a) for the

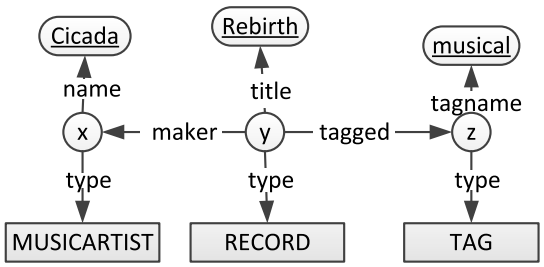


Figure 3.6 Query Pattern Graph.

RDF graph G of Figure 3.1. The underlined label in every *MC* determines the keyword instance for which the *MC* is defined.

3.4 Pattern Graphs: Structured Queries on Structural Summary

Pattern graphs represent different interpretations of the imprecise keyword query. They are, in fact, structured queries that can be evaluated against the RDF graph data to compute the keyword query answer. Pattern graphs are constructed over the structural summary. They comprise *MCs* for every keyword in the query and the connections between them without forming any cycle, on the structural summary.

Definition 3.4.1 (Pattern Graph)**.** A *(result) pattern graph* for a keyword query Q is a graph similar to a result graph for Q, and with the same keyword instances, but with the following two exceptions:

(a) the labels of the entity vertices in the result graph, if any, are replaced by distinct *entity variables* in the pattern graph.

(b) The labels of the value vertices are replaced by distinct *value variables* whenever these labels are not the keyword instances in the result graph.

Figure 3.6 shows an example of a pattern graph, for the keyword query $Q = \{Cicada, Rebirth, musical\}$. X, Y and Z are entity variables.

We will be using these definitions of the RDF data model, queries and answers, the structural summary and pattern graphs in all the following chapters of the dissertation.

CHAPTER 4

EXPLOITATION OF SEMANTIC RESULT CLUSTERING

This chapter introduces the semantic clustering hierarchical system that we have developed to address the typical problems of keyword search discussed in Chapter 1. This system is built on the RDF data model described in Section 3.1 and exploits the structural summary (Section 3.3) of the RDF data graph. Our hierarchical system comprises two levels. The matching constructs of the query keywords on the structural summary of the RDF data graph are the top level hierarchy components. The second level of the hierarchy consists of pattern graphs (Section 3.4). These pattern graphs are computed over the structural summary, strictly consisting of one matching construct for every keyword in the keyword query without forming any cycle. We present in this chapter an algorithm for computing pattern graphs on the structural summary, followed by a detailed description of our clustering hierarchical system. A user can navigate through our hierarchy system and provide feedback to enable the system to identify the most relevant pattern graph that represents his intent. On selection of the relevant pattern graph by the user, the system evaluates the pattern graph against the RDF data graph and returns result graphs to the user. Finally, we report the results of the experiments conducted to evaluate the effectiveness of our approach and efficiency of the algorithm in generating pattern graphs.

4.1 An Algorithm for Computing Query Pattern Graphs

Our algorithm computes r-radius Steiner pattern graphs. The r-radius Steiner graph computation is inspired by the algorithm of [63]. However, unlike that algorithm, our algorithm allows the keywords to match the edge labels of the graph.

Our algorithm is shown in Algorithm 1. It takes as input the structural summary G' of a data graph and a signature S of a query Q on G' (the set of *MCs* for the keywords of

Algorithm 1 : Query Pattern Graphs Computation

Input: S: signature, G': structural summary.
Output: $\mathscr{P}$: set of pattern graphs.

1: $\mathscr{K} \leftarrow$ extract distinct class vertices from S;
2: $\mathscr{A} \leftarrow$ adjacency matrix encoding class vertices V'_C and rel. edges E'_R in G';
3: $r \leftarrow 1$; ▷ radius of the Steiner graph
4: $\mathscr{P} \leftarrow \emptyset$;
5: $\mathscr{C} \leftarrow \emptyset$; ▷ set of connecting vertices
6: **while** $\mathscr{C} = \emptyset$ **do**
7: **for all** rows $i \in \mathscr{A}$ **do**
8: $conn \leftarrow \mathscr{A}(i,0)$; ▷ adding class id to set conn
9: **for all** cols $j \in \mathscr{A}$ **do**
10: **if** $\mathscr{A}(i,j) \neq \emptyset$ **then**
11: $conn = conn \cup (\mathscr{A}(0,j))$; ▷ adding class id to set conn
12: **if** $\mathscr{K} \subseteq conn$ **then**
13: $\mathscr{C} \leftarrow \mathscr{A}(i,0)$; ▷ a connecting node is found
14: $P \leftarrow$ GeneratePattern$(G', r, \mathscr{S}, \mathscr{A}(i,0))$;
15: **if** $P \notin \mathscr{P}$ **then**
16: $\mathscr{P} \leftarrow \mathscr{P} \cup P$;
17: $conn \leftarrow \emptyset$;
18: **if** $\mathscr{C} = \emptyset$ **then**
19: r=r+1;
20: $\mathscr{A}' \leftarrow$ modify adjacency matrix $\mathscr{A}$ to represent how the class vertices are connected to each other at a distance less than or equal to r;
21: $\mathscr{A} \leftarrow \mathscr{A}'$

Q on G'). It produces as output r-radius Steiner query pattern graphs on G' that contain S, and whose radius r is minimum.

Every MC in S is associated with one or two class vertices. Set $\mathscr{K}$ is initialized with all the class vertices in S. The adjacency matrix A is a square matrix of order $n+1$, where n is the number of class vertices in G'. The first row and column of A record the class identifiers. $\mathscr{A}(i,j)$ holds information about the relationship edges connecting the classes $\mathscr{A}(i,0)$ and $\mathscr{A}(0,j)$. Initially the algorithm tries to find Steiner pattern graphs with radius $r = 1$. If no graph is found, r is gradually increased. Sets $\mathscr{P}$ and $\mathscr{C}$ represent the data structures recording the pattern graphs and their corresponding connecting vertices, respectively. Procedure *GeneratePattern* generates the pattern graphs using the information recorded about the relationship edges between the connecting vertex and the class vertices in $\mathscr{K}$ and the MCs in S.

4.2 Semantic Hierarchical Clustering and Ranking

We now describe our result clustering hierarchy and how the user navigates through the hierarchy and pattern graph ranking.

Semantic Hierarchical Clustering. The hierarchy has two levels on top of the result graph layer. The pattern graphs of a query Q on an RDF graph G define a partition of the result graphs of Q on G. The pattern graphs constitute the *first level* of the clustering hierarchy. Multiple pattern graphs can share the same signature. The signatures determine a partition of the pattern graphs of Q on G. They, in turn, define a partition of the results which is coarser than that of the pattern graphs. The signatures constitute the *second (top) level* of the hierarchy.

Hierarchy Navigation. In order to navigate through the hierarchy after issuing a query the user starts from the top level. The top level may have numerous signatures. However, the user does not have to examine all the signatures. Instead, she is presented with the MC list for one of the query keywords. We describe below how this list of MCs is ranked. As mentioned earlier, the MCs of a keyword provide all the possible interpretations for this keyword in the data. The user selects the MC that she considers relevant to her intent. Subsequently, she is presented with the MC lists of the other query keywords, though some of the MC lists can be skipped. This can happen if the user selects an MC which involves more than one keyword instances that she wants to see combined together in one MC. Once MCs for all keywords have been selected, that is, a query signature has been determined, the system presents a ranked list of all the pattern graphs that comply with the signature. The user chooses the pattern graph of her preference which is evaluated by the system. The result graphs are returned to the user.

Ranking. The MCs for a keyword are ranked in an MC list based on the following rules: (a) MCs that involve more than one active keyword instances are ranked first in order of the number of active keyword instances they contain, (b) class MCs, relationship MCs and

property MCs are ranked next in that order, (c) value MCs follow next and are ranked in descending order of the frequency of their value. The *value frequency* f_m^v of a value MC m with property p, class c and value (keyword) v is the number $n_{p,c}^v$ of occurrences of the value v in matching constructs involving p and c in the data graph divided by the number $n_{p,c}$ of occurrences of property matching constructs in the data graph involving p and c. That is,

$$f_m^v = n_{p,c}^v / n_{p,c}$$

This ranking of the MCs favors MCs with multiple keyword instances based on the assumption that keywords that occur in close proximity are more relevant to the user's intent. Further, it favors MCs whose active keyword matches a schema element (class, relationship or property), favoring most class MCs which have unique occurrences in the data graph. Finally, value MCs are ranked at the end since they are more specific. The value frequency of a value MC reflects the popularity of this MC in the data. Therefore, value MCs with high value frequency are ranked higher than value MCs for the same value with low value frequency.

The pattern graphs the system ranks share the same signature. Thus, they are r-radius graphs with the same r. In almost all the cases they have the same number of edges and they differ only in the relationship edges which are not part of any MC. For this reason, the pattern graphs are ranked in descending order of their connecting edge frequency defined next. Given a pattern P, its *connecting edge frequency*, $f_c(P)$, is the sum of the number n_e occurrences in the data graph of the relationship edges e in P that do not occur in an MC in P divided by the total number $|E_R|$ of relationship edges in the data graph. That is, if E_c is the set of these relationship edges in P,

$$f_c(P) = \sum_{e \in E_c} n_e / |E_R|$$

In order to rank MCs and pattern graphs, our system needs statistics about value

MCs and their property edges and about connecting relationship edges in pattern graphs. This information is precomputed and stored with the structural summary when this one is constructed. Therefore, no access to the data graph is needed.

4.3 Experimental Evaluation

We implemented our approach and ran experiments to evaluate our system. The goal of our experiment is to assess: (a) the effectiveness of our clustering approach, (b) the efficiency of our techniques in providing suggestions to the user and in obtaining results from the selected pattern graphs in real time. It is not meaningful to run experiments to measure precision and recall since our approach exploits relevance feedback and returns all and only the results which are relevant to the user intent (perfect precision and recall).

4.3.1 Dataset and Queries

We use Jamendo, a large repository of Creative Commons licensed music. Jamendo is a dataset of 1.1M triples and of 85MB size containing information about musicians, music tracks, records, licenses of the tracks, music categories, track lyrics and many other details related to them. Its structural summary was extracted and stored in a relational database. Experiments are conducted on a standalone machine with an Intel i5-3210M @ 2.5GHz processors and 8GB memory.

Users provided different queries on the Jamendo dataset and navigated through our hierarchical clustering system to select a relevant MC (for every keyword) and a relevant pattern graph (when more than one were proposed by the system for the selected MCs). We report on 10 of them. The queries cover a broad range of cases. They involve from 3 to 7 keywords. Table 4.1 shows the keyword queries and statistics about them. For every query it shows the total number of keyword instances in the data graph (#I), the number of MCs (for all the keywords) in the structural summary (#MC), and the total number of signatures

Table 4.1 Queries used in the Experiments and their Statistics

Query	keywords	#I	#MC	#S
1	Track Obsession Divergence format title mp32	699	29	2,646
2	biography guitarist track lemonade	633	18	216
3	Knees Cicada recorded_as	59	7	9
4	sweet recorded_as Signal onTimeLine 104734	177	17	48
5	Track Nuts chillout ACExpress	618	21	252
6	Mako La deux date love time	2846	36	24,300
7	Fantasie recorded_as factor published_as format date title	145	25	588
8	Fantasie recorded_as Performance Paure	68	8	8
9	Briareus Vampires Infirmary Cool	154	18	288
10	Fantasie text Paure Document	144	13	42

(#S). As we can see, a query can have many pattern graphs (their number is greater than or equal to that of the signatures). However, thanks to our hierarchical clustering approach, the user has to examine only one or at most two of them. Further, exploiting our ranking of the MCs, the user has to examine only a fraction of the MCs for every query.

4.3.2 Effectiveness of Hierarchical Clustering

In order to measure the retrieval effectiveness of our hierarchical clustering, we adapted the *reach time* metric used in [55, 67]. The reach time sets forth to quantify the time spent by a user to locate the relevant results. In our case, the relevant results are represented by the relevant pattern graph. The relevant results in terms of subgraphs of the data graph can then be retrieved by evaluating the pattern graph against the database. For simplicity, we assume that the user always selects one relevant pattern graph. We employ different versions of reach time in two different settings: when the components (MCs and pattern graphs) are ranked, and when they are not. rt_{avg} and rt_{max} apply to the case the components are not ranked. rt_{avg} (resp. rt_{max}) denotes the average (resp. maximum) number of components the user needs to examine in order to retrieve the relevant pattern. rt_{rank} denotes the number of components the user examines in order to retrieve the relevant pattern when the components are ranked. For instance, if a query has k keywords and the user needs to examine m_i of the

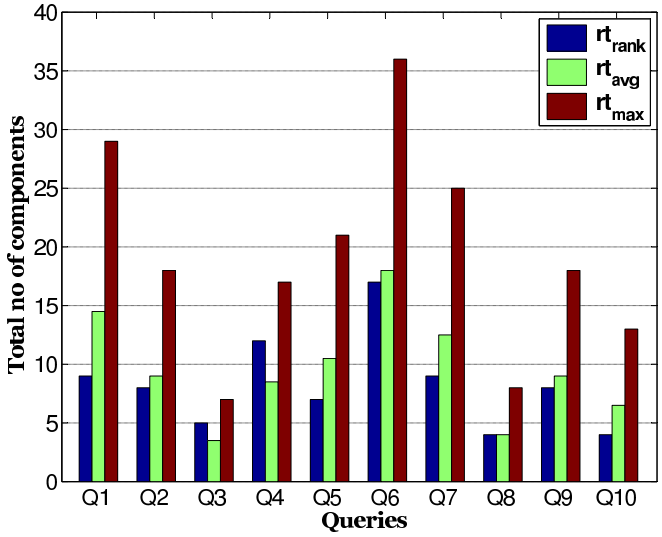


Figure 4.1 Reach times of the two clustering approaches.

ranked MCs for keyword i and p of the ranked pattern graphs for the selected MCs,

$$rt_{rank} = \sum_{i=1}^{k} m_i + p$$

Figure 4.1 shows the reach times for the queries of Table 4.1.

As one can see, the user has to examine on the average a small number of components even for queries with many keywords. Further, when the components are ranked, rt_{rank} is smaller than rt_{avg} in most cases and never comparable to rt_{max}. This demonstrates the feasibility of our hierarchical clustering system and the quality of the component ranking process.

4.3.3 Efficiency of the System

In order to asses the efficiency of our system, we measured the time needed to compute the ranked list of MCs for the query keywords and the time needed to evaluate the selected pattern graphs.

Figure 4.2(a) shows the total time (*totalMC*) needed to compute and rank the MCs

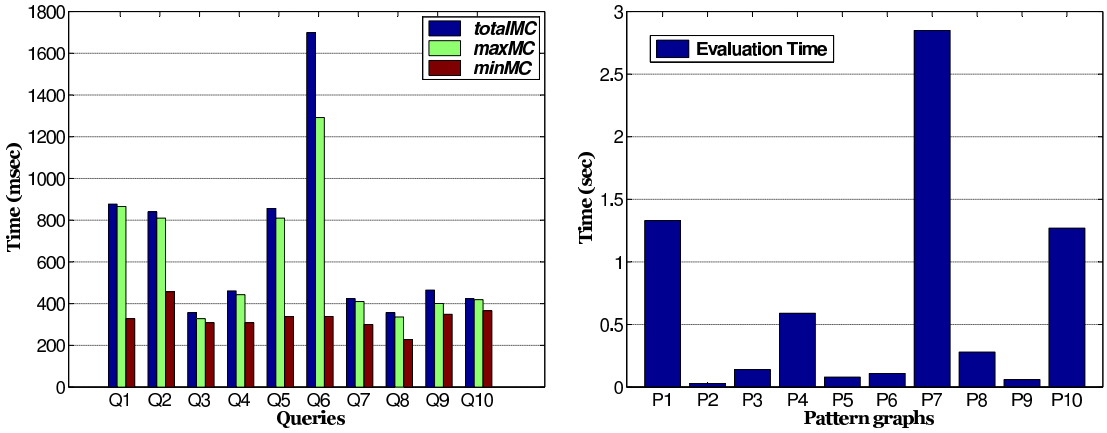


Figure 4.2 (a) Time to compute the MCs for the query keywords (b) Evaluation time for the selected pattern graphs.

for the keyword queries. It also shows the shortest(*minMC*) and the longest (*maxMC*) time needed to compute the rank list of MCs of a keyword in a given query. The list of MCs for the first keyword in the query is presented to the user for selection as soon as it is computed. The plot displays interactive time for the all the queries which do not delay the selection process. The time needed to compute the pattern graphs on the structural summary after the MCs for the query keywords are selected by the user are insignificant and are not displayed here. This is expected since the pattern graphs are computed using exclusively the structural summary whose size is very small compared to the size of the data.

Figure 4.2(b) displays the time needed to evaluate the selected pattern graphs on the data graph. This diagram again shows interactive times even though it is a prototype system and no optimizations have been applied.

4.4 Conclusion

We have presented a novel approach to address the problems related to keyword search on large RDF data. Our approach hierarchically clusters the result graphs and leverages relevance feedback from the user. In order to form the clustering hierarchy, we use matching constructs and pattern graphs which are subgraphs representing semantic interpretations

for keywords and queries, respectively. We presented an algorithm to efficiently compute r-radius Steiner pattern graphs. All hierarchy components are computed efficiently on the structural summary without accessing the (much larger) data graph. We presented a technique that ranks the hierarchy components based on their structural and semantic features and occurrence frequencies. Our approach allows the user to explore only a tiny portion of the clustering hierarchy in selecting the relevant graph patterns supported by the component ranking. The experimental evaluation of our approach shows its feasibility by demonstrating short reach times to the relevant pattern graphs and efficient computation of the relevant result graphs on the data graph.

CHAPTER 5

KEYWORD PATTERN GRAPH RELAXATION

In Chapter 1, we have discussed the benefits of structural summary-based approaches for keyword search on RDF data and also identified their drawbacks. In this chapter, we address the shortcomings of structural summary-based approaches while still leveraging the structural summary for keyword query processing on RDF data. We present a novel approach which combines the use of the structural summary and the user feedback with a relaxation technique for pattern graphs. We leverage pattern graph homomorphisms to define relaxed pattern graphs that are able to extract more results potentially of interest to the user. We introduce an operation on pattern graphs and we prove that it is complete, that is, it can produce all relaxed pattern graphs. To guarantee that the relaxed pattern graphs are as close to the initial pattern graph as possible, we devise different metrics to measure the degree of relaxation of a pattern graph. We design an algorithm that computes relaxed pattern graphs with non-empty answers in relaxation order. To improve the successive computation of relaxed pattern graphs, we suggest subquery caching and multiquery optimization techniques adapted to the context of this computation. Finally, we run experiments on different real datasets which demonstrate the effectiveness of our ranking of relaxed pattern graphs, and the efficiency of our system and optimization techniques in computing relaxed pattern graphs and their answers.

5.1 Computing and Selecting Pattern Graphs

We use the same definition of the data model, and the notions of structural summary, matching constructs on structural summary, inter-construct connections, signature and pattern graphs as defined in Chapter 3. Figure 5.1(a) shows an example of the RDF data graph over which all the other examples of this chapter are drawn. Figure 5.1 also have exam-

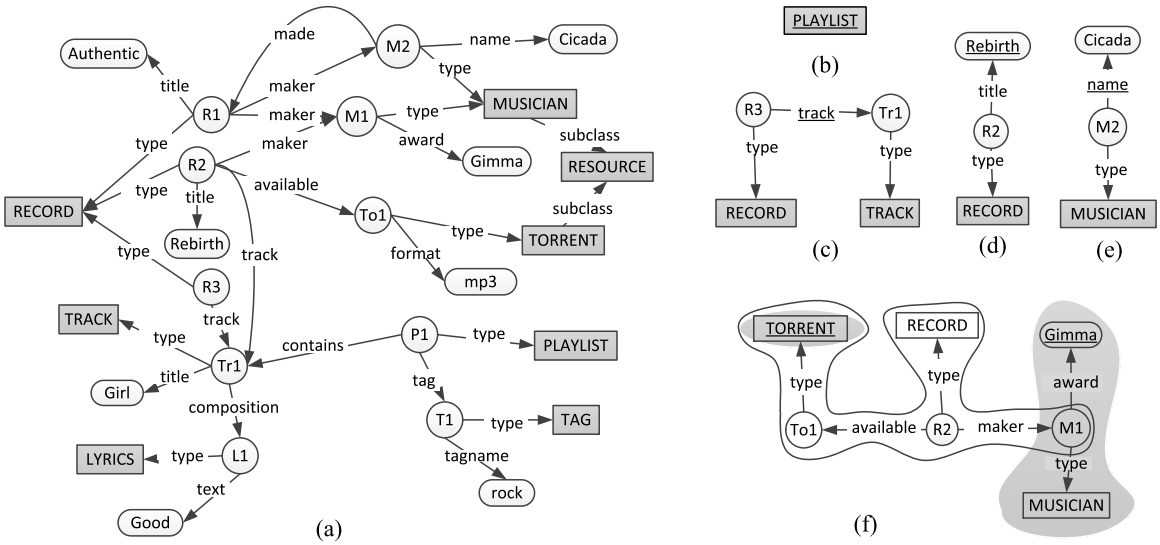


Figure 5.1 (a) An RDF graph, (b), (c), (d) and (e) class, relationship, value and property matching constructs, respectively, (f) inter-construct connection and result graph.

ples of different matching constructs (Figures 5.1(b), (c), (d) and (e)) and inter-construct connection (Figure 5.1(f)) over the RDF data graph in Figure 5.1(a).

For computing the pattern graphs of a query on the structural summary, we can use the algorithm, described in Section 4.1, which computes *r*-radius Steiner graphs [22]. The user selects a pattern graph by navigation through a two-level semantic hierarchical clustering system [22]. Nevertheless, the way the pattern graph is selected by the user is orthogonal to the relaxation method we present in this chapter. Any other approach like those in [32, 89, 90, 92] can be used for selecting the relevant pattern graph which will be relaxed.

5.2 Computing Relaxed Pattern Graphs

A pattern graph selected by the user might return no results, or if it does, it might miss some interesting results the user would like to see. In order to expand the result set of this pattern graph chosen by the user and get additional results for the same query signature that involve the same classes, relationships, properties and values but additional entities, we relax this pattern graph. In this section, we first define relaxed pattern graphs. We then

introduce an operation on pattern graphs, called vertex split operation, and we show that a pattern graph can be relaxed by applying vertex split operations. Pattern graphs which are less relaxed are preferable over pattern graphs which are more relaxed since, they are closer to the original pattern graph selected by the user. Therefore, we introduce different metrics to characterize the degree of relaxation of a relaxed pattern graph.

5.2.1 Relaxed Pattern Graphs

In order to define relaxed pattern graphs, we need the concept of homomorphism between pattern graphs.

Definition 5.2.1 (Pattern Graph Homomorphism)**.** Let P_1 and P_2 be two pattern graphs. A *homomorphism* from P_1 to P_2 is a function h from the variable vertices (entity variable and value variable vertices) of P_1 to the variable vertices of P_2 such that, if X is an entity variable vertex in P_1:

(a) for any type edge (X, c) in P_1, there is a type edge $(h(X), c)$ in P_2. That is, X in P_1 and $h(X)$ in P_2 are of the same type c.

(b) for every relationship edge (X, Y) in P_1 labeled by r, where Y is another entity variable in P_1, there is a relationship edge $(h(X), h(Y))$ in P_2 labeled by the same label r.

(c) for every property edge (X, Y) in P_1 labeled by p, where Y is a value variable vertex, there is a property edge $(h(X), h(Y))$ in P_2 labeled by the same label p.

(d) for every property edge (X, v) in P_1 labeled by p, where v is a value vertex labeled by the value (keyword) V, there is a property edge $(h(X), v')$ in P_2 labeled by the same label p, where v' is a value vertex also labeled by V.

Figure 5.2 shows four pattern graphs P_1, P_2, P_3 and P_4 and a homomorphism from the pattern graph P_2 to the pattern graph P_1. The vertex mapping is illustrated with dashed

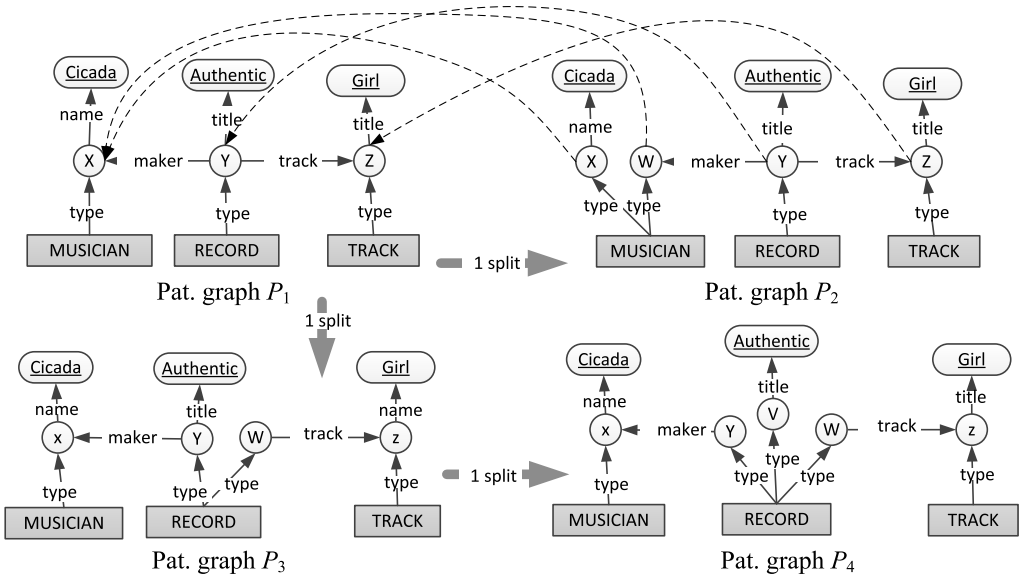


Figure 5.2 An original pattern graph P_1 and relaxed pattern graphs P_2, P_3, P_4.

arrows. One can see that there are also homomorphisms from the pattern graphs P_3 and P_4 to the graph pattern P_1. However, there is no homomorphism from pattern graph P_1 to any one of the other pattern graphs.

We use the concept of homomorphism to define a relation on pattern graphs.

Definition 5.2.2 (Relation $\prec$)**.** Let P_1 and P_2 be two pattern graphs. We say that P_2 is a *relaxation* of P_1 or that P_2 is a *relaxed version* of P_1 if there is a homomorphism from P_2 to P_1 but there is no homomorphism from P_1 to P_2. In this case, we write $P_1 \prec P_2$.

In the example of Figure 5.2, $P_1 \prec P_2$ and $P_1 \prec P_3 \prec P_4$. No other $\prec$ relationship holds between these pattern graphs.

Clearly, relation $\prec$ is a strict partial order on the set of pattern graphs (it is irreflexive, asymmetric and transitive). We call its minimal elements *original* pattern graphs. In an original pattern graph, every class vertex is connected through a type edge exactly to one entity variable vertex. The patterns initially presented to the user are original pattern graphs and one of them is selected and possibly relaxed. If an (original) pattern graph P has an embedding to an RDF graph, a relaxed version of P also has an embedding to the same RDF graph. The opposite is not necessarily true. Therefore, with relaxed pattern graphs

we can expand the result set of an original pattern graph.

5.2.2 Vertex Splitting

A pattern graph is relaxed by applying the *vertex split* operation to one or more of its entity variable vertices. The split operation "splits" an entity variable vertex in a pattern graph into two entity variable vertices of the same type and partitions the incident edges of the original entity variable vertex between the two new vertices as indicated by the operation.

Definition 5.2.3 (Vertex split operation)**.** Let P be a pattern graph, v be an entity variable vertex in P connected with a type edge to a class vertex c, and $E = \{e_1, \ldots, e_k\}$, $k \geq 1$, be a proper subset of the set of non-type edges incident to v in P. Assume the edges $e_1, \ldots, e_k$, are connecting the pairs of vertices $(v, v_1), \ldots, (v, v_k)$, respectively. The *vertex split* operation $split(P, v, E)$ returns a pattern graph constructed from P as follows:

(a) Add to P a new entity variable vertex v' of type c.

(b) Remove all the non-type edges (incident to v) that occur in E.

(c) Add k edges $(v', v_1), \ldots, (v', v_k)$ having the same labels as the edges $e_1, \ldots, e_k$, respectively.

Splitting one or more of the vertices of an original pattern graph P results in a relaxed pattern graph (a relaxed version of P). Applying the split operation in sequence can create a pattern graph where the non-type edges incident to v are partitioned into more than two sets attached to different vertices, as desired.

Not all the entity variable vertices are interesting for splitting. This operation is defined only on candidate split vertices. An entity variable vertex is a *candidate split* vertex if it has at least two non-type edges.

As an example, consider the original pattern graph P_1 of Figure 5.2. This is a pattern graph for the keyword query `{Cicada, Authentic, Girl}`. Applying

$split(P_1, X, \{maker\})$ to P_1 results in the pattern graph P_2. Applying $split(P_1, Y, \{track\})$ to P_1 results in the pattern graph P_3. Applying, in turn, $split(P_3, Y, \{title\})$ to P_3 produces the pattern graph P_4.

Since any partitioning of the edges incident to a vertex in an original pattern graph can be obtained in a relaxed pattern graph by a successive application of vertex split operations, the following proposition can be shown.

Proposition 5.2.1. *Let P_1 and P_2 be two pattern graphs. Then, $P_1 \prec P_2$ iff P_2 can be produced from P_1 by applying a sequence of vertex split operations.*

Proof: *If part*: If P_2 can be produced from P_1 by applying a vertex split operation then, as stated in Definition 5.2.3, the non-type edges of an entity variable vertex X in P_1 are partitioned into two sets of edges which are incident to vertex X and its split image in P_2. Then clearly, there is a homomorphism from P_2 to P_1. If a sequence of vertex split operations is applied then this homomorphism exists by transitivity. That is, $P_1 \prec P_2$.

Only-if part: Since P_1 and P_2 are both pattern graphs, there is a homomorphism from P_2 to P_1 only if for any entity variable vertex X in P_1 there are $X_1, \ldots, X_k$, $k \geq 1$, vertices of the same type as X in P_2 and the non-type edges of X are partitioned among these vertices such that each one of $X_1, \ldots, X_k$ has at least one non-type edge (Definition 5.2.1). Then clearly, P_2 can be obtained from P_1 by applying in sequence $k-1$ vertex split operations for every vertex in P_1 where $k > 1$.

The above proposition shows that the vertex split operation is sound and complete with respect to relaxed pattern graphs.

5.2.3 Measuring Pattern Graph Relaxation

Usually, we want to relax a pattern graph so that it is as close to the initial pattern graph as possible. To this end, we introduce three metrics of decreasing importance to measure the degree of relaxation of a pattern graph. All these three metrics depend on the vertex split

operations applied to the original pattern graph. The first one is called connectivity of the pattern graph. In order to define the connectivity of a pattern graph, we use the concept of tightly connected pair of keyword instances. Two keyword instances in a pattern graph P are *tightly connected* if there exists a simple path between them which does not go through a class vertex. For instance, in the pattern graph of Figure 5.3(b), the keyword instances `Rebirth` and `mp3` are tightly connected whereas the keyword instances `Cicada` and `Rebirth` are not.

Definition 5.2.4 (Pattern graph connectivity)**.** The *connectivity* of a pattern graph is the number of unordered keyword instance pairs that are strongly connected divided by the total number of unordered keyword instance pairs.

In an original pattern graph, all pairs of keyword instances are strongly connected. Therefore, its connectivity is 1. Relaxing such a pattern graph by applying the vertex split operation to any entity variable vertex produces a pattern graph of lower or same connectivity. Relaxing an acyclic original pattern graph by applying vertex splitting to any entity variable vertex always reduces the connectivity of the original pattern graph. For instance, the connectivity of the pattern graph in Figure 5.3(a) is 1. The connectivity of the relaxed pattern graphs of Figures 5.3(b) and (d) is 0.4 and the connectivity of those of Figures 5.3(c), (e) and (f) is 0.3.

In order to distinguish between relaxed pattern graphs of the same pattern graph which have the same connectivity, we introduce another metric called "dispersion" of the keyword instances of a pattern graph. Roughly speaking, this metric is used to capture how much the keywords are dispersed as a result of vertex split operations in the pattern graph. To formally define the keyword instance dispersion metric, we introduce the concept of "split distance." The *split distance* of two keyword instances in a pattern graph P is the minimum number of class vertices in the simple paths between these two keyword instances in P excluding the terminal vertices. The term "split distance" is explained by the fact that a class vertex is introduced in a simple path between two keyword instances only as a result

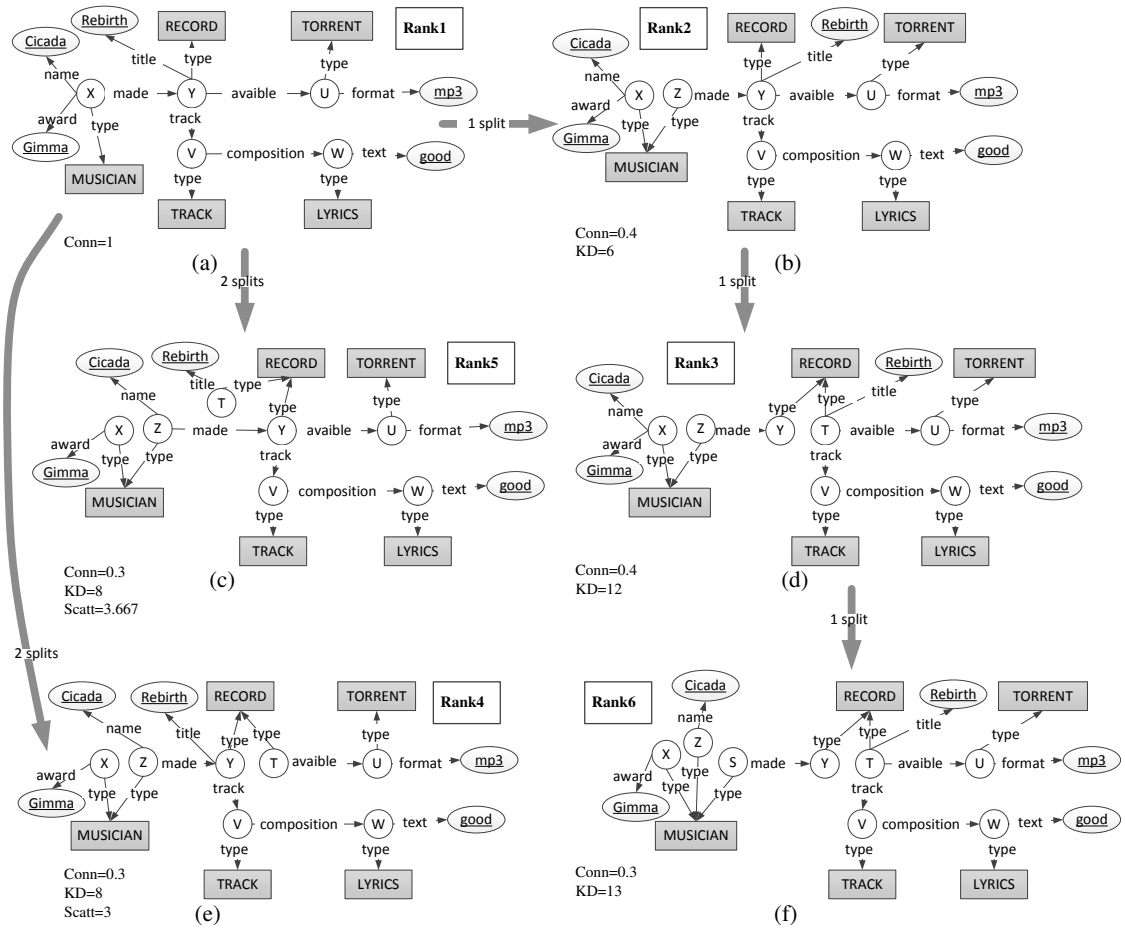


Figure 5.3 (a) Original pattern graph, (b), (c), (d), (e) and (f) relaxed pattern graphs.

of the application of a split operation. For instance, in the pattern graph of Figure 5.3(c), the split distance of the keyword instances of `Gimma` and `mp3` is 1 and that of `Gimma` and `Rebirth` is 2. The more split operations we apply to the vertices on a path between two keyword instances, the more syntactically dispersed these keyword instances become in the pattern graph, reflecting a weaker semantic connection between these keywords.

Definition 5.2.5 (Pattern graph keyword dispersion)**.** The *keyword dispersion* of a pattern graph P is the sum of the split distances of all unordered pairs of keyword instances in P.

A relaxed pattern graph with smaller keyword dispersion is preferred over a pattern graph of the same connectivity but higher keyword dispersion since its keywords are assumed to be more closely related. For example, the pattern graphs of Figures 5.3(b) and

(d) have the same connectivity of 0.4 whereas their keyword dispersion is 6 and 12, respectively. Hence, the pattern graph of Figure 5.3(b) will be ranked higher than that of Figure 5.3(d). Similarly, the connectivity of the pattern graphs of Figures 5.3(c), (e) and (f) is 0.3. However, the keyword dispersion of the pattern graphs of Figures 5.3(c) and (e) is 8 and that of Figure 5.3(f) is 13. Hence, the pattern graphs of Figures 5.3(c) and (e) will be ranked higher than that of Figure 5.3(f).

In order to differentiate between the degree of relaxation of pattern graphs having same connectivity and dispersion, we employ a third metric called *scatteredness* of a pattern graph. We first define the *distance* between two tightly connected keyword instances in a pattern graph as the number of vertices in a shortest path between them. The distance between a keyword instance which is the label of a value vertex and a keyword instance which is the label of a property edge incident to this vertex is 0. In the pattern graph of Figure 5.3(c), the distance between the tightly connected keyword instances of `Cicada` and `mp3` is 3 while the distance between the tightly connected keyword instances of `mp3` and `good` is 4.

A relaxed pattern graph partitions its keyword instances into sets of tightly connected keyword instances such that any two keyword instances which are tightly connected belong to the same set. The scatteredness of a pattern graph measures how sparsely are positioned the keyword instances within the sets of the partition.

Definition 5.2.6 (Scatteredness of a pattern graph)**.** Let N be the sum of the distances between all the unordered keyword instance pairs that are tightly connected, and S be the total number of tightly connected unordered keyword pairs in a pattern graph P. The *scatteredness* of the tightly connected keyword instances of P (scatteredness of P for short) is N/S.

In the example of Figure 5.3, the pattern graphs (c) and (e) have the same connectivity of 0.3 and the same keyword dispersion of 8. However, the scatteredness of the pattern graph of Figure 5.3(c) is 3.67 and that of the pattern graph of Figure 5.3(e) is 3. We use

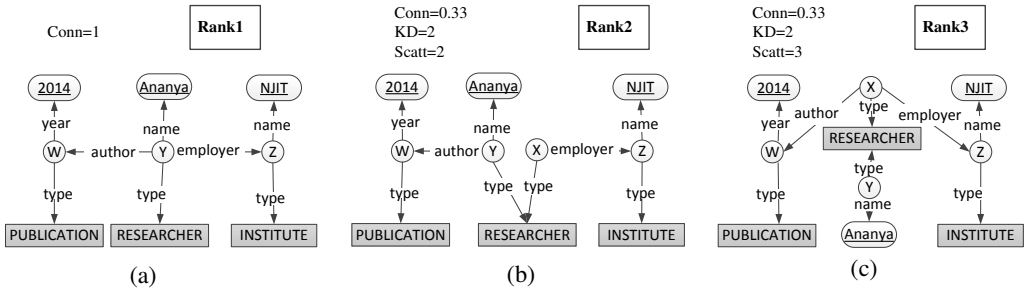


Figure 5.4 (a) Original pattern graph (b) and (c) relaxed pattern graphs.

the pattern graph scatteredness to rank the relaxed pattern graphs having the same connectivity and keyword dispersion. In our running example, the pattern graph of Figure 5.3(e) is ranked before the pattern graph of Figure 5.3(c), since the tightly connected keyword instances of the latter pattern graph are more sparsely positioned than that of the tightly connected keyword instances of the former pattern graph.

As another example, consider the pattern graph of Figure 5.4(a) and two relaxations of it shown in Figures 5.4(b) and (c). The relaxed pattern graphs of Figures 5.4(b) and (c) have the same connectivity and keyword dispersion but the scatteredness of the pattern graph Figure 5.4(b) is 2 and the pattern graph of Figure 5.4(c) is 3. Therefore, the pattern graph of Figure 5.4(b) should precede the pattern graph of Figure 5.4(c) in a ranking.

5.2.4 Relaxation Order

Given two pattern graphs P_1 and P_2, we say that, P_2 is "equally relaxed as" or "more relaxed than" P_1, and we write $P_1 \leq_r P_2$, if: (a) connectivity(P_1) $\geq$ connectivity(P_2), or (b) connectivity(P_1) $=$ connectivity(P_2) and dispersion(P_1) $\leq$ dispersion(P_2), or (c) connectivity(P_1) $=$ connectivity(P_2) and dispersion(P_1) $=$ dispersion(P_2) and scatteredness(P_1) $\leq$ scatteredness(P_2). Clearly, $\leq_r$ is reflexive and transitive and any two pattern graphs are comparable w.r.t. $\leq_r$. If a set of pattern graphs is ranked with respect to $\leq_r$, with the less relaxed pattern graphs ranked first, we say that it is ranked in *relaxation order*.

Since split operations introduce additional type edges in a pattern graph it is not difficult to see that the following statement holds.

Proposition 5.2.2. *Given two pattern graphs P_1 and P_2, if $P_1 \prec P_2$ then $P_1 \leq_r P_2$.*

Proof: By Proposition 5.2.1, if $P_1 \prec P_2$, there is a sequence of split operations which produce P_2 from P_1. Let now, P' be a pattern graph produced by applying split operation s to another pattern P. Since, s cannot not increase the number of tightly connected keyword instance pairs in P, connectivity(P') $\leq$ connectivity(P). Similarly, since it cannot reduce the split distance of any pair of keyword instance in P, dispersion(P) $\leq$ dispersion(P'). Finally, if s does not change the connectivity and keyword dispersion of P, P can only be a cyclic pattern graph, and P' is produced by applying a vertex split operation to an entity variable vertex which lies on a cycle in P. Hence, the sets of tightly connected keywords instances of P are not affected by s. Further, the distance between two tightly connected keyword instances within a set in P can only increase or remain the same when s is applied. Therefore, scatteredness(P) $\leq$ scatteredness(P'). Consequently, $P \leq_r P'$. Since this is true for all the split operations in the sequence that produced P_2 from P_1, and $\leq_r$ is transitive, $P_1 \leq_r P_2$.

Proposition 5.2.2 states that $P_1 \prec P_2$ is compatible with $P_1 \leq_r P_2$. If P_2 is produced by applying a vertex split operation to P_1, $P_1 \leq_r P_2$.

5.3 Computing Relaxed Pattern Graphs

In this section, we elaborate on the reasons for a pattern graph having an empty answer. Then, we design an algorithm which computes relaxed pattern graphs with non-empty answers ranked in ascending order of their degree of relaxation. Finally, we show how view materialization and multiquery optimization techniques can be exploited to support the computation of relaxed pattern graphs and their evaluation on the RDF graph.

5.3.1 Identifying Empty Vertices for Relaxation

If an original pattern graph for a query has an empty answer on an RDF graph, we would like to identify vertices in the pattern graph which if not split, the relaxed pattern graph will keep producing an empty answer. Splitting these vertices does not guarantee that the relaxed query does have a non-empty answer. However if we omit splitting any one of these vertices, the relaxed pattern graph will not return any results. We call these vertices *empty vertices*.

Definition 5.3.1 (Empty vertex)**.** An entity variable vertex X in a pattern graph P on a data graph G is an *empty vertex* iff P or any relaxed version of P where X is not split has an empty answer on G.

The following proposition characterizes empty vertices in a pattern graph. Let X be an entity variable vertex of type c in a pattern graph P, $p'_1(X,Z'_1),\ldots,p'_m(X,Z'_m)$ be the property edges incident to X whose value vertices $Z'_1,\ldots,Z'_m$ are variables, $p_1(X,v_1),\ldots,p_n(X,v_n)$ be the property edges incident to X whose value vertices $v_1,\ldots,v_n$ are not variables (they are keyword instances), $r_1(X,Y_1),\ldots,r_k(X,Y_k)$ be the relationship edges from X to some other entity variable vertices $Y_1,\ldots,Y_k$ of type $c_1\ldots c_k$, respectively, and $r'_1(X,Y'_1),\ldots,r'_l(X,Y'_l)$ be the relationship edges to X from some other entity variable vertices $Y'_1,\ldots,Y'_l$ of type $c'_1,\ldots,c'_l$, respectively (see Figure 5.5). We call the graph of Figure 5.5 the *star-join view* of the entity variable vertex X in P.

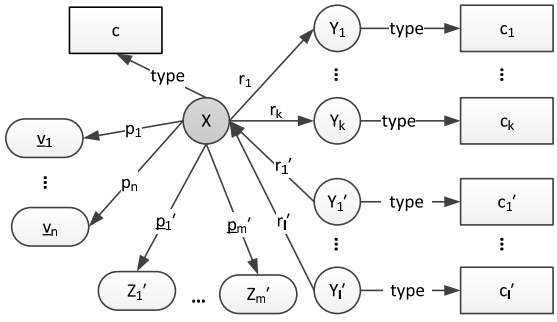


Figure 5.5 Star-join view of entity variable vertex X.

Proposition 5.3.1. *An entity variable vertex X is an empty vertex of a pattern graph P on an RDF graph G iff the star-join view for X in P has an empty answer on G.*

Proof: *If part*: If the star-join view V_X of X has an empty answer, then P or any relaxed version of P where X is not split has an empty answer since V_X is a subgraph of this graph. Therefore, X is an empty vertex.

Only-if part: Let us assume that X is empty and the star-join view of X is non-empty. We will show that this is a contradiction. Since X is empty, the pattern graph P or any relaxed version of it where X is not split do not have an answer on G. Let P' be a pattern graph obtained from P by splitting all entity variable vertices except X until no more split operations can be applied. Since the star-view join of X is non-empty, P' has an answer on G. This is a contradiction since we assumed that X is empty.

All empty vertices need to be split when relaxing a query in order to possibly get a nonempty answer for the query.

5.3.2 An Algorithm for Computing Relaxed Pattern Graphs

We provide now an algorithm which, given a pattern graph P chosen by the user (for instance, by navigating through the clustering hierarchy discussed in Chapter 4), gradually generates relaxed pattern graphs of P having non-empty answers. The algorithm returns these pattern graphs and their answers in ascending relaxation order. The number of relaxed pattern graphs returned is controlled by the user.

We provide now the intuition behind the algorithm. The chosen pattern graph might have an empty answer on the RDF data graph. For example, for the keyword query $Q =$ `{Gimma, Cicada, Rebirth, mp3, good}`, the user chosen pattern graph shown in Figure 5.3(a) does not have a match on the RDF data graph of Figure 5.1. Hence, it needs to be relaxed. One can see that this pattern graph has an empty entity variable vertex (vertex X) since the star join view of this vertex is empty (Proposition 5.3.1). All the empty

Algorithm 2 : Pattern Graph Relaxation Algorithm

Input: P: An original pattern graph.

Output: A list of relaxed pattern graphs of P with non-empty answers in ascending relaxation order. Every pattern graph is returned along with its answer.

```
1: R = {P};
2: MoreResults = True;
3: Ans = ∅;
4: while R ≠ ∅ and MoreResults do
5:     P_Top ← the pattern graph in R with the highest rank;
6:     R ← R − {P_Top};
7:     EV ← ComputeEmptyVertices(P_Top);
8:     Mark the new non-empty vertices in P_Top;
9:     if EV ≠ ∅ then
10:        NewR ← GetRelaxedFromEmptyVertices(P_Top, EV);
11:        Rank the pattern graphs in NewR in ascending relaxation order;
12:        R ← merge R and NewR into one list of patterns ranked in ascending relaxation
               order;
13:    else
14:        Ans ← Evaluate(P_Top);
15:        if Ans ≠ ∅ then
16:            Output (P_Top, Ans);
17:            MoreResults ← input from the user on whether more results are needed;
18:        if Ans = ∅ or MoreResults then
19:            MoreR ← GetRelaxed(P_Top);
20:            Rank the pattern graphs in R in ascending relaxation order;
21:            R ← merge R and MoreR into one list of patterns ranked in ascending
                   relaxation order;

22: function GetRelaxed(P)
23:    R = ∅
24:    for every candidate split vertex X in P do
25:        R_X = {pattern graphs obtained by applying one vertex split operation to X in all
                   possible ways}
26:        R = R ∪ R_X
27:    return R

28: function GetRelaxedFromEmptyVertices(P, EV)
29:    R = {P}
30:    for every vertex X in EV do
31:        R_X = ∅
32:        for every pattern graph P' in R do
33:            R_X = R_X ∪ {pattern graphs obtained by applying one vertex split operation
                       to X in P' in all possible ways}
34:            R = R − P'
35:        R = R_X
36:    return R
```

vertices of a pattern graph need to be split in order for the resulting pattern graph to have a non-empty answer (Definition 5.3.1). Therefore, vertex X needs to be split. Nevertheless, splitting all the empty vertices of a pattern graph does not guarantee that the resulting pattern graph will not have empty vertices. For instance, the pattern graphs 5.3(b) and (d) which are obtained from the pattern graph 5.3(a) by splitting its only empty vertex X, still have an empty vertex. Further, even if a relaxed pattern graph does not have empty vertices, it might still have an empty answer. In our running example of Figure 5.3 the relaxed pattern graphs of Figures 5.3(c) and (e) do not have empty vertices but have an empty answer on the RDF graph of Figure 5.1. In both the above cases, additional split operations need to be applied to candidate split vertices in order to reach a pattern graph with non-empty answer. On the other hand, splitting the empty vertices of a graph might result on a pattern graph which has a non-empty answer and this is the case of the pattern graph of Figure 5.3(f) which is obtained by applying a split operation to the only empty vertex X of the pattern graph of Figure 5.3(d). Since we want to return to the user relaxed pattern graphs with higher rank (w.r.t. $\leq_r$) first, we chose for relaxation a pattern graph with the highest rank at every iteration of the algorithm. For the same reason, if the pattern graph chosen for relaxation does not have empty nodes, we apply a split operation (in all possible ways) to all candidate split vertices separately. By choosing a pattern graph with the highest rank for relaxation at every iteration of the algorithm, we also avoid the redundant generation of relaxed pattern graphs.

Algorithm Description

The outline of our algorithm is shown in Algorithm 2. The input of this algorithm is an original pattern graph P. The algorithm generates as output a list of relaxed pattern graphs and their corresponding result graphs on the data graph in increasing order of relaxation. The data structure R is a list used to store pattern graphs (both original and relaxed). The variable $MoreResults$ reflects the user's choice of fetching more answers by further relaxing

the pattern graphs in R. The algorithm first chooses a pattern graph P_{Top} with the highest rank from R (line 5). The pattern graph P_{Top} is then checked for empty vertices (line 7). If P_{Top} has non-empty vertices, they are marked (line 8) and they (and their split images) remain marked in the relaxations of P_{Top}. If EV, the set of empty vertices in P_{Top} is non-empty, the function $GetRelaxedFromEmptyVertices(P_{Top}, EV)$ is called (line 10). This function relaxes P_{Top} by applying one vertex split operation to all of its empty vertices in all possible ways (lines 30-35). The resulting relaxed pattern graphs form a new list *NewR* of relaxed pattern graphs, which is then ranked in ascending relaxation order and is merged with the list R (lines 11-12). Otherwise, if P_{Top} does not have empty vertices, it is evaluated over the data graph and if the set *Ans* of result graphs is non-empty, P_{Top} is returned to the user along with *Ans* (lines 14-16). In case the user wants more results, or the pattern graph P_{Top} produces an empty answer when evaluated over the data graph, the function $GetRelaxed(P_{Top})$ is evoked (lines 18-19). This function relaxes P_{Top} by applying one vertex split operation to all of its candidate split vertices in all possible ways (lines 24-26). The list of relaxed pattern graphs returned by $GetRelaxed(P_{Top})$ is stored in a list *MoreR*. The relaxed pattern graphs in *MoreR* are then ranked and merged with the list R of pattern graphs (lines 20-21). The whole process, as described in lines 5-21, continues until the user is satisfied with the results or no more pattern graphs are left in R. The above discussion suggests the next proposition.

Proposition 5.3.2. *Algorithm 2 correctly computes in relaxation order the relaxed pattern graphs with non-empty answers for a given input pattern graph.*

Note that during the execution of the algorithm, the user can provide input on how to split empty or non-empty vertices when a pattern graph comes up for relaxation either because it has empty vertices or because it does not have empty vertices but has an empty answer. In this case, the number of split operations applied in this iteration of the algorithm is reduced since only the alternative dictated by the user is applied to the relevant vertex. We

have omitted this feature in the outline of the algorithm, showing only the fully automated version, for simplicity of presentation.

The execution cost of our pattern graph relaxation algorithm depends on: (a) the cost for determining the empty vertices (by evaluating star-join views over the data graph), (b) the evaluation cost of relaxed pattern graphs over the data graph, and (c) the cost for generating relaxed pattern graphs using the functions $GetRelaxed(P)$ and $GetRelaxedFromEmptyVertices(P,EV)$. The star-join views can be computed efficiently by exploiting the indexes defined on entity attributes of the relations for properties and relationships. For the efficient evaluation of the relaxed pattern graphs we devise and discuss in the next section evaluation plans for answering queries using materialized views and multiquery optimization techniques. Functions $GetRelaxed(P)$ and $GetRelaxedFromEmptyVertices(P,EV)$ can produce up to $(C_1^n + \ldots + C_{n-1}^n)/2 = 2^{n-1} - 1$ relaxed pattern graphs by applying vertex split operations on one vertex X, where n is the number of keywords. The worst case scenario can happen when each one of the n keyword instances in the pattern graph is linked to X through a different non-overlapping path. Since every pattern graph is a Steiner graph, it can have up to $nr+1$ entity variable vertices, where r is the radius of the Steiner graph. In the worst case, all of them are needed to be split. Nevertheless, even though in the worst case scenario an exponentially large number of relaxed pattern graphs can be produced, in practice only few of them are produced. Further, only a tiny portion of those produced are evaluated for empty vertices and empty results since otherwise the produced relaxed pattern graphs would be very irrelevant to the original pattern graph and not of interest to the user. This intuition is also confirmed in our experimental results.

5.3.3 Optimization Techniques to Support Query Relaxation and Evaluation

Materializing views in the main or secondary memory is a well-known technique for improving the performance of queries. This technique has been studied extensively over the years for queries on relational databases [85, 81], but the contributions for queries over

RDF databases are limited [58, 59]. Queries to be evaluated are rewritten (inclusively or exclusively) using the stored views [62] in order to produce a query evaluation plan involving the materialized views which is more efficient than a plan involving exclusively the base relations. The technique is useful both in a horizontal and in a vertical setting. In a vertical setting (query caching) queries and subqueries are cached on the assumption that they will be useful for evaluating subsequent queries. A new query to be evaluated is rewritten equivalently using previously cached views. The expectation is that the produced evaluation plan will be cheaper and the savings will amortize the cost for deciding what subqueries to cash and for finding a rewriting of the query using the materialized views. In a horizontal setting (multiquery optimization) multiple queries need to be evaluated concurrently. To this end, common subexpressions among the queries in a given workload are detected on the fly and a global evaluation plan for all the queries is derived, which might be more efficient to evaluate than evaluating each query in the workload separately. A global evaluation plan reflects rewritings of the given queries over the views (common subexpressions) which remain materialized until all the queries in the workload that use them are evaluated. The expectation is that the savings produced from the global evaluation plan will amortize the cost for detecting the common subexpressions and producing the alternative global evaluation plans. The multiquery optimization problem is a complex one. Not surprisingly, it has been shown to be NP-hard even for conjunctive relational queries [84]. There are different sources of complexity to these problems in the general relational context: (a) deciding whether a query Q can be answered using a set of materialized views and producing an equivalent rewriting of Q using the views, (b) detecting common subexpressions among queries in a query set, and (c) deciding which views/common subexpressions to materialize in order to produce an efficient evaluation plan.

Our pattern graph relaxation algorithm generates multiple queries to be evaluated sequentially or concurrently. We discuss in this section how the techniques discussed above can be leveraged to design a global evaluation plan for all the queries that need to be com-

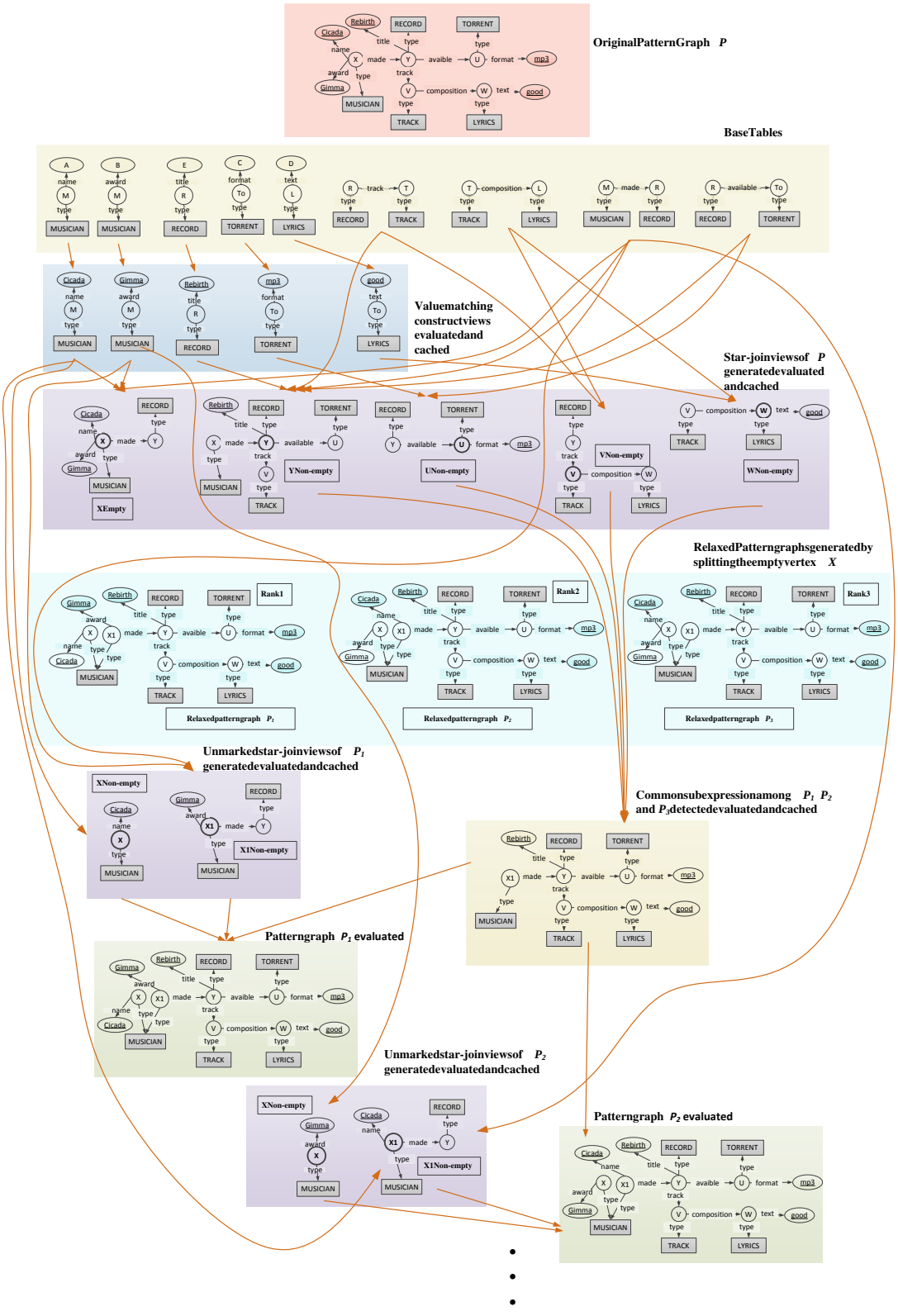


Figure 5.6 A global evaluation plan for relaxed queries.

puted. The goal is to exploit extensively common subexpressions among the generated queries. We consider a relational setting where the base relations are property and relationship relations. Initially, the value matching construct views of a given original pattern graph are evaluated and cached. Subsequently the star-join views are evaluated using the value matching constructs cached. Some of the generated queries are to be evaluated sequentially (when the pattern graphs with the highest rank is chosen for evaluation) while others are to be evaluated concurrently (when value matching construct views or star-join views are evaluated). Fortunately, these queries are not random queries but subgraphs of the original pattern graph or of its relaxations. As a consequence, common subexpressions among different queries can be detected easily based on the overlapping of the corresponding graphs because they are subgraphs of these graphs. Query rewritings can also be produced easily by simply joining the materialized subqueries (graphs) on their common entity variable vertices. Finally, common subexpressions among queries are selected so as to maximize the number of common entity variable vertices.

Figure 5.6 shows an example of caching and utilization of different subqueries for the successive evaluation of relaxed pattern graphs. The flow, from top to bottom, follows the execution of our algorithm. On the top of figure 5.6, the input original query is shown. Next follow the based tables needed to compute relaxations of this query. The global evaluation plan involves computing and caching the value matching constructs of the original query and its star-join views which are shown in the next two layers. The fifth layer displays a ranked list of relaxed pattern graphs produced by splitting the empty vertices of the original pattern graph. The first relaxed pattern graph is considered and checked for empty vertices by evaluating the star-join views of the unmarked entity variable vertices which are also cached. As no empty vertex is found, this relaxed pattern graph is evaluated. Its evaluation involves the computation of the maximal common subexpression of the relaxed pattern graphs and the cached star-join views of the entity variable vertices. The process continues with the next relaxed pattern graph.

5.4 Experimental Evaluation

We implemented our approach and run experiments to evaluate our system. The goal of our experiments is to assess: (a) the effectiveness of the metrics introduced in ranking the relaxed pattern graphs, and (b) the feasibility of our system in producing and presenting to the user the relaxed pattern graphs and their answers in real time.

5.4.1 Datasets and Queries

We used two real datasets Jamendo[1] and YAGO 1.0.0[2]. Jamendo is a large repository of Creative Commons licensed music. It consists of 1.1M triples and its size is 85MB. It contains information about musicians, records, music tracks and their licenses, music categories, track lyrics and many other details related to music production. This dataset has nearly 300,000 entities belonging to 12 classes. Jamendo has 14 properties and 10 relationships. Much larger, YAGO is an open domain dataset combining information about resources from different aspects of life extracted from Wordnet[3] and Wikipedia[4]. YAGO contains nearly 20 million triples about approximately 2 million entities belonging to over 180,000 classes. The entities in the YAGO dataset are characterized by 32 properties. The entities are associated to each other with 58 relationships.

The structural summary of each dataset was stored in a relational database which contains tables for classes, properties and relationships. The database also store in a table the set of values associated with each property of the dataset. The experiments are conducted on a standalone machine with an Intel i7-5600U@2.60GHz processors and 8GB memory.

---

[1] http://dbtune.org/jamendo/ (accessed on April 24, 2016)

[2] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/ (accessed on April 24, 2016)

[3] https://wordnet.princeton.edu/ (accessed on April 24, 2016)

[4] https://www.wikipedia.org/ (accessed on April 24, 2016)

Table 5.1 The Keyword Queries in the Two Datasets

Query #	Keywords	Structure of Pattern Graph
Jamendo Dataset		
J1	teenage, text, fantasie, Document	star-chain
J2	signal, onTimeLine, 10002, recorded_as, sweet	chain
J3	kouki, recorded_as, knees	star-chain
J4	briareus, reflectin, cool, girl	star
J5	kouki, revolution, electro, good	star
J6	nuts, spy4, chillout, track	star
J7	biography, guitarist, track, lemonade	chain
J8	divergence, track, obssession, format, mp32	star-chain
J9	fantasie, performance, recorded_as, slipstream	chain
J10	signal, recorded_as, fantasie, onTimeLine, 10001	chain
YAGO Dataset		
Y1	sonai, influences, poet, 1414, born, discovers, whirling	chain
Y2	dunderberg, interested, nyc, industrialist, influences, victor	star
Y3	richard, louis, pulitzer, award, american, book	star-chain
Y4	delhi, actor, shahrukh, acted, produced, india, films	cyclic
Y5	ridley, directed, gladiator, douglas, prize	cyclic
Y6	married, actor, wrestler, produced, directed, movie, tripper	cyclic
Y7	aristotle, influences, heliocentrism, astronomer, cambridge	star-chain
Y8	yoko, artist, grammy, huckleberry	star
Y9	neal, world, interface, cover, jensen	star-chain
Y10	grammy, sonny, produced, howard, created, westlife, songs	cyclic

Users were provided with different queries on those two datasets and in every instance they selected the most relevant pattern graph among those provided by the system. The queries are chosen in such a way so that the most plausible pattern graphs for the queries will have an empty answer when evaluated over the RDF data graph. We report on 20 queries (10 queries for each dataset). The queries cover a broad range of cases. They involve from 3 to 7 keywords, while the selected relevant pattern graphs form a star or a chain or a combination of them and in the case of YAGO dataset, they also form a cycle. Table 5.1 shows the keyword queries and information about their relevant pattern graph on both datasets.

5.4.2 Effectiveness in Ranking Relaxed Pattern Graphs

For our effectiveness experiments, we used expert users to determine the ground truth. For each query, the system produced the candidate pattern graphs. A user selected among them the pattern graph which is most relevant to the query. This is the original pattern graph. We then run our pattern graph relaxation algorithm until the third relaxed pattern graph with a non-empty answer is produced and collected the relaxed pattern graphs generated (which are many more). The generated relaxed pattern graphs are ranked by our system in relaxation order as described in Section 5.2.3. These relaxed pattern graphs are also provided to the user who ranks them based on their closeness to the original pattern graph. In order to measure the effectiveness of our technique in generating a ranked list of relaxed pattern graphs, we are using two metrics: (a) *normalized Discounted Cumulative Gain* (nDCG) [8], and (b) *Kendall tau-b rank correlation coefficient* [4]. Both of them allow comparing two ranked lists of items. Note that the list produced by the user and the one produced by our system might not form a strict total order. That is, there might be ties (relaxed pattern graphs with the same rank). We call a set of relaxed pattern graphs that have the same rank in a ranked list equivalence class of relaxed pattern graphs. Equivalence classes need to be taken into account in measuring the similarity of the ranked lists.

The nDCG metric was first introduced in [46] based on two key arguments: (a) highly important items are more valuable than marginally relevant items, and (b) the lower the position of the relevant item in the ranked list, the less valuable it is for the user because the less likely it is that the user will ever examine it. The first argument suggests that the relevance score of an item in the ranked list be used as a gained value measure. Then, the *cumulative gain* (CG) for position n in the ranked list is the sum of the relevance scores of the items in the ranked positions 1 to n. The second argument emphasizes that an item appearing at a lower position in the list should have a smaller share of its relevance score added to the cumulative gain. Hence, a discounting function is used over cumulative gain to measure *discounted cumulative gain* (DCG) for position n, which is defined as the sum

of the relevance scores of all the items at positions 1 to n, each divided by the logarithm of its respective position in the ranked list. The DCG value of a ranked list is the DCG value at position n of the list where n is the size of the list. The *normalized discounted cumulative gain* (nDCG) is the result of normalizing DCG with the DCG of the list that is correctly ranked (the ground truth list produced by the expert user). Thus, nDCG favors a ranked list which is similar to the correct ranked list. The DCG at n is given by the following formula:

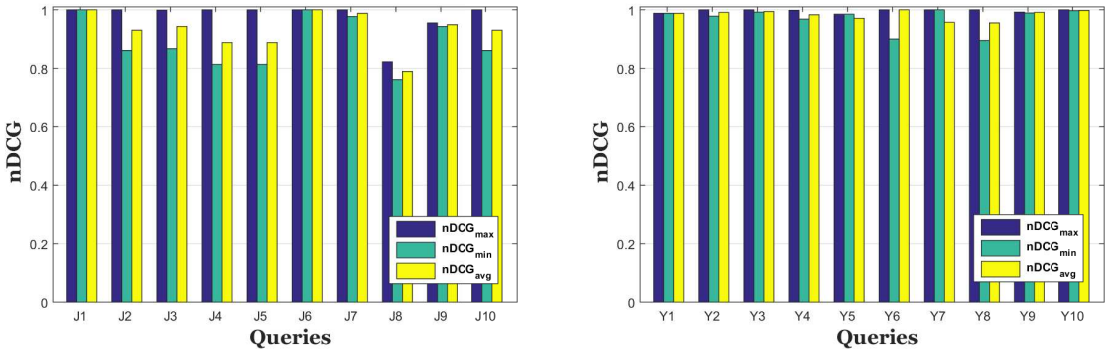
$$DCG_n = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

where rel_i, the relevance score of the item at position i in the ranked list, is the rank of this item's equivalence class in the inverse ground truth equivalence class list. For instance, if an item belongs to the 2nd equivalence class in a ground truth list of 5 equivalent classes, its relevance score is 3.

In order to take into account equivalent classes of pattern graphs in the system's ranked lists, we have extended nDCG by introducing minimum, maximum and average values for it. The nDCG_{max} value of a ranked list RL_e with equivalence classes corresponds to the nDCG value of a strictly ranked (that is, without equivalence classes) list obtained from RL_e by ranking the pattern graphs in the every equivalence classes correctly (that is, in compliance with their ranking in the ground truth list). The nDCG_{min} value of RL_e corresponds to the nDCG value of a strictly ranked list obtained from RL_e by ranking the pattern graphs in every equivalence classes in reverse correct order. The nDCG_{avg} value of RL_e is the average nDCG value over all strictly ranked lists obtained from RL_e by ranking the pattern graphs in every equivalence classes in all possible ways. The nDCG values range between 0 and 1.

Figure 5.7 shows the nDCG_{min}, nDCG_{max} and nDCG_{avg} values for the queries of Table 5.1 on the Jamendo and YAGO datasets. As one can see, all the values are very close to 1 and the min and max values of *nDCG* are close to each other.

The Kendall tau rank correlation coefficient [53] was proposed to address the prob-



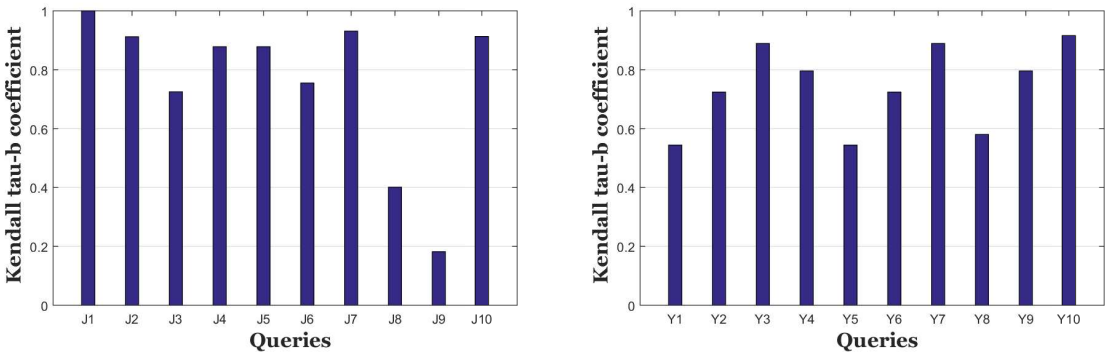
(a) Jamendo dataset (b) YAGO dataset

Figure 5.7 nDCG$_{max}$, nDCG$_{min}$ and nDCG$_{avg}$ for the queries on the Jamendo and YAGO datasets.

lem of measuring the association between two different rankings of the same set of items. For example, suppose that a set of items is given an order A which is correctly defined with reference to some quality q. An observer ranks the same set of items in an order B. A characteristic question that arises here is if the comparison of the orders B and A suggests that the observer possesses a reliable judgment of the quality q. In our context, we want to see if the comparison of the ranked list produced by our system (the relaxation order) with the correctly ranked list which is defined by the user suggests that the former possesses a reliable judgment of the closeness of the relaxed pattern graphs to the original pattern graph (which expresses the user's intention). However, the Kendall tau coefficient is useful when the ranked lists to be compared are strictly ranked. For this reason, we adopt here a variant called *Kendall tau-b coefficient* [4], which can deal with equivalent classes of items in the ranked lists. The Kendall tau-b coefficient is given by the following formula:

$$\tau_b = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{\sqrt{N_g} \times \sqrt{N_s}}$$

where N_g and N_s are the number of pairs of items which do not belong to the same equivalence class in the ground truth list and the system generated list, respectively. The value of τ_b ranges from -1 to 1. If two items have the same (resp. different) relative rank order in the two lists, then the pair is said to be *concordant* (resp. *discordant*) pair. If two items



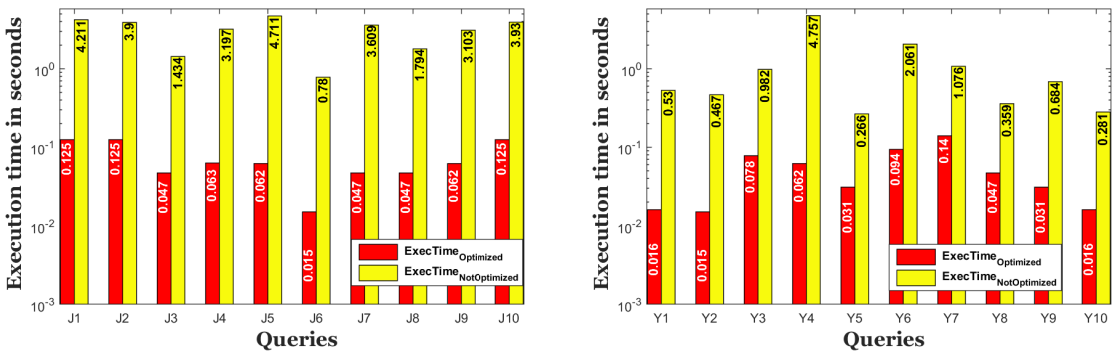
(a) Jamendo dataset (b) YAGO dataset

Figure 5.8 Kendall tau-b coefficient for the queries on the Jamendo and YAGO datasets.

are in an equivalence class in at least one of the lists then the pair is neither concordant nor discordant. If the number of concordant pairs is much larger than the number of discordant pairs, then the two lists are positively correlated (the coefficient is close to 1). If the number of concordant pairs is much less than the discordant pairs, then the two lists are negatively correlated (the coefficient is close to -1). Finally, if the number of discordant and concordant pairs are about the same, then the two lists are weakly correlated (the coefficient is close to 0). In this case, there is no association between the lists. Figure 5.8 shows the Kendall tau-b rank correlation coefficient for the queries of Table 5.1 on the Jamendo and YAGO datasets. As we can see, all the values are positive and in most cases very close to 1.

5.4.3 Efficiency of the System in Producing Relaxed Results

In order to asses the feasibility of our system, we ran our algorithm on the pattern graphs selected by the user for the queries of Table 5.1, and we measured the time needed to produce the first three consecutive nonempty relaxed pattern graphs and their answers. Many more relaxed pattern graphs are typically produced and ranked in the background, and a number of them are checked for empty answers. The queries were selected so that the original pattern graph for almost all of them has an empty answer. The Yago and the Jamendo datasets are stored in a relational database with one fully indexed relation for every distinct property and relationship in the datasets. To assess the efficiency of the



(a) Jamendo dataset (b) YAGO dataset

Figure 5.9 Efficiency improvement achieved by Multiquery optimization on the Jamendo and YAGO datasets.

system we evaluated the queries: (a) over the base relations with a cold cache, and (b) using the multi-query optimization and caching techniques presented in Section 5.3.3. Figure 5.9 shows the measured times. One can see that the displayed times for all the queries are interactive. Further, the optimization techniques are shown to substantially improve the execution time of the algorithm in most cases by more than one order of magnitude.

5.5 Conclusion

Exploiting the structural summary has emerged in recent years as a promising technique for evaluating keyword queries over RDF graphs. Structural summary-based approaches compute pattern graphs (structured queries) as possible interpretations of the unstructured keyword query and often rely on user feedback to identify the pattern graph which is most relevant to the user intent. However, since structural summaries are approximate representations of the data, these approaches might return empty answers or miss results which are relevant to the user intent. To address the drawback while maintaining the advantages of these approaches, we have presented a novel approach that permits the relaxation of the most relevant pattern graph selected by the user and expands its result space with similar results. We used pattern graph homomorphisms to introduce relaxed pattern graphs. We then defined an operation on pattern graphs and we proved that it is sound and complete with

respect to relaxed pattern graphs. In order to characterize the semantic closeness of relaxed pattern graphs to the original pattern graph, we introduced different syntax and semantic-based metrics that allow us to compare the degree of relaxation of relaxed pattern graphs. We identified the reasons for a pattern graph having an empty answer and we designed an algorithm which computes relaxed pattern graphs with non-empty answers in ascending relaxation order. We devised optimization techniques that exploit subquery caching and multiquery optimization to support the computation of relaxed pattern graphs. Our experimental results demonstrate the effectiveness of our approach in ranking the relaxed pattern graphs and the efficiency of our system and optimization techniques in producing relaxed pattern graphs and their answers.

CHAPTER 6

INCORPORATING COHESIVENESS INTO KEYWORD SEARCH

In the previous chapters 4, and 5, we proposed techniques that address the challenges posed by keyword search (Chapter 1). Our techniques take advantage of relevance feedback from the user in order to effectively identify relevant and high quality results. However, the users might not always be able or willing to provide feedback. Therefore, in this chapter we introduce a novel keyword query language that enables the user to convey his intent flexibly and effortlessly using cohesive keyword groups. A cohesive group of keywords in a query indicates that the keywords of the group should form a cohesive unit in the query results. We provide formal semantics of cohesive queries. We design a query evaluation algorithm which relies on the structural summary of the RDF graph to generate pattern graphs that satisfy the cohesiveness constraints. Pattern graphs are structured queries that can be evaluated over the RDF data to compute the query results. Our experiments demonstrate the efficiency of our algorithm and the effectiveness of cohesive keyword queries in improving the result quality and in pruning the space of pattern graphs compared to flat keyword queries. Most importantly, these benefits are achieved while retaining the simplicity and convenience of traditional keyword search.

6.1 Data Model and Flat Keyword Queries

Data Model. We follow the same RDF data model definition as in Chapter 3. Figure 6.1(a) shows an example RDF graph which is an excerpt from a large bibliographic RDF database. For simplicity, vertex and edge identifiers are not shown in the figure. All the other examples made in this chapter are based on this figure of the data graph.

Query Instance. A traditional keyword query Q on an RDF graph G is a set of keywords. An *instance* in G of a keyword k in Q is an occurrence of k (in a vertex or edge label) in G.

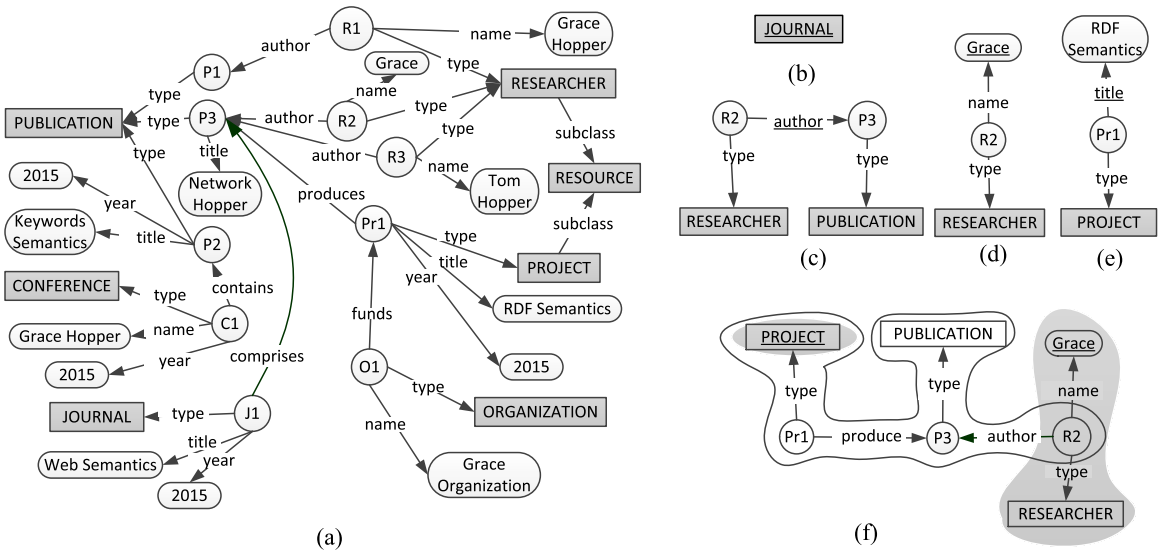


Figure 6.1 (a) An RDF Graph, (b), (c), (d) and (e) class, relationship, value and property matching constructs respectively, (f) inter-construct connection and query instance.

The *answer* of Q on G is a set of result graphs of Q on G.

In order to facilitate the interpretation of the semantics of the keyword instances, every instance of a keyword in Q is matched against a small subgraph of G which involves this keyword instance and the corresponding class vertex or vertices. This subgraph is called *matching construct* as defined in Chapter 3. Figures 6.1(b), (c), (d) and (e) show a class, relationship, value and property matching construct, respectively, for different keyword instances in the RDF graph of Figure 6.1(a). Underlined labels in a matching construct denote the keyword instances based on which a matching construct is defined. Each matching construct provides information about the semantic context of the keyword instance under consideration. For example, the matching construct of Figure 6.1(d) shows that `Grace` is the name of an entity $R2$ of type Researcher.

A *signature* of Q is a function that matches every keyword k in Q to a matching construct for k in G. Given a query signature S, an *inter-construct connection* between two distinct matching constructs C_1 and C_2 in S is a simple path augmented with the class vertices of the intermediate entity vertices in the path (if not already in the path). The formal definition is stated in Chapter 3. Figure 6.1(f) shows an inter-construct connection

between the matching constructs for keywords `Project` and `Grace` in the RDF graph of Figure 6.1(a). The matching constructs are shaded and the inter-construct connection is circumscribed. A subgraph of G is said to be *connection acyclic* if there is no cycle in the graph obtained by viewing its matching constructs as vertices and its inter-construct connections between them as edges. Given a signature S for Q on G, an *instance* of S on G is a connected, connection acyclic subgraph of G which contains only the matching constructs in S and possibly inter-construct connections between them. An *instance* for Q on G is an instance for a signature of Q on G. Figure 6.1(f) shows an instance for the query `{Grace, Project}` on the RDF graph of Figure 6.1(a). The instances of a flat query Q on G are all considered to be results of Q that together form the answer of Q on G. As we will see in the next section, if a query Q' has the same keyword occurrences as Q and involves in addition, cohesive keyword groups, only some of these instances are considered to be results that form the answer of Q' on G. The rest of the query instances are excluded as irrelevant. Therefore, the instances of Q' are its candidate results.

6.2 Keyword Queries with Cohesive Keyword Groups

We define in this section the syntax and semantics of keyword queries with cohesive keyword groups (called *cohesive keyword queries*).

6.2.1 Syntax

We start by providing a recursive definition of the concept of term which corresponds to a cohesive group of keywords: a *term* is a set of at least two keywords and/or terms.

Definition 6.2.1 (Cohesive Query)**.** A cohesive query is: (a) a set of a single keyword, or (b) a term. *Notation*: sets are delimited in a query using parentheses, and elements are separated within sets using spaces.

For instance, Q_1 =`(Publication (Grace Hopper) (Project`

`Semantics 2015))` is a cohesive keyword query and `(Grace Hopper)` and `(Project Semantics 2015)` are two terms in it. Query Q_2 = `((RDF Project) publication) (author (Tom Hopper)))` is another cohesive keyword query where the term `(Tom Hopper)` is nested within the term `(author (Tom Hopper))` and the term `(RDF Project)` is nested within the term `((RDF Project) Publication)`.

The same keyword may appear multiple times in a query but, of course, the same keyword or term cannot appear multiple times as an element of a set. For instance, in the cohesive keyword query Q_3 = `((author Grace) Publication (Conference (Grace Hopper)))`, the keyword `Grace` occurs twice: once in the term `(author Grace)` and once in the term `(Grace Hopper)`. In the following unless stated differently, 'query' refers to a cohesive keyword query.

6.2.2 Semantics

The queries are matched against a data graph G. An instance I of a cohesive query Q on G is defined similarly to an instance of the flat query that involves the same keywords. A difference appears only when Q involves multiple occurrences of the same keyword k. In this case, I might contain multiple instances of k and the occurrences of k in Q can be matched to the same or different instances of k in I.

Figures 6.2(a), (b) and (c) show different instances of the query `(Publication (Project (Semantics 2015)) (author (Grace Hopper)))` on a bibliography database. This bibliography database encompasses the one of Figure 6.1 and is not shown here for the sake of space.

As mentioned above, a term in a cohesive query contains keywords and/or other terms. A term expresses a cohesiveness relationship on its elements. Intuitively, a term states that the instances of its keyword occurrences in a result of the query should form a *cohesive unit*. That is, in a result graph of a query, they should form a subgraph where the

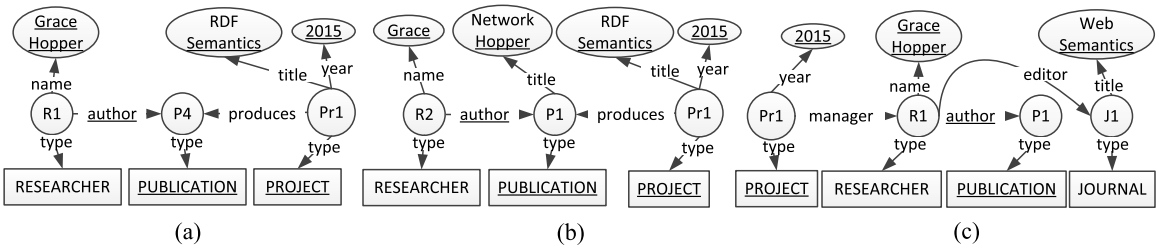


Figure 6.2 (a) result graph, (b) and (c) query instances which are not result graphs for the query `(Publication (Project (Semantics 2015)) (author (Grace Hopper)))`.

instances of the keyword occurrences which are external to the term do not interfere. More formally, let I be an instance of a query Q on G, and t be a term in Q. The *instance I_t of t in I* is a minimal connected subgraph of I that comprises the matching constructs of the keyword occurrences of t in I.

Definition 6.2.2 (Query result)**.** A *result* of Q on G is an instance I of Q on G such that for every term t in Q and for every keyword occurrence k in Q which is not in t, the following conditions hold:

(a) The instance of k in I does not occur in I_t unless it is a class vertex label,

(b) The instance of k in I is not the label of a property edge or a value of an entity vertex in I_t unless this entity vertex in I_t is incident to only one non-type edge (that is, only one relationship or property edge).

The *answer* of a query Q on G is the set of the results of Q on G.

Consider again the query and the query instances of Figure 6.2. With this query the user is looking for publications authored by Grace Hopper which were produced by a project on Semantics that started in 2015. The query instance of Figure 6.2(a) is a result graph for this query as it satisfies the conditions of Definition 6.2.2. In contrast, the query instance of Figure 6.2(b) is not a result graph for the query. Indeed, the instance of the keyword `author`, which is not in the term `(Grace Hopper)`, occurs within the instance of this term in the query instance (condition (a) in Definition 6.2.2). Similarly, the query instance of Figure 6.2(c) is not a result graph of the query since the instances of keyword

`Grace` and `Hopper`, which are not in the term `(Semantics 2015)`, are values of an entity vertex (vertex R1) in the instance of this term in the query instance (condition (b) in Definition 6.2.2).

6.3 An Algorithm for Evaluating Cohesive Queries

In this section, we describe an algorithm for evaluating cohesive keyword queries over RDF graph data. Our algorithm follows a recent trend which exploits a structural summary of data to compute pattern graphs [22, 89]. These pattern graphs represent different interpretations of the imprecise keyword query and are, in fact, structured queries that can be evaluated against the RDF data graph to compute the keyword query answer. Structural summaries are typically much smaller than the actual RDF data. Therefore, the pattern graphs can be generated efficiently. Moreover, this process scales smoothly when the size of the data increases. Our algorithm computes pattern graphs which are r-radius Steiner graphs and satisfy the cohesiveness of the terms in the keyword query. The algorithm proceeds bottom up in the cohesive query hierarchy to prune the search space of pattern graphs by excluding early on pattern subgraphs that breach the cohesiveness of terms (cohesive keyword groups) in the query.

6.3.1 Structural Summary and Pattern Graphs

Roughly speaking, the structural summary is a graph which summarizes an RDF graph. The details of structural summary construction and definition in provided in Chapter 3. Figure 6.3(a) shows the structural summary for the RDF graph G of Figure 6.1(a). Similarly to matching constructs on the data graph, we define matching constructs on the structural summary. Since the structural summary does not have entity vertices, a matching construct on a structural summary possesses one distinct entity variable vertex, for every class vertex, labeled by a distinct variable.

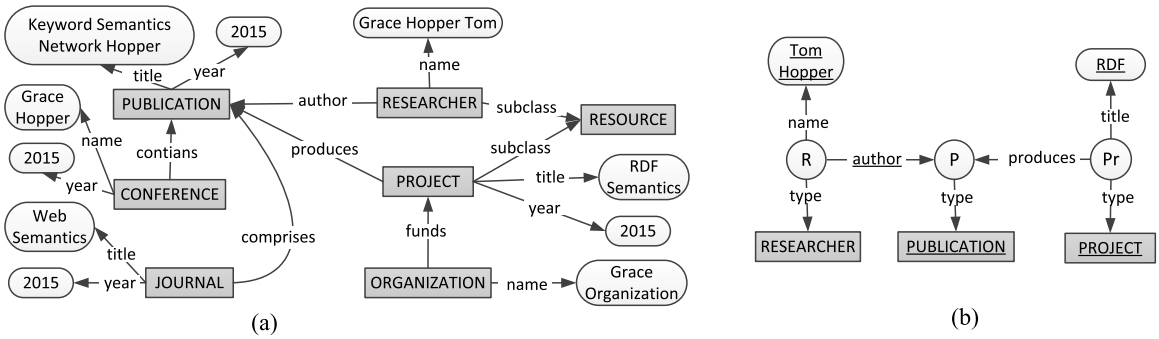


Figure 6.3 (a) Structural Summary (b) Query Pattern Graph.

Pattern graphs are subgraphs of the structural summary strictly consisting of one matching construct for every keyword in a query Q and the connections between them without these connections forming a cycle. Formal definition of a pattern graphs is stated in Chapter 3.

Figure 6.3(b) shows an example of a pattern graph for the query $Q_2 =$ `(((Project RDF) publication) (author (Tom Hopper)))` on the structural summary graph of Figure 6.3(a).

The pattern graphs of a cohesive query satisfy or violate the cohesiveness of its terms specified in it in the same way the instances of the query do. As mentioned above, pattern graphs are structured queries. Those pattern graphs that satisfy the cohesiveness of the query terms can be used to compute the results of the query. Interestingly, pattern graphs can be expressed as SPARQL queries, and all the machinery of query engines and optimization techniques developed for SPARQL can be leveraged to efficiently compute the results.

6.3.2 The Basic Components of the Algorithm

Our algorithm proceeds by first parsing the cohesive query. Then, it uses the produced query hierarchy to incrementally construct r-radius Steiner pattern graphs. During the process of pattern graph generation, it checks whether the pattern graph under construction

satisfies the cohesiveness constraints.

(a) Parsing the Query. The parsing of a cohesive query produces a parse tree. An example of such a parse tree for the query $Q_2 =$`(((Project RDF) publication) (author (Tom Hopper)))` is shown in Figure 6.4(a). The leaf vertices of the parse tree are labeled by the query keyword occurrences. The root represents the query and the internal vertices represent the query terms. The *level* of a vertex in the tree is the number of edges on its path from the root and the *height* of the tree is the number of edges in the longest path from root to leaf.

(b) Computing *r*-radius Steiner Graphs on the Structural Summary. Given a set of query keyword matching constructs and/or query term instances on the structural summary, the algorithm identifies a connecting vertex *cv* in the structural summary such that the distance between *cv* and any one of the vertices in the matching constructs and term instances is no more than *r*. The algorithm chooses the smallest *r* for the connecting vertex. There can be more than one connecting vertex connected to the matching constructs and term instances with paths of length *r* or less. There can also be different ways of connecting the same connecting vertex with all the matching constructs and term instances with paths of length *r* or less. All the alternative ways to link the connecting vertices to the matching constructs and term instances define alternative *r*-radius Steiner graphs. Given a term *t* in a query, we use the algorithm presented in [22] to compute all the *r*-radius Steiner graphs with minimal *r* for the matching constructs and the term instances corresponding to the keywords and nested terms, respectively, of *t*. This algorithm extends the one in [63], which computes *r*-radius Steiner graphs on general graphs, to allow for keyword instances on the edges. Once the *r*-radius Steiner graphs for a term of a query are computed, they can be used for computing the r-radius Steiner graphs of the parent term in the tree.

(c) Checking Cohesiveness Semantics. Given a set of matching constructs for the keywords and the term instances for the nested terms of a term *t*, the algorithm checks whether

Algorithm 3 : CohesivePGGen (Cohesive Pattern Graph Generation)

Input: Q: a cohesive keyword query.
Output: a set of pattern graphs.

```
1: for every keyword k ∈ Q do
2:     I_k ← set of matching constructs of k on the structural summary;
3: Δ ← ParseQuery(Q);
4: l = height(Δ);
5: while l ≥ 0 do
6:     for every vertex n at level l of Δ do
7:         if n is a leaf node labeled by keyword k then
8:             I_n = I_k
9:         if n is a term or the root of the tree then
10:            L ← I_{c_1} × ... × I_{c_m};      ▷ c_1, ..., c_m are the children of n.
11:            if n contains a term then
12:                for every combination L_i ∈ L do
13:                    if CheckCohessivenessSemantics(L_i) = false then
14:                        L ← L − L_i;
15:            for every combination L_i ∈ L do
16:                I_n ← I_n ∪ rRadiusSteinerGraphs(L_i);   ▷ I_n is the set of instances
                                                              of term n on the structural summary
17:    l ← l − 1
18: Return I_n
```

any two elements of the set overlap in a way that breaches the cohesiveness of t (Definition 6.2.2). If this is the case, the algorithm discards this set of matching constructs and term instances for term t, and does not use it to construct minimal r-radius Steiner graphs to be propagated to the parent term of t in the query parse tree.

6.3.3 Algorithm Description

Our Algorithm, called CohesivePGGen (Cohesive Pattern Graph Generation), is outlined in Algorithm 3. It takes as input a cohesive keyword query Q, and outputs a set of r-radius Steiner pattern graphs which satisfy the cohesiveness semantics. We assume that the structural summary of the RDF data graph is available. Initially, the algorithm computes the matching constructs for all the keywords in Q over the structural summary (lines 1-2), parses the query into the parse tree Δ (line 3), and instantiates a variable l representing the level of a node in Δ to the height of Δ (line 4). The algorithm constructs pattern graphs

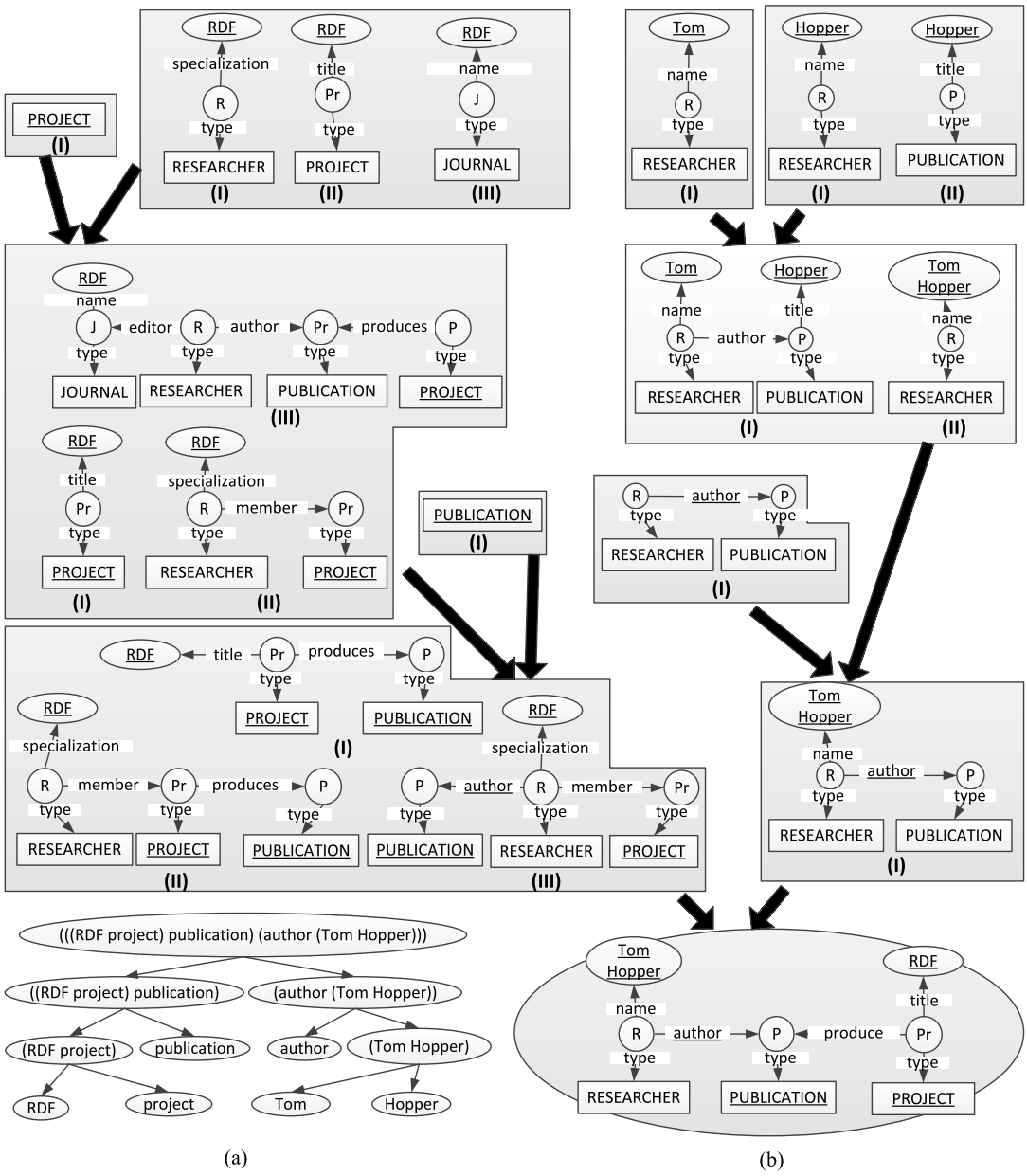


Figure 6.4 (a) Parse tree (b) Generation of pattern graphs of the query by the algorithm.

incrementally, in a bottom up manner over the parse tree Δ, starting with the deepest leaf vertices (lines 5-17). For a vertex n, variable I_n represents the set of matching constructs if n is a leaf (keyword) vertex (lines 7-8), and the set of term instances of n on the structural summary if n is a term or the root of Δ (lines 9-16). For a term vertex n, variable L denotes the Cartesian product of the sets I_{c_i} for the children c_i of n (line 10). Every element of L which violates the cohesiveness of at least one term instance in it is removed from L (lines

11-14). The rest of the elements of L are used to produce r-radius Steiner graphs with minimal r which are instances of the term vertex n (lines 15-16). The process continues until the root vertex is reached. At this point, I_n represents the pattern graphs of Q, which are returned to the user.

Figure 6.4 exemplifies the construction of the pattern graphs for the query $Q_2 =$ `(((Project RDF) publication) (author (Tom Hopper)))`.

6.4 Experimental Evaluation

We implemented our approach and ran experiments to evaluate: (a) the effectiveness of cohesive keyword queries, and (b) the efficiency of CohesivePGGen.

6.4.1 Datasets and Queries

We used DBLP and Jamendo[1] real datasets for our experiments. DBLP is a bibliography database of 600MB of size, containing 8.5M triples. Jamendo is a repository of Creative Commons licensed music of 85MB of size, containing 1.1M triples. The extracted structural summaries and the keyword inverted lists for both datasets were stored in a Relational database. The experiments were conducted on a standalone machine with an Intel i5-3210M @2.5GHz processor and 8GB memory.

We experimented with a large number of cohesive keyword queries and we report on 10 of them for each dataset. The queries cover a broad range of cases. They involve 4 to 6 keywords and 1 to 3 levels of term nesting. Table 6.1 shows the queries used and their statistics. #MCs denotes the total number of matching constructs for the keywords of a query, #Sigs denotes the total number of matching constructs combinations for a query (signatures), and #PGs denotes the number of pattern graphs of a query on the structural summary ignoring the cohesiveness semantics.

[1] http://dbtune.org/jamendo/ (accessed on April 24, 2016)

Table 6.1 Queries on Jamendo and DBLP Datasets

Dataset	Q#	Cohesive keyword query	#MCs	#Sigs	#PGs
Jamendo	1	(document (teenage (text fantasie)))	16	105	162
	2	((lyrics sweet) recorded_as onTimeLine)	18	64	64
	3	((MusicArtist Cicada) performance (track knees))	21	405	488
	4	((Record (date title)) track (Lyrics good))	43	127,008	185,916
	5	((MusicArtist Briareus) (cool (girl Reflections)))	27	1,440	1,950
	6	(time Mako (record (track (down passion))))	39	39,690	49,070
	7	(Kouki (electro (record revolution (track good))))	43	119,070	172,541
	8	(Nuts track (chillout (record spy4)))	28	1,764	2,248
	9	((biography guitarist) (track (title Lemonade)))	25	1,512	2,538
	10	((record (title divergence)) (track obsession))	27	1,323	1,805
DBLP	1	((journal design) creator (person (phdthesis CAD)))	49	69,120	447,086
	2	((name Charles) creator (Proceedings forward))	33	68,200	25,324
	3	((article editor person) creator (inproceedings hybridization))	32	4,320	12,519
	4	((person name) creator (performance 2002))	59	100,800	479,542
	5	(Oliver (Article (Linux year))	21	480	2,960
	6	(inproceedings Tolga (mastersthesis warehouses))	8	5	22
	7	(((compiler cite) Charles) (creator peephole))	34	5,280	20,660
	8	(creator (decentralized IEEE) (coscheduling 2004))	51	25,300	141,531
	9	((Milne 2005) homepage person)	35	1,320	3,788
	10	(((name Yahiko) person) (editor (conceptual springer)))	34	18,900	97,512

6.4.2 Effectiveness of the Cohesive Queries

In order to evaluate the effectiveness of the cohesive queries, we measured: (a) the reduction in the number of pattern graphs of a query, and (b) the improvement in the quality of the results of a query due to the cohesiveness constraints.

(a) Reduction in the Space of Pattern Graphs. We compare, for each cohesive keyword query, the number of pattern graphs generated with the number of pattern graphs of the corresponding flat keyword query (i.e., the flat keyword query obtained by removing cohesiveness constraints and keyword duplicates). Figure 6.5 reports on the percentage reduction of the number of pattern graphs for the queries of Table 6.1 on both datasets. As one can see, the cohesiveness constraints reduce substantially the number of pattern graphs from which the user has to choose the relevant ones.

(b) Improvement in the Quality of Results. The number of pattern graphs of a query can be very large in order to allow an expert user go through them and select the relevant ones.

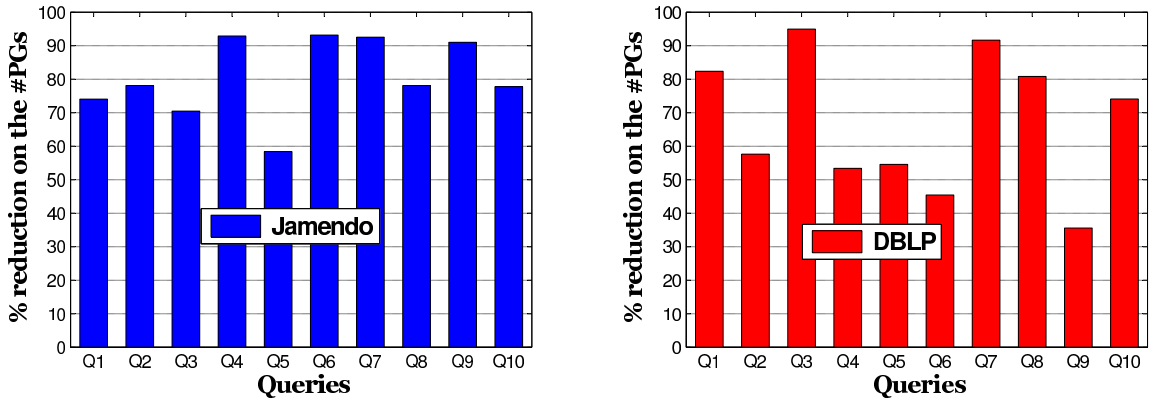


Figure 6.5 % reduction on the number of pattern graphs for the queries on the two datasets.

For instance, observe that in Table 6.1 some queries have hundreds of thousands of pattern graphs. Therefore, we adopt the *path length* and *popularity score* metrics introduced in [89] to rank the pattern graphs of a query. We then select the top-k pattern graphs and have an expert user identify those of them which are relevant. In order to measure the quality of the results, we use the *precision@k* ($p@k$) metric. The *precision@k* is the ratio of the number of relevant pattern graphs in the first k positions to k. Figure 6.6 displays the average $p@k$ over the queries of Table 6.1, for different values of k, on the two datasets. For comparison, the figure displays both: the average $p@k$ of the cohesive queries and the average $p@k$ of their corresponding flat queries. The results show that in all cases, the quality of the cohesive queries is several times higher than that of their corresponding flat queries. This

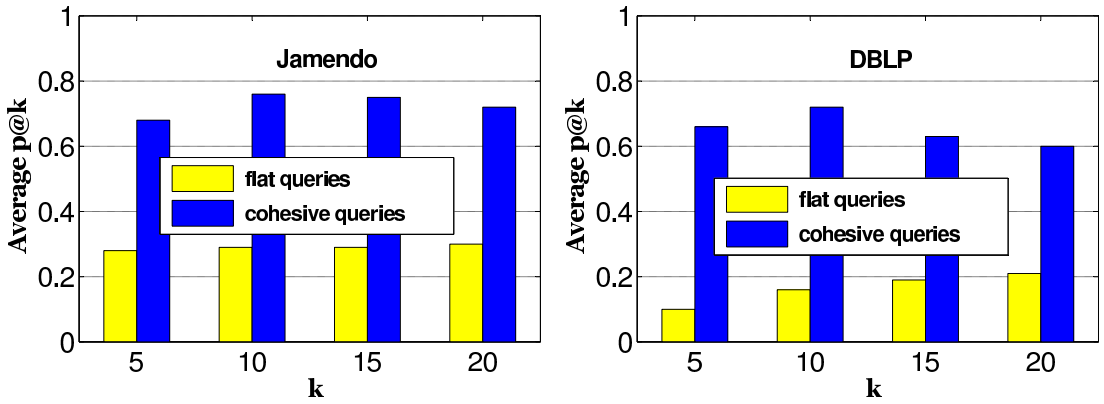


Figure 6.6 Average *precision@k* for cohesive queries and their corresponding flat queries varying k on the two datasets.

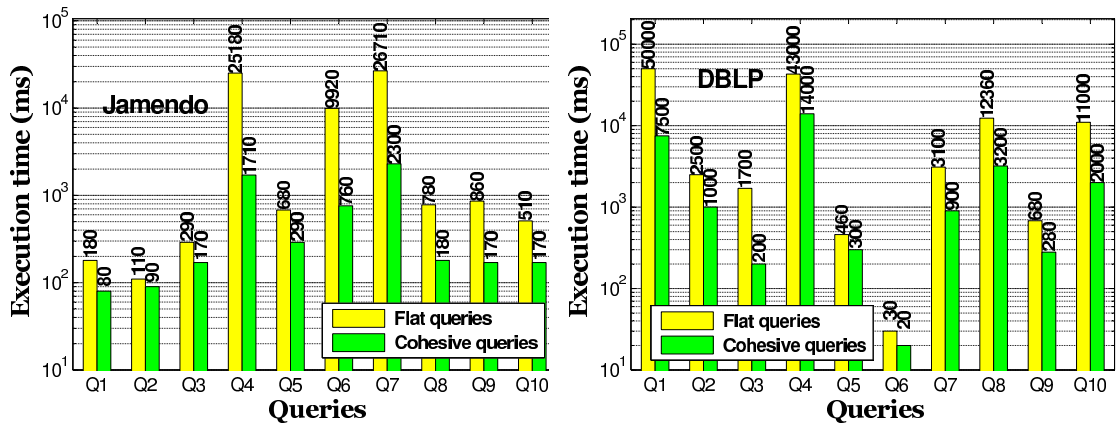


Figure 6.7 Execution time of CohesivePGGen on cohesive and flat queries.

is not surprising, since the cohesive queries benefit from the cohesiveness constraints that the user specified expressing her intention.

6.4.3 Efficiency of Algorithm CohesivePGGen

We compare the execution time of our algorithm on cohesive queries with the computation time of the pattern graphs of their corresponding flat keyword queries. Figure 6.7 shows the execution times of CohesivePGGen for the queries of Table 6.1 on the two datasets. Note that the Y axis is in logarithmic scale. Algorithm CohesivePGGen on cohesive queries is much faster, in some cases by more than one order of magnitude. In fact, algorithm CohesivePGGen on cohesive queries has to check for the satisfaction of cohesiveness constraints and this incurs additional cost. However, the algorithm does not produce all the pattern graphs of the flat version of the query to check if they satisfy the cohesiveness constraints. Instead, it stops the construction of a pattern graph as soon as the cohesiveness of a term is violated, and this early pruning of the search space ultimately pays off.

6.5 Conclusion

In this paper we claim that without additional information from the user, keyword queries cannot effectively retrieve information from RDF graph data. Therefore, we introduce a

novel keyword query language which allows the user to better express her intention by permitting the specification of cohesive keyword groups, keyword group nesting and keyword repetition. We provide formal semantics for cohesive keyword queries, and we design a query evaluation algorithm, called CohesivePGGen, which exploits the structural summary of the RDF graph to produce r-radius Steiner pattern graphs. Our algorithm prunes early on the search space of pattern graphs by retaining only those that satisfy the cohesiveness constraints. Our experiments show that CohesivePGGen largely outperforms the generation of pattern graphs for flat keyword queries. They also show that cohesive queries substantially improve the *precision*@k of flat keyword queries allowing the search for relevant pattern graphs in a much smaller set while retaining the simplicity and convenience of flat keyword queries.

CHAPTER 7

INTRODUCING DIVERSITY IN THE RESULT SETS OF KEYWORD QUERIES

Keyword queries are vague representations of users' information needs. In Chapter 1, we discussed different challenges posed by keyword search because of its ambiguous nature. Chapters 4, 5, and 6 address these challenges and tried to identify relevant results by filtering out irrelevant results with the enforcement of structural constraints, or by ranking the results using scoring functions, or by leveraging relevance feedback from the users. Although, these approaches are successful in retrieving high quality relevant results, they still might fail to capture users' intent as they ignore the diversity of the result interpretations. To address this issue, in this chapter, we introduce a novel technique for balancing the relevance and diversity of keyword search results on RDF graph data. We generate pattern graphs which are structured queries corresponding to alternative interpretations of a given keyword query. We model the problem as an optimization problem aiming at selecting k pattern graphs which maximize an objective function on relevance and diversity. We devise measures to estimate the diversity of a set of pattern graphs and its relevance to the user query. We design an algorithm that employs a greedy heuristic to generate a list of k relevant and diverse pattern graphs for a given keyword query. Our experimental results show that our relevance and diversity measures are effective and our algorithm can efficiently compute a list of top-k pattern graphs.

7.1 Data Model and Pattern Graph Computation

The same definition of the data model, and the notions of structural summary, matching constructs on structural summary, inter-construct connections, signature and pattern graphs as defined in Chapter 3 are exploited in this chapter. Figure 7.1(a) shows an example of an RDF data graph. This RDF data graph has been used to draw all the other examples of

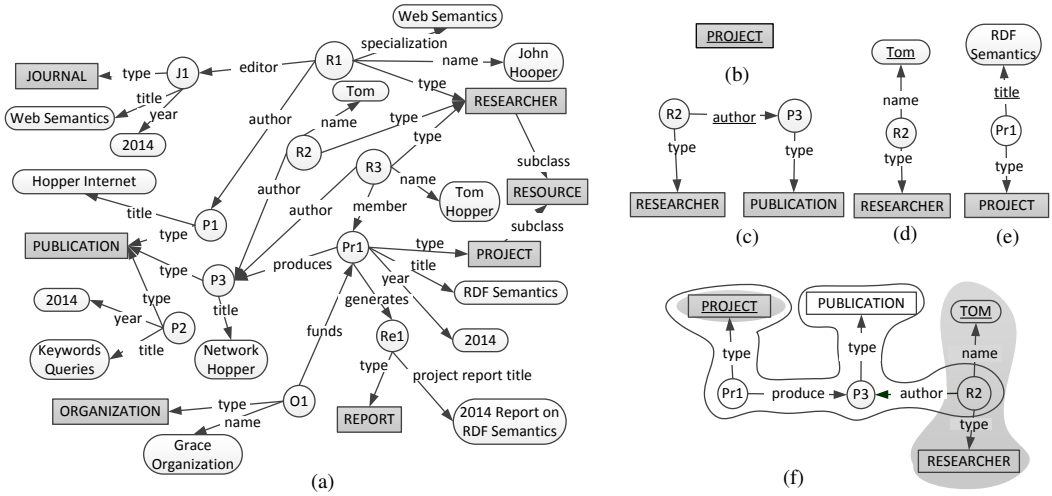


Figure 7.1 (a) An RDF graph, (b), (c), (d) and (e) class, relationship, value and property matching constructs, respectively, (f) inter-construct connection and result graph.

this chapter. Figure 7.1 also includes examples of different matching constructs (Figures 7.1(b), (c), (d) and (e)) and an inter-construct connection (Figure 7.1(f)) over the RDF data graph in Figure 7.1(a).

We compute pattern graphs of a keyword query on the structural summary of an RDF data graph. An example of structural summary, matching constructs on the structural summary and a pattern graph are shown in Figure 7.2. Figure 7.2(a) is the structural summary of the RDF data graph shown in Figure 7.1(a). Figures 7.2(b), (c), (d), and (e) are the matching constructs for the keywords `Project`, `author`, `Tom`, and `title` of the

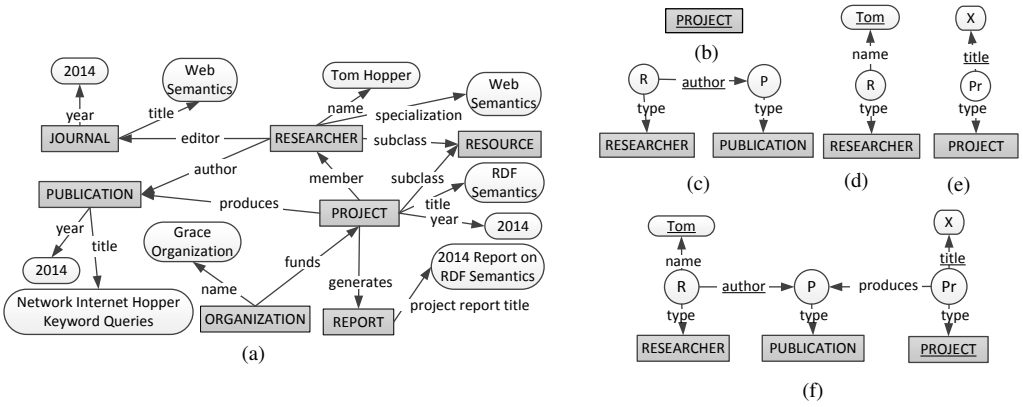


Figure 7.2 (a) Structural Summary G', (b), (c), (d), and (e) are matching constructs for keywords in the keyword query `Q1={Tom, author, Project, title}` on G', (f) Pattern Graph of Q on G'.

keyword query `Q1={Tom, author, Project, title}` on the structural summary of Figure 7.1(a), respectively. The keywords are underlined in the matching constructs. The pattern graph shown in Figure 7.1(f) is computed on the structural summary of Figure 7.1(a) for the query `Q1` and consists of the matching constructs shown in the Figures 7.1(b), (c), (d), and (e).

7.2 Balancing Relevance and Diversity

We provide in this section a formal definition of the problem we address in this chapter and then elaborate on its components: how to assess the relevance and the diversity of sets of pattern graphs.

7.2.1 Problem Statement

Our goal is to provide the user with a set of pattern graphs which is relevant and diverse. To this end, we define the problem as an optimization problem. Let G denote an RDF data graph, Q be a keyword query on G, $\mathscr{P}$ be the set of pattern graphs of Q on G and k be a positive integer. Given a subset $\mathscr{S}$ of P, let $relevance(\mathscr{S},Q)$ denote the relevance of $\mathscr{S}$ with respect to Q, and $diversity(\mathscr{S})$ denote the diversity of set $\mathscr{S}$. We aim at selecting a subset $\mathscr{S}$ of $\mathscr{P}$ which maximizes the objective function $\alpha * relevance(\mathscr{S},Q) + (1-\alpha) * diversity(\mathscr{S})$, where $\alpha \in [0,1]$, is a parameter which tunes the importance of relevance and diversity. In other words,

$$\mathscr{S} \in \underset{\mathscr{S}' \subseteq \mathscr{S}, |\mathscr{S}'|=k}{\operatorname{argmax}} (\alpha * relevance(\mathscr{S}',Q) + (1-\alpha) * diversity(\mathscr{S}'))$$

The tuning parameter α allows us to give more importance to the relevance or the diversity of the pattern graph set to be selected. If $\alpha = 1$, the selected pattern graph set will contain the most relevant pattern graphs without considering diversity. If $\alpha = 0$, the pattern graph set will be selected solely based on its diversity.

7.2.2 Assessing the Relevance of a Pattern Graph Set

We show now how the relevance of a set of pattern graphs is estimated. Our approach exploits statistical information for the popularity (frequency) of the class and value vertices and the property and relationship edges of the pattern graphs in the RDF graph. In doing so, it also takes into account structural and semantic information of the pattern graphs. In this sense, two edges with the same label are different if they involve entity variable vertices of different types. In order to assess the popularity of value vertices with keyword instances in the pattern graph we employ the well known *tf*idf* (term frequency, inverse document frequency) metric [82] of Information Retrieval(IR) adapted to the syntactic and semantic features of the RDF data model.

Consider a pattern graph P over an RDF data graph G. Let $C_1, \ldots, C_n$ be the class vertex labels in P. Let also $|V_{C_i}|$ denote the number of entities of type C_i in the RDF graph G, and $|V_E|$ denote the total number of entities in G. The popularity of the class vertices of P is given by the formula:

$$pop_c(P) = 1/n * (\sum_{C_i \in \{C_1,\ldots,C_n\}} |V_{C_i}|/|V_E| \quad)$$

Let $P_1, \ldots, P_m$ denote the distinct (owner class vertex, property edge label) pairs in P. Let also $|E_{P_i}|$ denote the number of property edges complying with P_i in the RDF graph G, and $|E_P|$ denote the total number of property edges in G. The popularity of the property edges of P is given by the formula:

$$pop_p(P) = 1/m * (\sum_{P_i \in \{P_1,\ldots,P_m\}} |E_{P_i}|/|E_P| \quad)$$

Let $R_1, \ldots, R_u$ denote the distinct (domain class vertex, relationship edge label, range class vertex) triples in P. Let also $|E_{R_i}|$ denote the number of relationship edges complying with R_i in the RDF graph G, and $|E_R|$ denote the total number of relationship

edges in G. The popularity of the relationship edges of P is given by the formula:

$$pop_r(P) = 1/u * (\sum_{R_i \in \{R_1,\ldots,R_u\}} |E_{R_i}|/|E_R| \quad)$$

For defining the popularity of value vertices with keyword instances in a pattern graph, we modify the *tf*idf* metric so that it applies to RDF graphs as explained below. The metric *tf*idf* used in IR reflects how important a term is to a document in a corpus of documents. tf denotes the frequency of a term in a document while idf is the logarithmically scaled inverse fraction of the documents that contain the term. In the context of an RDF graph G, a document corresponds to the set of property edges in G which have the same label L and are incident to entity vertices of type C. This set of property edges is denoted by $E(C,L)$. Given a keyword k_i and a set of property edges $E(C,L)$, let $E(k_i,C,L)$ be the subset of $E(C,L)$ which contains only those property edges whose value comprises k_i. Then:

$$tf(k_i,E(C,L)) = |E(k_i,C,L)|/|E(C,L)|$$

Let W denote the set of all property edge sets $E(C,L)$ in G. For a given keyword k_i, let W_i be the subset of W consisting of those property edge sets $E(C,L)$ such that $tf(k_i,E(C,L)) > 0$ (that is, property edge sets where k_i occurs in the value of at least one of their property edges). Then:

$$idf(k_i) = log(|W|/|W_i|)$$

Let $k_1,\ldots,k_j$ denote the keywords which appear in the labels of value vertices in a pattern graph P. Note that, multiple keywords can appear in the label of a value vertex in P. Let v_i denotes the value vertex whose label contains the keyword k_i and L_i is label of the property edge connecting v_i to an entity variable vertex of type C in P. Therefore, k_i also appears in the values for the set of property edges for (C_i,L_i) pair. Then, the popularity of value vertices containing keywords in P is given by the formula:

$$pop_v(P) = 1/j * (\sum_{k_i \in \{k_1,\ldots,k_j\}} tf(k_i, E(C_i, L_i)) * idf(k_i) \quad)$$

We define the relevance of pattern graph P to keyword query Q as the sum of the popularity of the components of P as follows:

$$relevance(P,Q) = 1/4 * (\sum_{i \in \{c,p,r,v\}} pop_i(P) \quad)$$

Clearly, the values of $pop_i(P)$ are in range $[0,1]$. By dividing the sum by 4, we guarantee that $relevance(P,Q)$ is also between 0 and 1.

We assume that the relevance of one pattern graph is independent of the relevance of another pattern graph. The relevance of a set of pattern graphs S of size k to a keyword query Q is the average relevance of its pattern graphs:

$$relevance(S,Q) = 1/k * (\sum_{P \in S} relevance(P,Q))$$

7.2.3 Assessing the Diversity of a Pattern Graph Set

In order to measure the diversity of a set of pattern graphs for a keyword query, we introduce a distance metric to measure the similarity of two pattern graphs. Our metric takes into account both structural and semantic features of the pattern graphs.

The first factor we consider in assessing the distance of two pattern graphs is the similarity of their matching constructs. Remember that the matching constructs are small graphs that involve only a single keyword instance and provide a context for interpreting the keywords. Given a pattern graph P for a keyword query $Q = \{k_1, \ldots k_n\}$, let $mc(P)$ denote the set of matching constructs of Q—one for every keyword in Q. The larger the number of keywords which are interpreted in the same way in the two pattern graphs, the more similar the pattern graphs are. The similarity of the matching constructs in the two

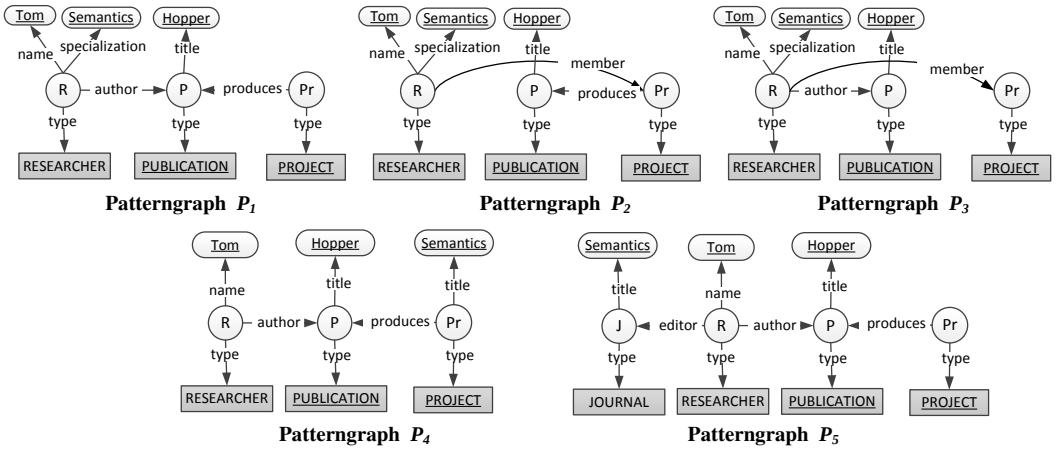


Figure 7.3 Five pattern graphs for the keyword query `Q={Tom, semantics, publication, hopper, project}` on the structural summary of Figure 7.2(a).

pattern graphs is given by the formula

$$mc\_sim(P_1, P_2) = (|mc(P_1) \cap mc(P_2)|)/n$$

where n is the number of matching constructs in $mc(P_1)$ or $mc(P_2)$. Clearly, $mc\_sim(P_1, P_2) = 1$ if P_1 and P_2 share the same matching constructs, and $mc\_sim(P_1, P_2) = 0$ if they have no common matching constructs.

For instance, Figure 7.3 shows 5 pattern graphs of a query with 5 keywords. Intuitively, P_2 and P_3 are more similar to P_1 than P_4 and P_5 because P_4 and P_5 interpret the keyword `semantics` differently. Metric *mc_sim* catches this intuition since $mc\_sim(P_1, P_2) = mc\_sim(P_1, P_3) = 5$ while $mc\_sim(P_1, P_4) = mc\_sim(P_1, P_5) = 4$.

Although P_2 and P_3 have the same common matching constructs with P_1, P_2 looks more similar to P_1 than P_3 does. Therefore, we consider the second factor which is to what extent matching constructs for the same keywords are connected in the same way in the two pattern graphs. The higher the number of pairs of keywords in P_1 and P_2 whose matching constructs are connected in the same way in the two pattern graphs, the more similar P_1 and P_2 are. Of course, if the matching constructs of two keywords are not the same in P_1 and P_2, their connections cannot be the same in the two pattern graphs and this pair of keywords

does not contribute to the similarity of P_1 and P_2. We define a connection between two keywords k_i and k_j of Q in a pattern graph P of Q as a graph consisting of the matching constructs of k_i and k_j, respectively, and a simple path between these matching constructs in P augmented with type edges and class vertices for every entity variable vertex in the path. Let z be the number of unordered pairs of query keywords which have the same connection in the two pattern graphs. The similarity of the keyword pair connections in P_1 and P_2 is given by:

$$conn\_sim(P_1, P_2) = z / (n(n-1)/2)$$

where n is the number of keywords in Q. The denominator reflects the number of unordered keyword pairs for the keywords in Q. Similarly to $mc\_sim(P_1, P_2)$, $conn\_sim(P_1, P_2)$ ranges between [0,1], with 1 indicating that the matching constructs for all the keywords and all the connections between them are the same.

In the example of Figure 7.3 both pattern graphs P_2 and P_3 have five common matching constructs with P_1. However, $conn\_sim(P_1, P_2) = 6$ and $conn\_sim(P_1, P_3) = 4$. Intuitively, P_2 looks more similar to P_1 than P_3 to P_1.

Measuring the similarity of two pattern graphs P_1 and P_2 based solely on the similarity of matching constructs and matching construct connections, $mc\_sim(P_1, P_2)$ and $conn\_sim(P_1, P_2)$, cannot entirely capture their semantic closeness. Compare, for instance, the pattern graphs P_4 and P_5 with the pattern graph P_1 in Figure 7.2. Both P_4 and P_5 have 4 keyword matching constructs and 6 pairs of matching construct connections in common with P_1. However, our intuition suggests that P_5 is less similar (more dissimilar) to P_1 than P_4 as it has the class vertex (concept) "Journal" which does not appear in P_1. In contrast, P_1 and P_4 have the same class vertices. Therefore, we introduce the metric of concept dissimilarity to capture the dissimilarity of two pattern graphs. Let $c(P)$ denote the set of class vertices in a pattern graph P. Given two pattern graphs P_1 and

P_2 of a keyword query,

$$concept\_dsim(P_1,P_2) = |(c(P_1) \cup c(P_2)) - (c(P_1) \cap c(P_2))| \,/\, |c(P_1) \cup c(P_2)|$$

$concept\_dsim(P_1,P_2)$ ranges between 0 (when P_1 and P_2 have all their class vertices in common) and 1 (when P_1 and P_2 do not have common class vertices). The higher the value of $concept\_dsim(P_1,P_2)$, the more distant the pattern graphs P_1 and P_2 are.

Taking into account all the factors, we define the distance $dist(P_1,P_2)$ of two pattern graphs P_1 and P_2 as follows. Note that $concept\_dsim(P_1,P_2)$ is considered with a negative sign since it expresses dissimilarity.

$$dist(P_1,P_2) = \frac{1 - [(mc\_sim(P_1,P_2) + conn\_sim(P_1,P_2))/2 - concept\_dsim(P_1,P_2)]}{2}$$

$dist(P_1,P_2) = 0$ when the two pattern graphs are the same and $dist(P_1,P_2) = 1$ when the $concept\_dsim(P_1,P_2) = 1$.

We now define the diversity of a set of pattern graphs S of size k as:

$$diversity(S) = \sum_{P_i,P_j \in S, P_i \neq P_j} dist(P_i,P_j)/k(k-1)$$

Dividing the sum by the total number of pattern graph pairs, normalizes $diversity(S)$ in the [0,1] range.

7.3 Algorithm

In this section, we present an algorithm for the problem of balancing the relevance and diversity of sets of pattern graphs stated in Section 7.2.1. Exhaustively generating all size-k subsets of a set of pattern graphs and computing their relevance and diversity in order to find an optimal one has exponential complexity in the number of the pattern graphs. In fact, different versions of the diversification problem have previously been shown to be NP-

Algorithm 4 : PGDiversification (Pattern Graph Diversification)

Input: $Q = \{k_1, \ldots, k_n\}$: a keyword query with n keywords, S: Structural Summary of the data graph, α: tuning factor, k: size of the output list.
Output: $\mathscr{P}_{div}$: set of diversified pattern graphs of size k.

```
1:  for all k_i ∈ Q do
2:      L_i ←{set of all matching constructs of k_i on S};
3:  P ← ComputePatternGraphs({×_{i=1}^{n} L_i}, S);
4:  P ← SortByRelevance(P);
5:  P_div ← P[0];
6:  i = 1;
7:  while i < k do
8:      j = i;
9:      NextIndex = −1;
10:     NextScore = 0;
11:     while j ≤ |P| do
12:         distance = 0;
13:         for all p_i ∈ P_div do
14:             distance = distance + dist(p_i, P[j])
15:         CurrentScore = α ∗ relevance(P[j], Q) + (1 − α) ∗ distance/|P_div|;
16:         if CurrentScore > NextScore then
17:             NextScore = CurrentScore;
18:             NextIndex = j;
19:         j = j + 1;
20:     P_div.add(P[NextIndex]);
21:     swapPGs(P[i], P[NextIndex]);
22:     i = i + 1;
```

hard [2, 18, 29, 35]. Therefore, we design a heuristic algorithm, called *PGDiversification*, which greedily selects a new pattern graphs at every iteration and incrementally computes the relevance and diversity of pattern graph sets. Algorithm *PGDiversification* takes as input a keyword query Q, the structural summary S of an RDF graph, the tuning parameter α, and a positive integer k. The output is a subset of the set of pattern graphs of Q on S of size k.

The algorithm starts by finding all the matching constructs of the keywords in query Q on S (lines 1-2) and then generates the set $\mathscr{P}$ of pattern graphs for all possible signatures of Q (line 3). The pattern graphs are generated as r-radius Steiner graphs using the algorithm in [22]. The pattern graphs of $\mathscr{P}$ are ranked in descending order of their relevance (line 4). The variable $\mathscr{P}_{div}$ represents the output set of size k which is a subset of the set of

pattern graphs $\mathscr{P}$. Initially, the set $\mathscr{P}_{div}$ contains a pattern graph with the highest relevance (line 5). Subsequently, at every iteration, a pattern graph is chosen for inclusion in $\mathscr{P}_{div}$ so that the new $\mathscr{P}_{div}$ set maximizes the objective function (line 8-22). The process terminates when $|\mathscr{P}_{div}| = k$.

7.4 Experimental Results

We implemented our approach and ran experiments to examine: (a) the effectiveness of our distance metric in assessing the semantic similarity of pattern graphs, and the quality of our approach in retrieving relevant results, and (b) the efficiency of our *PGDiversification* algorithm in computing the set of pattern graphs that trades off relevance for diversity.

7.4.1 Datasets and Queries

We used the DBLP and Jamendo[1] real datasets for our experiments. DBLP is a bibliography database of 600MB of size, containing 8.5M triples. Jamendo is a repository of Creative Commons licensed music of 85MB of size, containing 1.1M triples. The extracted structural summaries and the keyword inverted lists for both datasets were stored in a Re-

[1] http://dbtune.org/jamendo/ (accessed on April 24, 2016)

Table 7.1 Keyword Queries on Jamendo and DBLP Datasets

Keyword Queries on Jamendo		Keyword Queries on DBLP	
Q ID	**Keywords**	**Q ID**	**Keywords**
J1	doument, teenage, fantasie	D1	concatenable, aspectisation, oliver
J2	nuts, spy4, lemonade	D2	dataflow, quantization
J3	divergence, obsession, lyrics	D3	donatella, intermittent, congestion
J4	reflection, record	D4	balvinder, coscheduling, article
J5	document, cool, divergence	D5	springer, inproceedings
J6	cicada, performance	D6	skogstad, tensorial, morphology
J7	extraordinary, blissful, madness	D7	hierarchical, hybridization
J8	awesome, passion, spy4	D8	person, tolga, coscheduling
J9	guitarist, lemonade	D9	charles, peephole, inproceedings
J10	disgusting, revenge, fantasie	D10	tolga, forward, normalizability

lational database. The experiments were conducted on a standalone machine with an Intel i7-5600U@2.60GHz processor and 8GB memory. We experimented with a large number of queries and we report on 10 of them for each dataset. Table 7.1 shows the queries used.

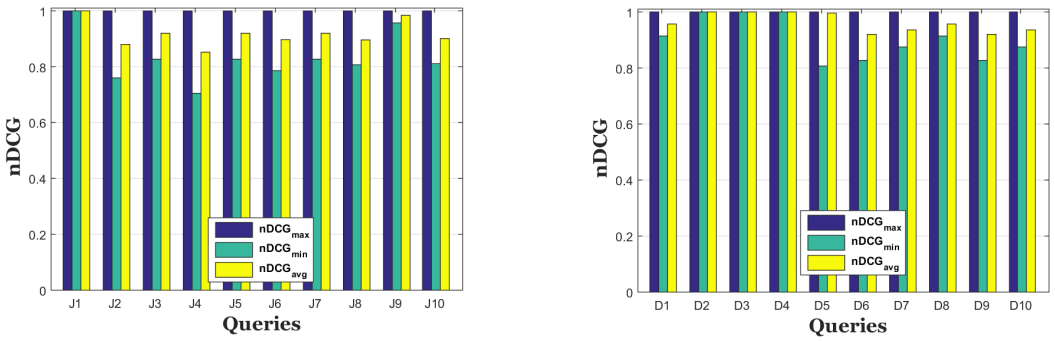
7.4.2 Effectiveness Results

Effectiveness of the Distance Metric. We first want to examine the quality of our distance metric. To this end, for each of the queries in Table 7.1, we select five of their pattern graphs. We ask an expert user to score the semantic similarity of each one of these five pattern graphs with another pattern graph (the pattern graph with the highest relevance). The scores are integers in the range [0, 3]. A score of 0 denotes that the two pattern graphs are totally dissimilar. We also use our distance metric $dist(P_1, P_2)$ to rank the five pattern graphs in descending order of their distance from the most relevant pattern graph. We assess the quality of the ranking based on $dist(P_1, P_2)$ using the *normalized Discounted Cumulative gain* (nDCG) metric which is defined as follows. The *cumulative gain* (CG) for position n in the ranked list is the sum of the scores of the items in the ranked positions 1 to n. A discounting function is used over cumulative gain to measure *discounted cumulative gain* (DCG) for position n. The DCG at n is given by the following formula:

$$DCG_n = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

The DCG value of a ranked list is the DCG value at position n of the list where n is the size of the list. The *normalized discounted cumulative gain* (nDCG) is the result of normalizing DCG with the DCG of the correct list (the one that reflects the scoring of the expert user), that is, by dividing the DCG value of the system's ranked list by the DCG value of the correct ranked list. Thus, nDCG favors a ranked list which is similar to the correct ranked list.

In order to take into account equivalent classes of pattern graphs in the ranked lists

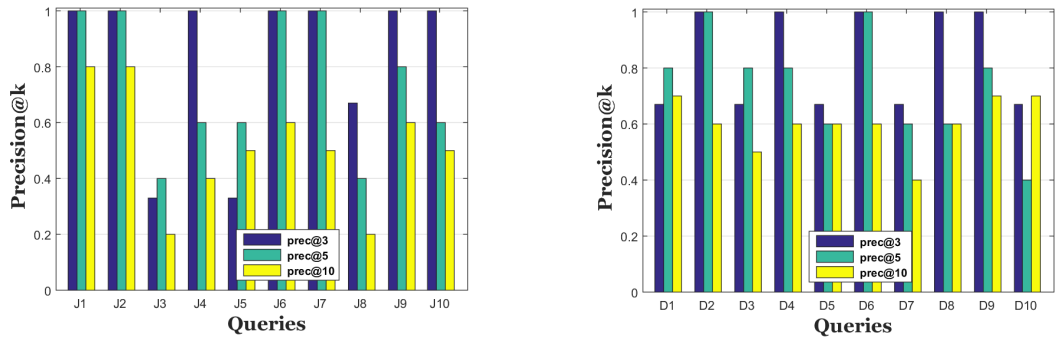


Jamendo dataset DBLP dataset

Figure 7.4 nDCG$_{max}$, nDCG$_{min}$ and nDCG$_{avg}$ for the queries of Table 7.1 on the two datasets.

(that is, pattern graphs which have the same rank), we have extended nDCG by introducing minimum, maximum and average values for it. The nDCG$_{max}$ value of a ranked list RL_e with equivalence classes corresponds to the nDCG value of a strictly ranked (that is, without equivalence classes) list obtained from RL_e by ranking the pattern graphs in the equivalence classes correctly (that is, in compliance with the scores given by the expert user). The nDCG$_{min}$ value of RL_e corresponds to the nDCG value of a strictly ranked list obtained from RL_e by ranking the pattern graphs in the equivalence classes in reverse correct order. The nDCG$_{avg}$ value of RL_e is the average nDCG value over all strictly ranked lists obtained from RL_e by ranking the pattern graphs in the equivalence classes in all possible ways. The nDCG values range between 0 and 1. Figure 7.4 shows the nDCG$_{min}$, nDCG$_{max}$ and nDCG$_{avg}$ values for the queries on DBLP and Jamendo datasets. As one can see, all the values are very close to 1. This implies that our distance metric successfully assesses the semantic similarity of two pattern graphs.

Effectiveness of the Approach. In order to evaluate the quality of the approach in retrieving relevant results, we measure the relevant results retrieved by Algorithm *PGDiversification* for different queries when only our relevance metric, and when our metric which balances relevance and diversity is taken into account. An expert user characterizes the retrieved pattern graphs as relevant or not to the query based on whether the pattern graphs



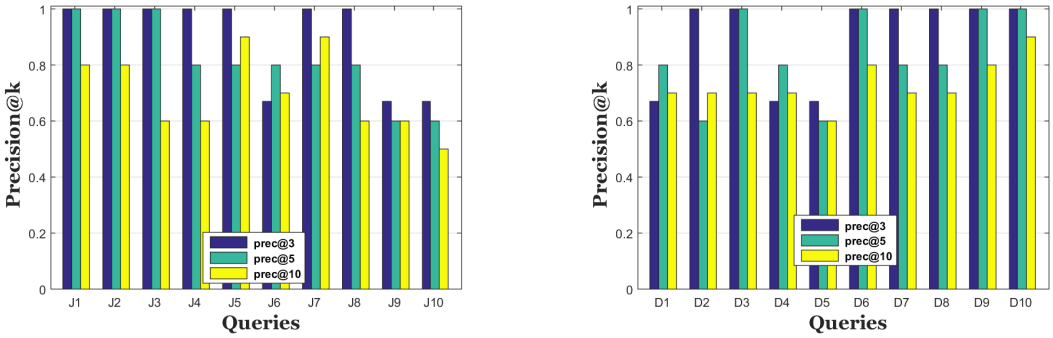
Jamendo dataset DBLP dataset

Figure 7.5 Prec@k for $k = 3$, 5 and 10, for the queries of Table 7.1 on the Jamendo and DBLP datasets based solely on the relevance metric.

express meaningful interpretations of the query. The quality of our approach on a query is expressed by *precision@k* (*prec@k*), which is the ratio of the number of relevant pattern graphs in the set of k pattern graphs returned by our algorithm to k. Figure 7.5 displays *prec@k* for $k = 3$, 5 and 10 for the queries of Table 7.1 on the two datasets when only the relevance metric is taken into account (that is, when the tuning parameter α in the objective function is set to 1). Figure 7.6 displays *prec@k* for $k = 3$, 5 and 10 for the queries of Table 7.1 on the two datasets when both the relevance and diversity metrics are taken into account (that is, when the tuning parameter α in the objective function is set to 0.5). As we can see, for all values of k and for both datasets the *precision@k* is the same or better in most cases. This observation demonstrates the benefit of introducing diversity in the process of selecting the set of k pattern graphs as a larger number of users can be satisfied by the returned pattern graph set.

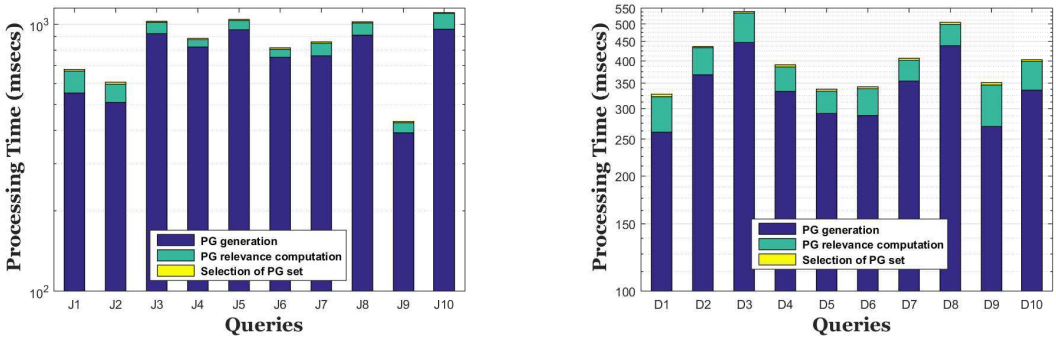
7.4.3 Efficiency Results

We ran Algorithm *PGDiversification* for the queries of Table 7.1 on the two datasets. The execution time is reported in Figure 7.7 for pattern graphs sets of size $k = 5$ and tuning parameter $\alpha = 0.5$. The execution time for each query comprises three components: (a) the generation of possible pattern graphs, (b) the computation of the relevance of the pattern



Jamendo dataset DBLP dataset

Figure 7.6 Prec@k for $k = 3$, 5 and 10, for the queries of Table 7.1 on the Jamendo and DBLP datasets based on the relevance and diversity metrics.



Jamendo dataset DBLP dataset

Figure 7.7 Processing time of *PGDiversification* algorithm for the queries on the Jamendo and DBLP datasets.

graphs and the selection of one with with highest relevance, and (c) the application of the greedy heuristic for generating the list of k pattern graphs which is both relevant and diversified. One can see that the execution time is dominated by the pattern graph generation process. This is expected since computing the pattern graphs requires access to the database for finding all the matching constructs for the query keyword and then generating the pattern graphs using the structural summary of the data graph. In contrast, the other two processes need much less time, especially the pattern graph selection process as it benefits from the greedy heuristic.

7.5 Conclusion

In this chapter, we presented a novel technique for trading off relevance for diversity in the result sets for keyword queries on RDF data graphs. We have formally defined the problem of diversification as an optimization problem. The goal of our problem is to generate a result set that provides a broad overview of the aspects of a keyword query and ensures relevance of the results. We have applied our diversification scheme to pattern graphs which are clusters of result graphs having the same structural and semantic features and represent alternative interpretations of a keyword query. In doing so, we ensure diverse query interpretations in the result set. Furthermore, we introduced metrics for estimating the relevance of pattern graphs to a keyword query and the diversity of pattern graph sets. We designed and implemented a greedy algorithm for incrementally generating a set of pattern graphs which maximizes our objective function on relevance and diversity. We conducted experiments to establish the effectiveness of our proposed metric for measuring the semantic similarity of pattern graphs and the overall quality of our approach. The experiments also showed that our algorithm is efficient in generating a set of pattern graphs which is relevant and diverse.

CHAPTER 8

CONCLUSION AND RESEARCH DIRECTIONS

This dissertation investigates different issues related to the efficient and effective evaluation of keyword queries on RDF data graphs. Keyword search over RDF data departs from traditional keyword search over unstructured (flat), semistructured and structured data due to the semantic nature of RDF graph data. As keyword queries are imprecise and ambiguous, they typically return a huge of candidate results of which very few are relevant to the user intent. This characteristic of keyword search entails three fundamental problems of keyword search over RDF data graphs: (a) identify an appropriate form for the query results, (b) selecting the relevant results among a plethora of candidates, and (c) designing query evaluation techniques which can scale when the size of the data graph and/or the number of keyword queries increases. We address these problems following an approach which exploits a structural summary of the graph data and additional information from the user.

Our first contribution defined query results as meaningful subgraphs of the RDF data graph that appropriately connect together elementary subgraphs representing semantic interpretations of the keyword instances. We presented an alternative solution to keyword search over RDF graphs which hierarchically clusters the results based on a semantic interpretation of the keyword instances and takes advantage of relevance feedback from the user. Our clustering hierarchy exploits pattern graphs which are structured queries clustering together result graphs with the same structural and semantic characteristics and represent possible interpretations for the keyword query. We designed an algorithm which computes r-radius Steiner patterns graphs using exclusively the structural summary of the data graph. The user selects relevant pattern graphs by exploring only a small portion of the hierarchy supported by a ranking of the hierarchy components. Our experimental results showed the feasibility of our system by demonstrating short reach times and efficient computation of the relevant results.

Although structural summary-based approaches are promising, they suffer from a drawback: as structural summaries are approximate representations of the data the computed pattern graphs might return empty answers or miss results which are relevant to the user intent. To address this problem, we introduced a novel relaxation technique on pattern graphs that can extract more results potentially of interest to the user. We leveraged pattern graph homomorphisms to define relaxed pattern graphs. We introduced an operation on pattern graphs which was proved to be complete, that is, it can produce all relaxed pattern graphs. The goal of our relaxation process is to guarantee that the produced relaxed pattern graphs are as close to the initial pattern graph as possible. We devised different metrics in descending order of importance to estimate the degree of relaxation of a relaxed pattern graph. We designed an algorithm to compute relaxed pattern graphs with non-empty answers in relaxation order. We adapted subquery caching and multiquery optimization techniques to the context of relaxed pattern graphs in order to improve the successive computation and evaluation of relaxed pattern graphs. Lastly, we ran experiments on different real datasets to assess the effectiveness of our ranking of relaxed pattern graphs, and the efficiency of our algorithm and optimization techniques in computing relaxed pattern graphs and their answers.

The success of the previous two approaches largely relies on the users' relevance feedback. However, a user might not always be able or willing to provide feedback. In the absence of users' interaction with the system, identifying relevant results in very challenging. To address this problem, we introduced cohesive keyword queries on RDF data. Using cohesive queries a user can flexibly and effortlessly convey her information need by specifying cohesive keyword groups in it. A cohesive group of keywords in a query indicates that the keywords of the group should form a cohesive unit in the query results. We provided formal semantics of cohesive queries. We designed a query evaluation algorithm to generate pattern graphs satisfying cohesiveness constraints over the structural summary of an RDF graph. The pattern graphs can be expressed as structured queries which are then

evaluated over the RDF data to compute the query results. Experimental results demonstrated that our algorithm is efficient in computing all the pattern graphs that satisfy the cohesiveness constraints. Additionally, our experiments showed the effectiveness of cohesive keyword queries in improving the result quality and in pruning the search space of pattern graphs compared to flat keyword queries. It is to be noted, that these benefits are achieved while retaining the simplicity and convenience of traditional keyword search.

The techniques discussed above deal with the problems of keyword search on RDF data by solely taking into account the criteria of relevance of the retrieved pattern graphs to the user intent. Although these approaches are successful in retrieving high quality relevant results, they might still fail to capture some user's intent as they do not consider the diversity of the returned pattern graph set. Therefore, we addressed the problem of diversifying keyword search results over RDF data, by devising a novel technique that returns to the user a relevant and diverse set of pattern graphs. We modeled the problem as an optimization problem that selects a set of k pattern graphs which maximizes an objective function on relevance and diversity. We proposed metrics to estimate the relevance and diversity of a set of pattern graphs. We designed a greedy heuristic algorithm for trading relevance and diversity in a computed set of k pattern graphs for a given keyword query. Finally, we ran experiments that demonstrate the effectiveness of our relevance and diversity metrics and the efficiency of our algorithm.

A number of research directions might deserve further attention. In relation to the pattern graph relaxation problem, it can be observed that by combining multiple relaxed pattern graphs it might be possible to identify results which are relevant and cannot be computed by any one of these pattern graphs separately. It would then be interesting to design algorithms which identify which relaxed pattern graphs to examine and how to combine them in order to discover such relevant results. In relation to the cohesive query language, it is important to further study and refine the semantics of cohesive keyword queries on RDF data in order to reduce more the search space of pattern graphs considered and return

results of higher quality. Finally, in relation to the pattern graph set diversification problem, the design of metrics for assessing diversity customized to the diversification problem on RDF graphs would benefit the selection of diverse sets of pattern graphs for keyword queries.

REFERENCES

[1] B. Aditya, S. Chakrabarti, R. Desai, A. Hulgeri, H. Karambelkar, R. Nasre, and S. Sudarshan. User interaction in the banks system: a demonstration. In *ICDE, Proceedings of International Conference on Data Engineering,* Bangalore, India, pages 786–788. IEEE, 2003.

[2] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM, Proceedings of the Second International Conference on Web Search and Web Data Mining,* Barcelona, Spain, pages 5–14. ACM, 2009.

[3] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE, Proceedings of the 18th International Conference on Data Engineering,* San Jose, CA, USA, pages 5–16. IEEE, 2002.

[4] A. Agresti. *Analysis of ordinal categorical data*, volume 656. John Wiley & Sons, Hoboken, NJ, 2010.

[5] C. Aksoy, A. Dimitriou, D. Theodoratos, and X. Wu. *XReason*: A semantic approach that reasons with patterns to answer XML keyword queries. In *DASFAA, Proceedings of the 18th International Conference on Database Systems for Advanced Applications,* Wuhan, China, pages 299–314. Springer, 2013.

[6] C. Aksoy, A. Dass, D. Theodoratos, and X. Wu. Clustering query results to support keyword search on tree data. In *WAIM, Proceedings of the 15th International Conference on Web-Age Information Management,* Macau, China, pages 213–224. Springer, 2014.

[7] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT, Proceedings of the 8th International Conference on Extending Database Technology,* Prague, Czech Republic, pages 496–513. OpenProceedings.org, 2002.

[8] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[9] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB, Proceedings of the 30th International Conference on Very Large Data Bases,* Toronto, Canada, pages 564–575. Springer, 2004.

[10] T. Berners-Lee, M. Fischetti, and M. L. Foreword By-Dertouzos. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperBusiness, New York City, NY, 2000.

[11] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):28–37, 2001.

[12] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE, Proceedings of the 18th International Conference on Data Engineering,* San Jose, CA, USA, pages 431–440. IEEE, 2002.

[13] N. Bikakis, G. Giannopoulos, J. Liagouris, D. Skoutas, T. Dalamagas, and T. Sellis. Rdivf: Diversifying keyword search on rdf graphs. In *TPDL, Proceedings of International Conference on Theory and Practice of Digital Libraries,* Valletta, Malta, pages 413–416. Springer, 2013.

[14] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[15] T. Brodianskiy and S. Cohen. Self-correcting queries for xml. In *CIKM, Proceedings of the 16th International Conference on Information and Knowledge Management,* Lisbon, Portugal, pages 11–20. ACM, 2007.

[16] J. Broekstra and A. Kampman. Serql: An RDF query and transformation language. In *ISWC, Proceedings of International Semantic Web Conference*, volume 2004. Springer, 2004.

[17] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR, Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval,* Melbourne, Australia, pages 335–336. ACM, 1998.

[18] B. Carterette. An analysis of NP-completeness in novelty and diversity ranking. *Information Retrieval*, 14(1):89–106, 2011.

[19] H. Chen and D. R. Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *SIGIR, Proceedings of the 29th Annual International Conference on Research and Development in Information Retrieval,* Seattle, Washington, USA, pages 429–436. ACM, 2006.

[20] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR, Proceedings of the 31st Annual International Conference on Research and Development in Information Retrieval,* Singapore, pages 659–666. ACM, 2008.

[21] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *Proceedings of the VLDB Endowment*, 1(1):1189–1204, 2008.

[22] A. Dass, C. Aksoy, A. Dimitriou, and D. Theodoratos. Exploiting semantic result clustering to support keyword search on linked data. In *WISE, Proceedings of 15th International Conference on Web Information Systems Engineering,* Thessaloniki, Greece, pages 448–463. Springer, 2014.

[23] A. Dass, C. Aksoy, A. Dimitriou, and D. Theodoratos. Keyword pattern graph relaxation for selective result space expansion on linked data. In *ICWE, Proceedings of the 15th International Conference on Web Engineering,* Rotterdam, The Netherlands, pages 287–306. Springer, 2015.

[24] A. Dass, A. Dimitriou, C. Aksoy, and D. Theodoratos. Incorporating cohesiveness into keyword search on linked data. In *WISE, Proceedings of the 16th International Conference on Web Information Systems Engineering,* Miami, Florida, USA, pages 47–62. Springer, 2015.

[25] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. *DivQ*: diversification for keyword search over structured databases. In *SIGIR, Proceedings of the 33rd International Conference on Research and Development in Information Retrieval,* Geneva, Switzerland, pages 331–338. ACM, 2010.

[26] A. Dimitriou and D. Theodoratos. Efficient keyword search on large tree structured datasets. In *KEYS, Proceedings of the Third International Workshop on Keyword Search on Structured Data,* Scottsdale, AZ, USA, pages 63–74. ACM, 2012.

[27] A. Dimitriou, A. Dass, D. Theodoratos, and Y. Vassiliou. Cohesive keyword search on tree data. In *EDBT, Proceedings of the 19th International Conference on Extending Database Technology,* Bordeaux, France, pages 137–148. OpenProceedings.org, 2016.

[28] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE, Proceedings of the 23rd International Conference on Data Engineering, 2007,* Istanbul, Turkeyz, pages 836–845. IEEE, 2007.

[29] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1): 41–47, 2010.

[30] S. Elbassuoni and R. Blanco. Keyword search over RDF graphs. In *CIKM, Proceedings of the 20th ACM Conference on Information and Knowledge Management,* Glasgow, United Kingdom, pages 237–242. ACM, 2011.

[31] S. Elbassuoni, M. Ramanath, R. Schenkel, and G. Weikum. Searching RDF graphs with SPARQL and keywords. *Data Engineering Bull.*, 33(1):16–24, 2010.

[32] H. Fu, S. Gao, and K. Anyanwu. Disambiguating keyword queries on RDF databases using "Deep" segmentation. In *ICSC, Proceedings of the 4th International Conference on Semantic Computing,* Pittsburgh, PA, USA, pages 236–243. IEEE, 2010.

[33] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB, Proceedings of the 23rd International Conference on Very Large Data Bases,* Athens, Greece, pages 436–445. Springer, 1997.

[34] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Vancouver, BC, Canada, pages 927–940. ACM, 2008.

[35] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW, Proceedings of the 18th International Conference on World Wide Web,* Madrid, Spain, pages 381–390. ACM, 2009.

[36] R. V. Guha and D. Brickley. Rdf vocabulary description language 1.0: Rdf schema. *World Wide Web Consortium recommendation*, 2004.

[37] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *SIGMOD, Proceedings of the International Conference on Management of Data,* San Diego, California, USA, pages 16–27. ACM, 2003.

[38] M. Hasan, A. Mueen, V. J. Tsotras, and E. J. Keogh. Diversifying query results on semi-structured data. In *CIKM, Proceedings of the 21st International Conference on Information and Knowledge Management,* Maui, Hawai, USA, pages 2099–2103. ACM, 2012.

[39] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Beijing, China, pages 305–316. ACM, 2007.

[40] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011.

[41] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. Owl 2 web ontology language primer. *World Wide Web Consortium recommendation*, 27(1): 123, 2009.

[42] V. Hristidis and Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *VLDB, Proceedings of 28th International Conference on Very Large Data Bases,* Hong Kong, China, pages 670–681. Springer, 2002.

[43] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB, Proceedings of the 29th International Conference on Very Large Database,* Berlin, Germany, pages 850–861. Springer, 2003.

[44] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. *Trans. Knowl. Data Eng.*, 18(4):525–539, 2006.

[45] A. Hulgeri, G. Bhalotia, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword search in databases. *Data Engineering Bulletin*, 24(3):22–31, 2001.

[46] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *Trans. Inf. Syst.*, 20(4):422–446, 2002.

[47] M. Jiang, Y. Chen, J. Chen, and X. Du. Interactive predicate suggestion for keyword search on RDF graphs. In *ADMA, Proceedings of the 7th International Conference on Advanced Data Mining and Applications,* Beijing, China, pages 96–109. Springer, 2011.

[48] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB, Proceedings of the 31st International Conference on Very Large Data Bases,* Trondheim, Norway, pages 505–516. Springer, 2005.

[49] Z. Kaoudi. *Distributed RDF query processing and reasoning in peer-to-peer networks*. PhD thesis, National and Kapodistrian University of Athens, Greece, 2011.

[50] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *Proceedings of the VLDB Endowment*, 4(10):681–692, 2011.

[51] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: a declarative query language for RDF. In *WWW, Proceedings of the Eleventh International World Wide Web Conference,* Honolulu, Hawaii, pages 592–603. ACM, 2002.

[52] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering indexes for branching path queries. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Madison, Wisconsin, USA, pages 133–144. ACM, 2002.

[53] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[54] L. Kong, R. Gilleron, and A. Lemay. Retrieving meaningful relaxed tightest fragments for XML keyword search. In *EDBT, Proceedings of the 12th International Conference on Extending Database Technology,* Saint Petersburg, Russia, pages 815–826. OpenProceedings.org, 2009.

[55] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW, Proceedings of the 13th international conference on World Wide Web, 2004,* New York, NY, USA, pages 658–665. ACM, 2004.

[56] G. Ladwig and T. Tran. Combining query translation with query answering for efficient keyword search. In *ESWC, Proceedings of the 7th Extended Semantic Web Conference,* Heraklion, Crete, Greece, pages 288–303. Springer, 2010.

[57] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification, 1999.

[58] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *WWW, Proceedings of the 20th International Conference on World Wide Web,* Hyderabad, India, pages 655–664. ACM, 2011.

[59] W. Le, A. Kementsietsidis, S. Duan, and F. Li. Scalable multi-query optimization for SPARQL. In *ICDE, Proceedings of the 28th International Conference on Data Engineering,* Washington, DC, USA, pages 666–677. IEEE, 2012.

[60] W. Le, F. Li, A. Kementsietsidis, and S. Duan. Scalable keyword search on large RDF data. *Trans. Knowl. Data Eng.*, 26(11):2774–2788, 2014.

[61] Y. Lei, V. S. Uren, and E. Motta. Semsearch: A search engine for the semantic web. In *EKAW, Proceedings of the 15th International Conference on Managing Knowledge in a World of Networks,* Podebrady, Czech Republic, pages 238–245. Springer, 2006.

[62] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *SIGACT-SIGMOD-SIGART, Proceedings of the Fourteenth Symposium on Principles of Database Systems,* San Jose, California, USA, pages 95–104. ACM, 1995.

[63] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Vancouver, BC, Canada, pages 903–914. ACM, 2008.

[64] J. Li, C. Liu, and J. X. Yu. Context-based diversification for keyword queries over XML data. *Knowledge and Data Engineering*, 27(3):660–672, 2015.

[65] X. Li, C. Li, and C. Yu. Entity-relationship queries over wikipedia. *Transactions on Intelligent Systems and Technology*, 3(4):70, 2012.

[66] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Chicago, Illinois, USA, pages 563–574. ACM, 2006.

[67] X. Liu, C. Wan, and L. Chen. Returning clustered results for keyword search on XML documents. *Trans. Knowl. Data Eng.*, 23(12):1811–1825, 2011.

[68] Z. Liu and Y. Chen. Processing keyword search on XML: a survey. *World Wide Web*, 14(5-6):671–707, 2011.

[69] Z. Liu and Y. Cher. Reasoning and identifying relevant matches for XML keyword search. *Proceedings of the VLDB Endowment*, 1(1):921–932, 2008.

[70] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *Proceedings of the VLDB Endowment*, 2(1):313–324, 2009.

[71] Z. Liu, S. Natarajan, and Y. Chen. Query expansion based on clustered results. *Proceedings of the VLDB Endowment*, 4(6):350–361, 2011.

[72] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Beijing, China, pages 115–126. ACM, 2007.

[73] A. Markowetz, Y. Yang, and D. Papadias. Keyword search on relational data streams. In *SIGMOD, Proceedings of the International Conference on Management of Data, Beijing, China*, pages 605–616. ACM, 2007.

[74] C. S. Nikolaou. *Keyword Search in RDF Databases*. PhD thesis, National and Kapodistrian University of Athens, Greece, 2010.

[75] J. Pound, I. F. Ilyas, and G. E. Weddell. Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Indianapolis, Indiana, USA, pages 423–434. ACM, 2010.

[76] E. Prud Hommeaux and A. Seaborne. SPARQL query language for RDF. *World Wide Web Consortium recommendation*, 15, 2008.

[77] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of RDBMS. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Providence, Rhode Island, USA, pages 681–694. ACM, 2009.

[78] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE, Proceedings of the 25th International Conference on Data Engineering,* Shanghai, China, pages 724–735. IEEE, 2009.

[79] L. Qin, J. X. Yu, and L. Chang. Scalable keyword search on large data streams. *The VLDB Journal*, 20(1):35–57, 2011.

[80] F. Radlinski and S. T. Dumais. Improving personalized web search using result diversification. In *SIGIR, Proceedings of the 29th Annual International Conference on Research and Development in Information Retrieval,* Seattle, Washington, USA, pages 691–692. ACM, 2006.

[81] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD, Proceedings of the International Conference on Management of Data,* Dallas, Texas, USA, pages 249–260. ACM, 2000.

[82] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[83] A. Seaborne. Rdql-a query language for RDF. *World Wide Web Consortium Member submission*, 9(29-21):33, 2004.

[84] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *Transactions on Knowledge & Data Engineering*, 2(2):262–266, 1990.

[85] T. K. Sellis. Multiple-query optimization. *Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[86] A. Simitsis, G. Koutrika, and Y. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal*, 17(1):117–149, 2008.

[87] M. Sintek and S. Decker. TRIPLE - A query, inference, and transformation language for the semantic web. In *ISWC, Proceedings of the 1st International Semantic Web Conference,* Sardinia, Italy, pages 364–378. Springer, 2002.

[88] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based interpretation of keywords for semantic search. In *ISWC, Proceedings of the 6th International Semantic Web Conference Semantic Web,* Busan, Korea, pages 523–536. Springer, 2007.

[89] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE, Proceedings of the 25th International Conference on Data Engineering,* Shanghai, China, pages 405–416. IEEE, 2009.

[90] H. Wang, K. Zhang, Q. Liu, T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. In *ESWC. Proceedings of 5th European Semantic Web Conference,* Tenerife, Canary Islands, Spain, pages 584–598. Springer, 2008.

[91] H. Wang, Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, and Y. Pan. Semplore: A scalable IR approach to search the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):177–188, 2009.

[92] K. Xu, J. Chen, H. Wang, and Y. Yu. Hybrid graph based keyword query interpretation on RDF. In *ISWC, Proceedings of International Semantic Web Conference, Posters & Demonstrations Track,* Shanghai, China. Springer, 2010.

[93] Y. Xu and Y. Papakonstantinou. Efficient LCA based keyword search in XML data. In *EDBT, Proceedings of the 11th International Conference on Extending Database Technology,* Nantes, France, pages 535–546. OpenProceedings.org, 2008.

[94] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *ICDE, Proceedings of the 25th International Conference on Data Engineering, 2009,* Shanghai, China, pages 1299–1302. IEEE, 2009.

[95] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys, Proceedings of the International Conference on Recommender Systems,* Lausanne, Switzerland, pages 123–130. ACM, 2008.

[96] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW, Proceedings of the 14th international conference on World Wide Web,* Chiba, Japan, pages 22–32. ACM, 2005.