

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

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

SEMANTICS AND RESULT DISAMBIGUATION FOR KEYWORD SEARCH ON TREE DATA

**by
Cem Aksoy**

Keyword search is a popular technique for searching tree-structured data (e.g., XML, JSON) on the web because it frees the user from learning a complex query language and the structure of the data sources. However, the convenience of keyword search comes with drawbacks. The imprecision of the keyword queries usually results in a very large number of results of which only very few are relevant to the query. Multiple previous approaches have tried to address this problem. Some of them exploit structural and semantic properties of the tree data in order to filter out irrelevant results while others use a scoring function to rank the candidate results. These are not easy tasks though and in both cases, relevant results might be missed and the users might spend a significant amount of time searching for their intended result in a plethora of candidates. Another drawback of keyword search on tree data, also due to the incapacity of keyword queries to precisely express the user intent, is that the query answer may contain different types of meaningful results even though the user is interested in only some of them.

Both problems of keyword search on tree data are addressed in this dissertation. First, an original approach for answering keyword queries is proposed. This approach extracts structural patterns of the query matches and reasons with them in order to return meaningful results ranked with respect to their relevance to the query. The proposed semantics performs comparisons between patterns of results by using different types of homomorphisms between the patterns. These comparisons are used to organize the patterns into a graph of patterns which is leveraged to determine ranking and filtering semantics. The experimental results show that the approach produces query results of higher quality compared to the previous ones. To address the second problem, an original approach for

clustering the keyword search results on tree data is introduced. The clustered output allows the user to focus on a subset of the results, and to save time and effort while looking for the relevant results. The approach performs clustering at different levels of granularity to group similar results together effectively. The similarity of the results and result clusters is decided using relations on structural patterns of the results defined based on homomorphisms between path patterns. An originality of the clustering approach is that the clusters are ranked at different levels of granularity to quickly guide the user to the relevant result patterns. An efficient stack-based algorithm is presented for generating result patterns and constructing the clustering hierarchy. The extensive experimentation with multiple real datasets show that the algorithm is fast and scalable. It also shows that the clustering methodology allows the users to effectively retrieve their intended results, and outperforms a recent state-of-the-art clustering approach. In order to tackle the second problem from a different aspect, diversifying the results of keyword search is addressed. Diversification aims to provide the users with a ranked list of results which balances the relevance and redundancy of the results. Measures for quantifying the relevance and dissimilarity of result patterns are presented and a heuristic for generating a diverse set of results using these metrics is introduced.

**SEMANTICS AND RESULT DISAMBIGUATION
FOR KEYWORD SEARCH ON TREE DATA**

by
Cem Aksoy

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

Department of Computer Science

January 2016

Copyright © 2016 by Cem Aksoy
ALL RIGHTS RESERVED

APPROVAL PAGE

SEMANTICS AND RESULT DISAMBIGUATION FOR KEYWORD SEARCH ON TREE DATA

Cem Aksoy

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

Dr. Yi Chen, Committee Member Associate Professor, School of Management, NJIT	Date
--	------

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

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

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

BIOGRAPHICAL SKETCH

Author: Cem Aksoy
Degree: Doctor of Philosophy
Date: January 2016

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
New Jersey Institute of Technology, Newark, New Jersey, 2016
- Master of Science in Computer Engineering,
Bilkent University, Ankara, Turkey, 2010
- Bachelor of Science in Computer Engineering,
Bilkent University, Ankara, Turkey, 2008

Major: Computer Science

Presentations and Publications:

- Aksoy, C.,** Dimitriou, A., Dass, A., & Theodoratos, D., (2015) Clustering Query Result Patterns for Effective Keyword Search on Tree Data, Submitted to *Transactions on Knowledge and Data Engineering*, Under 2nd Review.
- Dass, A., Dimitriou, A., **Aksoy, C.,** & Theodoratos, D., (2015) Incorporating Cohesiveness into Keyword Search on Linked Data, in *The 16th International Conference on Web Information System Engineering (WISE)*, pp. 47-62.
- Aksoy, C.,** Dimitriou, A., & Theodoratos, D., (2015) Reasoning with Patterns to Effectively Answer XML Keyword Queries, *The VLDB Journal*, 24(3): 441-465.
- Dass, A., **Aksoy, C.,** Dimitriou, A., & Theodoratos, D., (2015) Keyword Pattern Graph Relaxation for Selective Result Space Expansion on Linked Data, in *The 15th International Conference on Web Engineering (ICWE)*, pp. 287-306.
- Dass, A., **Aksoy, C.,** Dimitriou, A., & Theodoratos, D., (2014) Exploiting Semantic Result Clustering to Support Keyword Search on Linked Data, in *The 15th International Conference on Web Information System Engineering (WISE)*, pp. 448-463.
- Aksoy, C.,** Dass, A., Theodoratos, D., & Wu, X., (2014) Clustering Query Results to Support Keyword Search on Tree Data, in *The 16th International Conference on Web-Age Information Management (WAIM)*, pp. 213-224.

- Aksoy, C.,** Dimitriou, A., Theodoratos, D., & Wu, X., (2013) XReason: A Semantic Approach That Reasons with Patterns to Answer XML Keyword Queries, in *The 18th International Conference on Database Systems for Advanced Applications (DASFAA)*, pp. 299-314.
- Aksoy, C.,** Can, F., & Kocberber, S., (2012) Novelty Detection for Topic Tracking, *Journal of the Association for Information Science and Technology*, 63(4): 777-795.
- Aksoy, C.,** Bugdayci, A., Gur, T., & Uysal, I., Can, F., (2009) Semantic Argument Frequency-based Multi-document Summarization, in *The 24th International Symposium on Computer and Information Sciences (ISCIS)*, pp. 460-464.

*To my late grandfather, Mustafa Kayacan,
To my parents, Hatice and Hasan Aksoy,
To my wife, Özlem Aksoy.*

ACKNOWLEDGMENT

I would like to express my gratitude to my dissertation advisor Dr. Dimitri Theodoratos for his guidance during my Ph.D. studies. I especially appreciate his efforts to make sure that my studies were always on track and to make me improve myself.

I also would like to thank Dr. Yi Chen, Dr. James Geller, Dr. Michael Halper and Dr. Vincent Oria for being a part of my dissertation committee. I am very grateful for their time and valuable comments on my work.

I would like to acknowledge Aggeliki Dimitriou for her contributions to my studies. It has been a pleasure working with her. I also would like to thank my colleagues Ananya Dass and Xiaoying Wu for their help.

I am also thankful to the Department of Computer Science at New Jersey Institute of Technology for their financial support during the course of my Ph.D.

I thank my office mates Ananya Dass, Xiguo Ma, Sheetal Rajgure, Jichao Sun, Arwa Wali and Xiangqian Yu for their companionship.

I would like to thank my family for supporting me with all my decisions and making many sacrifices for my education. I am especially grateful to my wife, Özlem, for her patience and support during my Ph.D. study. Her love has always been an anchor for my soul.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
1.1 Focus of the Investigation	2
1.1.1 Keyword Search Semantics	3
1.1.2 Disambiguation of Results	7
1.2 Organization	12
2 STATE OF THE ART	13
2.1 Searching on XML	13
2.1.1 Structured Query Languages	13
2.1.2 Keyword Search	14
2.2 Search Result Clustering	18
2.3 Search Result Diversification	19
3 TREE DATA MODEL	21
3.1 Data Model	21
3.2 Keyword Queries	22
4 SEMANTICS OF KEYWORD QUERIES OVER TREE DATA	25
4.1 Instance Tree Patterns	25
4.2 Pattern Homomorphism and Homomorphism Relation	26
4.3 Path Homomorphism and Path Homomorphism Relations	28
4.4 <i>XReason</i> Semantics	34
4.5 Analysis of <i>XReason</i>	37
4.5.1 Containment Relationships between Keyword Search Semantics . . .	38
4.5.2 Comparison of Filtering Semantics	41

TABLE OF CONTENTS (Continued)

Chapter	Page
4.6 Algorithms	45
4.6.1 Pattern Generation	46
4.6.2 Graph Construction and Ranking	51
4.6.3 An Extension of PatternStack	56
4.7 Experimental Evaluation	58
4.7.1 Metrics	59
4.7.2 Effectiveness of Filtering Semantics	60
4.7.3 Effectiveness of Ranking Semantics	64
4.7.4 Efficiency	66
4.8 Conclusion	69
5 SEARCH RESULT CLUSTERING	72
5.1 Clustering Methodology	72
5.1.1 First (Bottom) Level of Clustering: Patterns	73
5.1.2 Second Level of Clustering: Classes	74
5.1.3 Third Level of Clustering: Collections	75
5.2 Cluster Ranking and Navigation among Clusters	79
5.2.1 Ranking Patterns	80
5.2.2 Ranking Classes	81
5.2.3 Ranking Collections	81
5.2.4 Cluster Navigation.	82
5.3 The Algorithm	84
5.3.1 Running Example of ClusterStack	91
5.4 Experimental Evaluation	95
5.4.1 Metrics	96

TABLE OF CONTENTS (Continued)

Chapter	Page
5.4.2 Effectiveness Experiments	98
5.4.3 Efficiency Experiments	102
5.5 Conclusion	105
6 DIVERSIFICATION	107
6.1 Formal Problem Definition	107
6.2 Relevance of Patterns	109
6.3 Similarity of Patterns	111
6.4 Algorithm	114
6.5 Experimental Evaluation	116
6.5.1 Evaluation Metrics	116
6.5.2 Experiments	117
6.6 Conclusion	117
7 CONCLUSION AND FUTURE WORK	118
REFERENCES	120

LIST OF TABLES

Table	Page
4.1 Formal Definitions of Different Previous Filtering Semantics in Terms of ITs . . .	39
4.2 Mondial, SIGMOD, DBLP and NASA Dataset Statistics	59
4.3 Definitions of XReal Semantics in Terms of ITs	61
4.4 Queries Used in the Experiments on the Mondial Dataset	62
4.5 Queries Used in the Experiments on the SIGMOD Dataset	63
4.6 Average Precision and Recall Scores for the Queries of Table 4.4 and 4.5 . . .	64
4.7 Best and Worst MAP Scores for the Queries of Tables 4.4 and 4.5	64
4.8 Average $P@10_{exp}$ Scores for the Queries of Tables 4.4 and 4.5	65
4.9 Queries Used in Scalability Experiments	67
4.10 Queries on the DBLP Datasets Used to Compare the Performance of the Original vs. the Extended Algorithm	70
5.1 Inverted Lists of Keywords in Query $\{Advanced, Database, Systems\}$ on the Data Tree T of Figure 1.3	91
5.2 Patterns Under the Node <code>courses</code> of the Data Tree T	93
5.3 Pattern Classes of Q on T	94
5.4 Descendant Path Homomorphisms	94
5.5 ComputeCollections Running Example	95
5.6 Mondial, SIGMOD, DBLP and NASA Dataset Statistics.	96
5.7 Average Reach Time Values (for Retrieving All Relevant Patterns) and Hierarchy Sizes for the Queries of Tables 4.4 and 4.5	99

LIST OF FIGURES

Figure	Page
1.1 A data tree T	2
1.2 Three patterns of the query <i>Physics, James, Harrison</i> on the data tree of Figure 1.1.	5
1.3 A data tree T	8
3.1 An XML tree T	22
3.2 (a) An IT and (b) its MCT.	23
4.1 Some patterns for $Q = \{Physics, James, Harrison\}$ on the tree of Figure 3.1. . .	25
4.2 Pattern MCTs M, M' and M'' and homomorphisms between them.	26
4.3 Pattern MCTs M_2 and M_1 and three path homomorphisms from the paths of M_2 to paths of M_1	29
4.4 Pattern MCTs M_a and M_b and three path homomorphism from the paths of M_a to paths of M_b	30
4.5 Two patterns P and P' that are related with \prec_{aph}	32
4.6 A path homomorphism from a path of pattern P_4 to a path of P_5	34
4.7 (a) The graph G_{\prec} , (b) Pattern order \mathcal{O}	35
4.8 Containment relationships between different filtering semantics.	38
4.9 An XML tree.	42
4.10 PatternStack pattern encoding and combination for $Q = \{Physics, James, Harrison\}$ on the data tree of Figure 4.9.	47
4.11 Precision scores for the queries of Table 4.4 on the Mondial dataset.	62
4.12 Precision scores for the queries of Table 4.5 on the SIGMOD dataset.	63
4.13 Recall scores for the queries of Table 4.4 on the Mondial dataset.	64
4.14 Best and worst P@10 scores for the queries of Table 4.4 on Mondial dataset. .	65

LIST OF FIGURES (Continued)

Figure	Page
4.15 Best and worst P@10 scores for the queries of Table 4.5 on SIGMOD dataset. .	65
4.16 <i>XReason</i> execution times (in secs) for the queries of Tables 4.4 and 4.5.	66
4.17 Computation time vs. output size for the original algorithm using queries with 4, 5 and 6 keywords.	68
4.18 Computation time vs. input size for the extended algorithm using queries with 4, 5 and 6 keywords.	69
4.19 (a) number of ITs, (b) number of generated patterns, and (c) the computation time of the original and the extended algorithm on the DBLP dataset using the queries of Table 4.10.	70
5.1 Some patterns for $Q = \{Advanced, Database, Systems\}$ on the tree of Figure 1.3.	73
5.2 Path correspondances between two patterns, P_4 and P_5	75
5.3 A class of patterns consisting of four patterns.	75
5.4 Descendant path homomorphism from a path in P_4 to a path in P_8	76
5.5 A collection of classes.	79
5.6 Two sample collections	80
5.7 One path isomorphism between two patterns P_3 and P_4	82
5.8 Graph of collections for our running example.	83
5.9 Stack states	92
5.10 Reach time values (for retrieving all relevant patterns) for the queries of Table 4.4 on the Mondial dataset.	99
5.11 Reach time values (for retrieving all relevant patterns) for the queries of Table 4.5 on the SIGMOD dataset.	99
5.12 Average min, exp and max reach time values (for retrieving at most k patterns) for <i>RTCluster</i> with and without ranking of the clusters for the queries of Tables 4.4 and 4.5.	100

LIST OF FIGURES (Continued)

Figure	Page
5.13 Average min, exp and max reach time values (for retrieving at most k patterns) for <i>RTCluster</i> and <i>XMean</i> for the queries of Tables 4.4 and 4.5.	101
5.14 Hierarchy sizes constructed by <i>RTCluster</i> and <i>XMean</i> for the queries of Table 4.4 on the Mondial dataset.	102
5.15 Hierarchy sizes constructed by <i>RTCluster</i> and <i>XMean</i> for the queries of Table 4.5 on the SIGMOD dataset.	102
5.16 Computation time (in msec) for the queries of Table 4.4 on the Mondial dataset.	103
5.17 Computation time (in msec) for the queries of Table 4.5 on the SIGMOD dataset.	103
5.18 Average computation time vs. number of keywords of <i>ClusterStack</i> using queries with 2 to 7 keywords on the DBLP and NASA datasets.	104
5.19 Computation time vs. input size for <i>ClusterStack</i> using queries with 5,6 and 7 keywords.	105
6.1 A data tree which represents a university database consisting of courses and seminars	110
6.2 Four patterns for $Q = \{Quantum, Physics, Miller\}$ on the tree of Figure 6.1 . .	111
6.3 Three patterns which match the keyword query $\{sarah, miller\}$	112
6.4 Two paths p and p' shown with bold edges and possible mappings between their edges depicted with different dashed arrows	113

CHAPTER 1

INTRODUCTION

The amount of published data, both structured and unstructured, on the World Wide Web (WWW) is increasing continuously [63]. In particular, tree structured data (e.g., XML, JSON) is used commonly as a data model in various application areas including bioinformatics [52, 73], web mining [75, 91], and data integration and Web publishing [66]. Its popularity is largely due to the convenience of its self-descriptive and flexible semi-structured characteristics [86].

The increased volume of tree data has led to the development of different techniques for querying. Structured query processing has traditionally been a popular way of searching on tree data [58]. Formal query languages such as XPath [22] and XQuery [23] have been designed to take advantage of the tree structure of XML to retrieve information. In order to formulate queries with these languages, the users have to learn their syntax and be familiar with the schema of the data (including element tags and attribute names).

Keyword search techniques have been employed as a technique for retrieving information from tree data to allow a more flexible search. Keyword search has been established in recent years as the most popular technique for searching on the web. It became initially popular as a technique for searching flat (unstructured) documents [5] but it soon expanded its popularity to structured [34] and semi-structured data [58]. The reason of the popularity of keyword search on tree-structured data is twofold: (a) the users can retrieve information from the web without mastering a complex query language (e.g., XQuery), and (b) they can issue queries against the data without having full or even partial knowledge of the structure (schema). Therefore, the same query can be issued against multiple, differently structured data sources on the web. Following is an example of application of keyword search on tree data.

Example 1.1. Figure 1.1 shows a sample data tree. This tree contains information about courses offered in a university. Consider also the keyword query {Physics, James, Harrison} against this data tree. As one can see, each keyword has multiple instances (nodes that contain the keyword) in the tree which are shown in bold. There are courses on Physics offered by one or two instructors whose name contains James and/or Harrison, and therefore it is reasonable to assume that the user is looking for such courses. The candidate results of a keyword query on a data tree can be defined as the minimum connecting trees (MCTs) [35] in the data tree that contain an instance of all the keywords. The roots of the MCTs are the lowest common ancestors (LCA) of the included keyword instances and are often used to identify the candidate results [35, 32, 71, 88, 89]. Assuming that our results are represented by LCA nodes, there are five candidate results for our keyword query whose node ids are shown in the figure encircled.

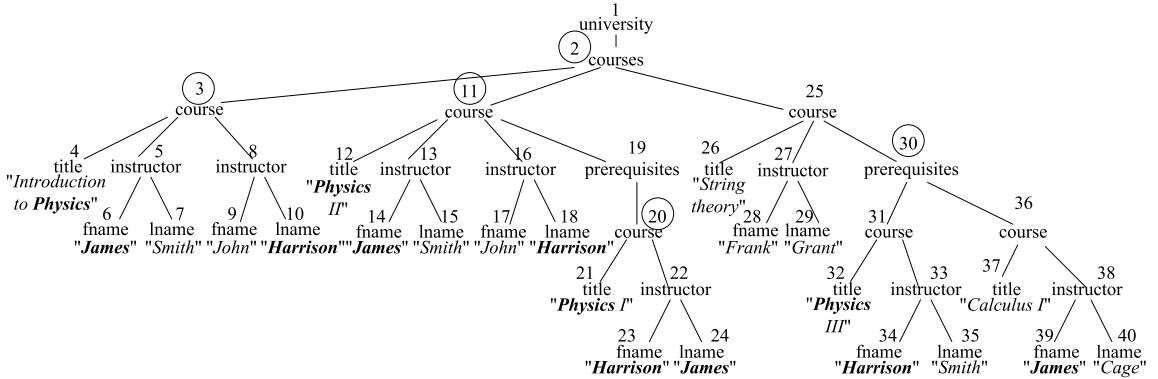


Figure 1.1 A data tree T .

1.1 Focus of the Investigation

As shown in the example above, keyword search is very convenient from the users' end. However, there are several issues related to keyword search that has to be addressed in order to develop effective and efficient keyword search systems over tree data. Two different aspects of keyword search on tree data are investigated in this dissertation: (a) assigning

semantics to keyword queries and (b) disambiguating keyword search results for facilitating browsing.

1.1.1 Keyword Search Semantics

Keyword queries are inherently imprecise both on structured and unstructured data [37, 58]. The imprecision becomes even more apparent on structured data since the queries cannot specify structural constraints. Since the results of keyword search on tree data are not whole documents but fragments of trees, an exponential explosion of candidate results is possible. As a consequence, keyword queries usually return a very large number of results on structured data of which only a tiny portion are relevant to the query. The large number of candidate results makes it difficult or even impossible for the users to identify the interesting results.

Many recent works focus on addressing the problem of the multitude of candidate LCAs by appropriately assigning semantics to keyword queries on XML data. A number of these semantics, characterized as *filtering*, aim at filtering out a subset of the candidate LCAs that are irrelevant. Some filtering semantics prune LCAs based exclusively on *structural* information (e.g., SLCA semantics [35, 55, 88] and ELCA semantics [32, 89]) while others take also into account *semantic* information, that is, the labels of the nodes in the data tree (e.g., the Valuable LCA or VLCA semantics [21, 43], and the Meaningful LCA or MLCA semantics [48]). Other recent works assign *ranking* semantics to keyword queries, that is, they rank the results aiming at placing on top those that are more relevant [6, 18, 21, 32, 65, 79]. Ranking the results improves the usability of the system by allowing the users to find their intended results in the top ranks of the list. In order to perform the ranking these works exploit: (a) structural characteristics of the results, and/or (b) statistical information or information theory metrics adapted to the tree structure of the data. All the ranking approaches rank the results based on some scoring function which assigns scores

to the results.

The problems. Although filtering approaches are intuitively reasonable for specific cases of data, they are ad-hoc and they are frequently violated in practice resulting in low precision and/or recall [79]. Further, most ranking approaches are combined with filtering approaches, that is, they rank only the LCAs accepted by the respective filtering semantics, this way inheriting the low recall of the filtering semantics. This weakness is due to the fact that most existing filtering semantics do not examine and compare the way the keyword instances are combined in the data tree to form tree patterns. Instead, most of them depend on the structural relationships of the keyword instances and LCAs locally in the data tree. However, local relationships are not sufficient and a global view of the results is necessary in order to decide effectively on their relevance. Consider for instance the following example.

Example 1.2. *Consider again Figure 1.1 as the data tree and the keyword query {Physics, James, Harrison}, there are five candidate results whose node ids are shown in the figure encircled. Among them only node 3, 11 and 20 are relevant results since node 30 represents the prerequisites of a course and node 2 represents the set of all the courses offered by the University. The ELCA semantics filters out candidate LCAs whose keyword instances are descendants of other descendant candidate LCAs, while SLCA prunes candidate LCAs that are ancestors of other candidate LCAs. In the context of our example, both ELCA and SLCA return the wrong result 30 and SLCA misses the correct result 11. One version of the MLCA semantics excludes candidate LCAs if the LCA of two of its keyword instances is not also an ELCA node of the labels of these instances. Therefore, it fails to return the correct result 3 because the LCA of the keyword fname (the label of the instance 6 of keyword James) and lname (the label of the instance 10 of keyword Harrison) is not also an ELCA of fname and lname (the label lname of node 10 comes closer to the label fname of node 9). Further, it incorrectly returns the node 30 (prerequisites).*

The reason of these failures is that all these approaches are based on the relation-

ship of the keyword instances locally (e.g., whether two LCAs have an ancestor-descendant relationship) and they miss a global view of how the keyword instances are combined in the data tree.

Our approach. In this dissertation, we argue that a meaningful semantics for keyword queries does not depend on the local properties of the keyword instances in the data tree but on the *patterns* the keyword instances define on the data tree. Each pattern might and usually does represent many results. For instance, Figure 1.2 shows three patterns for the query $\{Physics, James, Harrison\}$ on the data tree of Figure 1.1. These patterns which are minimal trees rooted at the root of the XML tree and containing all the keywords indicate different ways the keyword instances in the data tree are combined to form results. Pattern (a) represents the LCAs 3 and 11, pattern (b) the LCA 20 and pattern (c) the LCA 30.

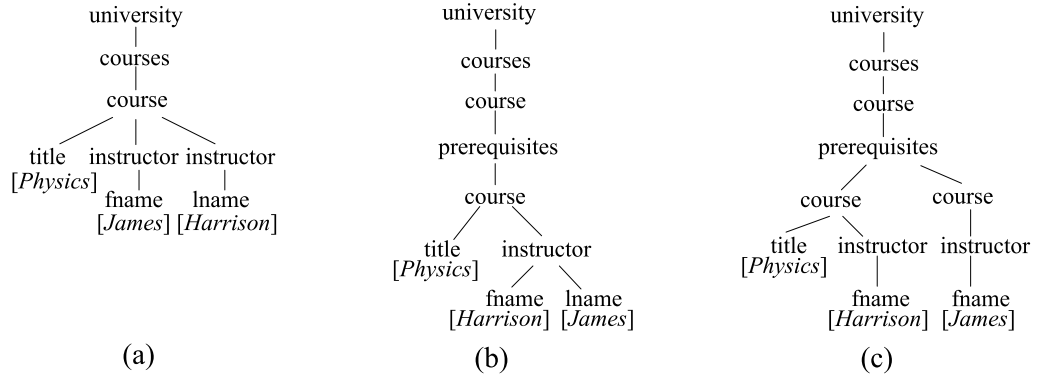


Figure 1.2 Three patterns of the query *Physics, James, Harrison* on the data tree of Figure 1.1.

Further, we argue that assigning simply a score to the results is not sufficient for producing a ranking of high quality. Rather, a ranking or a filtering of the results should be obtained by directly comparing the different structural and semantic properties of their patterns, that is, by *reasoning* on the patterns. For instance, by comparing the patterns of Figure 1.2 one can easily see that pattern (c) is not relevant in the presence of patterns (a) and (b) and if one wants to rank the patterns in terms of relevance to the query, patterns

(a) and (b) should precede pattern (c). These considerations apply to all the results these patterns represent. That is, the patterns act as representatives of the respective results.

Different techniques could be devised to compare patterns. In the present work, this comparison is realized based on *homomorphisms* between patterns. We define two types of homomorphisms between patterns and we use them to define different kinds of homomorphism relations on patterns. As the number of patterns is typically much smaller than the number of the results they represent, we show that this comparison is computationally feasible. Based on these relations, we organize the patterns of a keyword query into a graph of patterns which we leverage to determine a ranking for patterns and ranking semantics for queries thereof. We also provide filtering semantics for queries by selecting the top-k patterns in the ranking.

A number of approaches define filtering or ranking semantics without relying on the structural relationships of the keyword instances locally in the XML tree. As an example, Cohen et al. [21] filter out a query match containing a pair of keyword instances linked through a path in the XML tree which has duplicate labels. Schema-level SLCA [42] excludes LCAs whose label paths from the root of the XML tree are a proper prefix of that of another LCA. More recently, Liu et al. [53] cluster query results based on the patterns they comply with, and define filtering semantics based on the conceptual relationships between entity nodes (in the sense of the Entity-Relationship model) in the XML tree. Finally, Coherency Ranking [79] ranks query results based on an extension of the concepts of data dependencies and mutual information. None of these approaches define ranking or filtering semantics by globally comparing the structural and semantic properties of the query result patterns as we do in this work.

In Chapter 4, we describe in detail our approach called *XReason*, and the algorithms we designed to implement it. Comprehensive experiments on multiple datasets with different characteristics, presented in Section 4.7, compared our approach with previous ones in order to assess the effectiveness of *XReason* and the efficiency of our algorithms.

The results show that the *XReason* filtering and ranking semantics outperform previous approaches with respect to various metrics and our algorithms are fast and scale well with respect to the input and output size.

1.1.2 Disambiguation of Results

Another drawback of keyword search on tree data, also due to inherent imprecision of the keyword queries in expressing user intent, is the ambiguity of the type of the desired results. The answer of keyword queries on tree data usually contain different types of results which are meaningful with respect to the query even though the user is interested in only some of them. Consider the following example.

Example 1.3. *Consider the query $Q = \{\text{Advanced}, \text{Database}, \text{Systems}\}$ on the data tree of Figure 1.3, which represents a university database recording courses and seminars. There are courses with title “Advanced Database Systems” and a seminar whose topic also contains all the query keywords. The user might be interested in both courses and seminars or she can be interested only in one of them. Usually, the search systems do not group results and interleave in the answer results representing different concepts (e.g., courses or seminars). The users might have to examine many results which are not of interest to them in order to find their intended results. Even worse, assume there are multitudes of courses that match the keyword query but only a few seminars, a relevance ranked list might place all the courses at the top and the user might think that there are not any matches for seminars since they are not included in the top results.*

Result filtering [43, 48, 88, 89] and result ranking [6, 65] or a combination of them [2, 21, 32] have been proposed to address the problem of the large number of candidate results. In addition, some papers developed *top-k* algorithms for ranking and selecting the top-k results without necessarily generating of all the results [18, 44, 65]. The answers provided by these semantics are not satisfactory since still a large number of results may

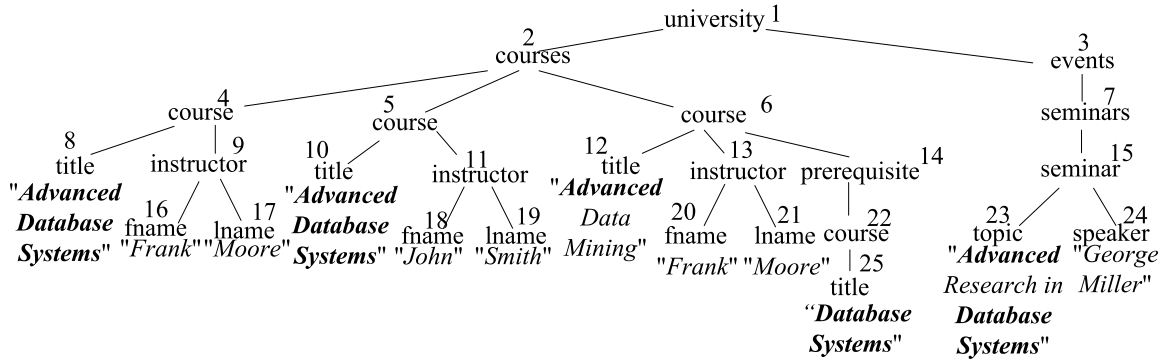


Figure 1.3 A data tree T .

qualify in the answer and the quality of ranking is low [79]. Further, without disambiguation of the results (i.e., identification of the different types of results), the users may still need to examine a large number of results which are not of interest to them.

In this dissertation, we address the ambiguity problem from two aspects: (a) clustering of keyword search results and (b) diversification of keyword search results.

Clustering keyword search results

In order to provide the users with a result structure that they can browse easily and reach their intended results easily, we propose applying clustering to the results of keyword search. The main idea is that if the results can be clustered effectively with respect to the semantics they represent, the users can only examine the clusters which are of interest to them and prune a huge number of results that are not interesting to them.

Clustering has been used as an alternative method of retrieving results in information retrieval [13, 40, 92]. The results of a query are clustered and the clusters form categories from which the users can choose their category of interest. Unfortunately, the techniques applied for clustering web search results are not directly applicable to keyword search over tree data because: (a) tree data have a structure, and (b) the granularity of the results is different, i.e., flat documents vs. fragments of documents with structure.

Clustering of tree data has been studied extensively before for data mining purposes

[24, 49, 51]. However, these techniques are not query-aware and they are applied at the document-level instead of at the result-level. Some recent studies have applied clustering on the results of keyword search on XML data. Liu et al. [53] introduce a methodology which clusters the XML search results based on the semantics that they represent. They cluster the results based on the patterns that they define. Liu and Chen [57] cluster the results based on the categories of the terms in the results, return nodes and predicates. Users can also specify the granularity of the clustering and input a requested number of clusters. The clustering methodology we present is a more refined one in the sense that it not only clusters the patterns of the results, but further clusters patterns according to the common subpatterns that they share by utilizing homomorphism relations defined on patterns. Also, we provide an ordering among clusters, a concept which has not been suggested before.

Our approach. The introduced clustering approach is a multi-level methodology for clustering the results of a keyword query on tree data. The results are clustered in three different levels. The clusters at every level are nested within the clusters of the higher level. For each cluster at each level, an appropriate representative is chosen as an interface to the user. The clusters of the bottom level consist of structural patterns of the keyword query results. These patterns represent the different interpretations of the query results. The clusters of the higher levels are formed by exploiting different homomorphism relations we introduce on patterns. The clusters at the highest (third) level and the clusters at the second level which are nested within the same parent (third level) cluster are organized into graphs. The edges in the graphs represent a partial order on the clusters and this partial order is exploited to produce a ranking for the clusters of the graph. The users can navigate through the system by selecting clusters initially at the top level and by drilling down to their nested clusters and finally to the results. The selection of clusters at every level is facilitated by the ranking of the relevant clusters which is provided to the user.

In Chapter 5 the details of the clustering methodology and our algorithm for its implementation are described. The experimental results obtained through extensive exper-

imentation on different real datasets show that our clustering hierarchy helps the users to effectively retrieve the relevant results and our approach outperforms a previous state-of-the-art clustering methodology in terms of reach time. Our algorithm constructs the result patterns and the clustering hierarchy efficiently, displaying interactive times. It also scales smoothly when the number of keywords and the size of the data increase.

Diversification of Keyword Search Results

Another possible solution to the ambiguity problem is through diversification of the results. Diversification aims to return the users a list of results (a subset of all results) that balances the relevance and diversity of the results in the list. Definition of diversity might vary according to different applications. For example, when a keyword query is applied to a set of news documents, diversity is mostly about the novelty of the information contained in the documents but in the regular keyword search it is possibly defined as the dissimilarity of the content of the documents [28].

Example 1.4. *Consider again the query $Q = \{\text{Advanced, Database, Systems}\}$ on the data tree of Figure 1.3, which represents a university database recording courses and seminars. If we want to retrieve a diverse list of results with only 2 results, an ideal list would contain a pattern that represents one of the relevance course results and the seminar pattern.*

There are also queries which do not focus only on a subset of results but rather they want an overview of the results. These exploratory queries should return a diverse set of results which covers different aspects of the result set [28]. Consider the following example.

Example 1.5. *Consider a bibliography database (possibly including the texts of the papers as well) and a user who wants to investigate the papers working on the topic of result diversification. Since the user is probably new to the area, she would like to see the big picture first and then, she might want to specialize on different domains. Providing first a*

diverse set of results to the user would summarize the entire result set better.

Search result diversification attracted a lot of attention in both information retrieval and recommendation systems [28, 30, 99]. With the help of a diversified result set, the systems can address the problems of result multitude and keyword query ambiguity, and support exploratory search. When the user intent cannot be inferred from a keyword query, result diversification can minimize the user dissatisfaction. It can also address the over-specialization problem which might be caused by a relevance ranked list. For example, consider the keyword *chelsea*. With this query, a user might be referring to the Chelsea district in London, the Chelsea Football Club in UK or the Chelsea neighborhood in Manhattan, New York. A ranked list solely based on relevancy may exhaustively return results of one of the types above, and other types might be ranked very low. A diverse set should include results of different aspects by keeping a balance between relevancy and diversity.

The diversification problem is usually defined as an optimization problem of selecting a subset of the result set of a query such that the diversity among these k results is maximized [28, 30]. One of the earliest works which focused on diversity among retrieved results is [12]. Carbonell and Goldstein [12] describe the concept of maximal marginal relevance (MMR) as a trade-off between relevance and novelty. There is a limited amount of work which elaborates on diversification of keyword search results on structured data which is surveyed in Section 2.3.

In this dissertation, we propose applying diversification techniques on the results of keyword search over tree data to provide the users with a diverse set of results. To this end, we formally define the diversification problem in the domain of tree data. We introduce a similarity measure between the patterns of keyword search results that can effectively be used to maximize the diversity of the results included in the result set. We also devise a relevance measure that can be combined with our similarity measure for building the diverse result set. Finally, we present a heuristic algorithm which builds a diverse result set incrementally by choosing the patterns to be included in a greedy fashion. The proposed

experimentation and an analysis of different evaluation metrics that can be applied to the diversification problem are also presented. We further elaborate on this issue in Chapter 6.

1.2 Organization

This dissertation is organized as follows. In Chapter 2, a review of the state-of-the-art is given for different aspects of keyword search on tree data. In Chapter 3, definitions and notation are provided for the data model we adopt. Chapter 4 introduces the proposed keyword search semantics for answering keyword queries over tree data. In Chapter 5, the clustering methodology for keyword search results over tree data is presented. In Chapter 6, we introduce the diversification of search results problem in the domain of tree data and propose our diversification scheme. Finally, Chapter 7 presents the conclusion and the directions for future work.

CHAPTER 2

STATE OF THE ART

This chapter reviews the literature on searching, result clustering and result diversification on tree data. Most of the studies reviewed in this chapter focus on XML data specifically but techniques are applicable to any kind of tree structured data with similar characteristics.

2.1 Searching on XML

XML documents are usually modeled as trees (in few studies, they are also modeled as graphs due to circular references [16, 85]). Nodes in this tree model represent elements and attributes. Also, the text associated with the XML elements or attribute values are either represented as separate text nodes or they are attached to their element or attribute node in the data tree. Description of the tree model considered in this work is given in Chapter 3.

Since XML has become a standard for exporting and exchanging data on the web, techniques to allow users query XML data have been developed over the years. In this section, two most frequently studied and popular techniques are mentioned.

2.1.1 Structured Query Languages

Due to the semi-structureness of XML, different structured query languages have been introduced. Two of the most well-known ones are XPath [22] and XQuery [23].

XPath [22] utilizes the tree structure of the XML documents and can retrieve nodes based on the provided path specifications in the user query. Paths can be specified as a series of element tags delimited by “/” or descendant paths are also accepted which are shown as “//”. For example, consider the XML tree in Figure 1.3 and the XPath expression `//instructor/fname`. This expression returns nodes 16, 18 and 20 in Figure 1.3.

XQuery [23] is another important XML querying language which is based on path expressions similar to XPath [22] but it has more advanced language constructs. It is a functional language where one can define loops and join information from different XML databases. For example, consider again the XML tree in Figure 1.3. The following query will return the first names of instructors who teach a course with a title containing the word *Advanced*.

```
for $a in //course/[contains(title, ``Advanced``)]
return $a/instructor/fname
```

Processing of tree pattern queries is the basis of structured query languages on XML. This problem has been addressed quite extensively in the literature [11, 31, 38, 87]. As a less restricted version of tree pattern queries, partial tree pattern queries have also been investigated on XML data [74, 81].

2.1.2 Keyword Search

Information Retrieval (Flat Documents) Keyword search is the most popular way of accessing information on the web. Search engines provide simple user interfaces to enter keyword queries and usually, the results are returned to the user in a ranked list which is ordered by the relevance of the results. There are different measures which are used to quantify the relevance of results (documents in WWW). PageRank [10] is a technique which estimates importance of a web page by the importance of other pages which have a link to that web page. The vector space model is a very commonly used model to represent documents in the literature [70]. Using the vector space model and a scoring function such as Term Frequency-Inverse Document Frequency (TF-IDF) [70], similarity of documents and the keyword queries can be computed. The TF-IDF function scores terms in a document based on their frequency of occurrence in the document (proportional) and their

frequency of occurrence in the entire document base (inversely proportional).

Structured Databases. Keyword search on databases is a complicated task. In contrast to the flat documents in the classical information retrieval model, the proposed approaches should also take into account the structure of the databases. The structure of the databases have two very important effects. First, the information which can be gained from the structure should be utilized by the approaches. Second, the results of keyword queries on databases is not the whole database but rather a portion of the database: In case of XML, this might be a subtree of the XML tree whereas in case of relational databases, this might be a subgraph.

Keyword search has been addressed in structured databases [8, 34, 50, 62]. Keyword search techniques applied on the Web cannot be applied directly on relational databases. This is due to the fact that the information in relational databases are spread over multiple tables and the structure of the data should also be utilized during the keyword search [62]. Relational databases and the query answers are usually modeled as graphs [8, 34] and some adaptations of information retrieval techniques have been used for assigning semantics to keyword queries [62].

Semi-structured Databases. Semi-structured databases such as XML usually do not follow a strict schema, so combining data from different parts of the database is more challenging than in fully structured databases [42]. Since keywords can be related through different instances in the XML tree, an important task of XML keyword search is to be able to identify results with meaningful interpretations of the query. Because of this, several papers elaborate on filtering semantics for keyword search on XML data. The results are usually modeled as the lowest common ancestors (LCAs) of the keyword instances or subtrees of the XML tree which contains a query match. Most of the filtering semantics are based exclusively on the structural properties of the results and only few of them take into account the semantic information (that is the labels of the nodes). The SLCA [77, 88], ELCA [32], XSearch [21], VLCA [43] and MLCA [48] and their properties are extensively reviewed

in Section 4.5. Schema-level SLCA [42] excludes LCAs whose label paths from the root of the XML tree are a proper prefix of that of another LCA. In [80], tree pattern queries are extracted from the structural summary and those which present a meaningful result are used. MaxMatch [55] groups SLCA nodes and eliminate some keyword matches under these subtrees by considering additional rules. Consistency and monotonicity concepts are also introduced in MaxMatch as an axiomatic framework for evaluation of keyword search semantics. Kong et al. [39] improve over MaxMatch by considering all LCAs instead of only SLCA nodes, and address MaxMatch’s false positive and redundancy issues. XMean [53] defines conceptually related entity nodes to find relevant results. Ancestor entity nodes of two nodes in the XML data are utilized to decide if they are meaningfully related.

Ranking semantics for answering XML keyword queries return a ranked list of results (LCA nodes or subtrees) with respect to their relevance to the user query. XRank [32] uses a variation of PageRank algorithm to rank the results. XSearch [21] ranks the results using a TF-IDF function [70] adapted to the tree structure of XML documents. XReal [6] introduces a similarity function to rank nodes with respect to their similarity to the query. Termehchy and Winslett [79] exploit mutual information to calculate coherency ranking measures for ranking the query answer. Nguyen and Cao [65] use mutual information to compare results and to define a dominance relationship between the results for ranking. SAIL [44] introduces the concept of minimal-cost trees and identifies the top-k answers by using link analysis and keyword-pair relevancy.

The proposed approach in this dissertation contains both filtering and ranking semantics [2]. It is different from the previously proposed semantics because it considers patterns of results both for filtering and ranking instead of relying on local properties of results as, e.g., SLCA [77, 88] and ELCA [32] does. In addition, for ranking, instead of assigning simply a score to the results, the proposed approach obtains the ranking by directly comparing the different structural and semantic properties of patterns. Further explanation of the semantics is given in Chapter 4.

Some works focus on developing efficient algorithms for XML keyword search semantics. Algorithms for finding SLCA and ELCA for a keyword query are presented in [88, 89, 95, 98]. Hristidis et al. [35] develop efficient algorithms for finding a compact representation of the result subtrees. In [26, 27], a multi-stack algorithm to return a size-ranked result list to a keyword query is presented. Chen and Papakonstantinou [18] introduce algorithms to support top-k SLCA and ELCA calculation. The common-ancestor-repetition (CAR) problem has been addressed by [96, 97]. While trying to find SLCA/ELCA nodes, approaches perform two basic operations: testing the order of two nodes and computing the LCA of two nodes. Usually the Dewey encoding scheme is used for labelling nodes. Since Dewey labels consists of components of all of the ancestors, these operations are equal to visiting all of the common ancestor nodes [97]. In [97], a new indexing structure is introduced to overcome this problem which assigns a unique id to each node which is consistent with the document order.

Additionally, different problems within the context of XML keyword search have been addressed in some studies. XReal [6, 7] and XBridge [46] propose approaches to find the user-intended result type. XReal [6, 7] defines the type of a node as the label path from the root to the node in the XML tree. A variation of TF-IDF is used to find the candidate result type of a query. XBridge [46] uses a scoring measure for the types as well, but it also takes into account the structure of the results while scoring the type. XSeek [54] utilizes entity, attribute or connection nodes to decide upon the nodes to be returned in the results. XMean [53], and Liu and Chen [57] address the problem of clustering XML keyword search results. XMean [53] uses patterns of the results to define clusters. In order to facilitate browsing the results, a relaxation graph for the patterns is created. Liu and Chen [57] built on XSeek [54] to detect the nodes to be included in the results and the results are clustered by using the types of keyword instance nodes (i.e., entity or attribute). A system called eXtract is introduced in [36] to address the problem of creating snippets for XML keyword search results. Context-sensitive keyword search on XML is addressed

in [9]. The context is defined in the form of a path in the XML tree and the results are ranked by taking into account the specified context. Materialized views for supporting the evaluation of XML keyword queries have been proposed in [56, 72]. Keyword query refinement and/or keyword suggestion techniques in the XML keyword search context are studied in [61, 67]. Most of these problems have been summarized in [58].

2.2 Search Result Clustering

Even though querying a data source with keyword queries is convenient, it has an ambiguity problem. Keyword queries are limited in terms of the information that they can represent. In WWW, one can usually provide additional keywords to make the search space narrower. Even then, there might be different types of results where the user is interested in only some of them.

Result clustering has been applied to solve the ambiguity problem in the domain of information retrieval [40, 92]. Results of a query are clustered, the clusters are labeled with representative terms [41, 84] and then users can choose their cluster of interest by examining the cluster labels. Some commercial examples are also available such as the clustering engine, Carrot [14, 76]. Unfortunately, the techniques applied for clustering web search results are not directly applicable to XML keyword search results because XML documents are structured and the granularity of clustering is different, i.e., flat documents vs. fragments of documents with structure.

Clustering XML keyword search results has not been studied extensively. In addition to the problem of query ambiguity, multitude of keyword search results on XML is another motivation of application of clustering to the XML keyword search results. Clustering of whole XML documents has been studied before for data mining purposes [24, 49, 64]. However, these techniques are not query-aware and they are applied on the document-level instead of the result-level. Some recent studies have applied clustering on the results of keyword search on XML data. Liu et al. [53] introduces a methodology which clusters

the XML search results based on the semantics that they represent. They cluster the results based on the patterns that they define. Liu and Chen [57] clusters the results based on the categories of the terms in the results; return nodes and predicates. Users can also specify the granularity of the clustering and input a requested number of clusters.

In addition to clustering, XReal [6, 7] and XBridge [46], as mentioned above, addresses the problem of identification of user intent. These approaches might also serve as a basis of disambiguation but still they lack clustering of semantically similar results together which would further help the users.

The clustering methodology presented in this dissertation [3] is a refined one in the sense that it not only clusters the patterns of the results, but further clusters patterns according to the common subpatterns that they share by utilizing homomorphism relations that are defined. Also, an ordering among the clusters is provided which has not been studied before. More details about the proposed clustering methodology is given in Chapter 5.

2.3 Search Result Diversification

Search result diversification attracted a lot of attention in both information retrieval and recommendation systems. The first reason of this interest is again the ambiguity of the query in expressing the user intent. Result diversification aims to present a diverse set of results to minimize user dissatisfaction. This is also an attempt to solve the over-specialization problem [28] where a highly homogeneous set of results is returned to the user due to relevance-based ranking and/or personalization. Over-specialization problem has been addressed in recommendation systems [90, 93, 99] and information retrieval [68] by trying to retrieve items which are dissimilar to each other.

In addition to the ambiguity of keyword queries, there are users who do not focus only on a subset of results but rather they want an overview of the results. These queries are in exploratory nature [28] and in this case, providing a diverse set of results which covers different aspects of the entire result set is more important than providing a ranked list solely

based on relevance.

In general, the diversification problem is defined as selecting a subset of the result set with k results such that the diversity among these k results is maximized [28, 30]. One of the earliest works which focused on diversity among retrieved results is [12]. Carbonell and Goldstein [12] describe the concept of maximal marginal relevance (MMR), a trade-off between relevance and novelty. Gollapudi and Sharma [30] give an axiomatic approach for result diversification. Liu et al. [60] address the problem of query expansion by expanding queries from clustered results. Expanded queries can be taken as different interpretations of the original query. Drosou and Pitoura [28] review different definitions of diversity, and examine the algorithms and evaluation metrics. They categorize diversity definitions as content-based [99], novelty-based [19, 94] and coverage-based [1]. Tian et al. [82, 83] consider the ambiguity problem in the context of web search by suggesting query completions for homonyms.

There is a limited amount of works on diversification of search results on databases. Demidova et al. [25] proposed a technique to diversify keyword search results on structured databases. Li et al. [47] addressed the ambiguity of XML keyword queries by expanding the keyword queries with additional keywords to narrow down their result set. Liu et al. [59] introduce an approach to differentiate search results which can be used as a basis for diversification. Hasan et al. [33] introduced an extension of tree edit distance for diversification of XML search results for structured queries.

CHAPTER 3

TREE DATA MODEL

3.1 Data Model

In this dissertation, we address keyword search and result disambiguation problems on tree data. The most frequently used type of tree data is XML data. Therefore, we define our data model in accordance with the XML data but it is generic enough to handle any kind of conventional tree data.

As is usual, we view XML documents as ordered node labeled trees. In our study, nodes represent and are labeled by elements and attributes. The leaf nodes may have a content which is text. Edges represent element to element and element to attribute relationships. For any two nodes n and n' in an XML tree T , $n < n'$ ($n > n'$) denotes that n is an ancestor (descendant) of n' in T . Without loss of generality, we assume that the label of the root of the XML tree is unique. A function *label* on a node returns the label of that node. We want to allow keywords to match not only the content of a node but also its label. To this end, we define a function *value* on nodes in Definition 3.1.1.

Definition 3.1.1. *Let n be a node in a data tree T , $value(n)$ returns the set of words in the content and the label of the node.*

If $value(n)$ of a node n includes a keyword k we say that n *contains* keyword k and that node n is an *instance* of k . We assume that XML tree nodes are enumerated using the Dewey encoding scheme [78]. The Dewey encoding scheme allows easily determining the LCA of multiple nodes and can be efficiently exploited by stack based algorithms for computing query matches.

Figure 3.1 shows an XML tree which is a variation of the one shown in the introduction. The data tree represents a university database which consists of courses and seminars. The values of the leaf nodes are presented in double quotation marks. Dewey indices of the

nodes are omitted for clarity. Plain numbers are used instead to identify the nodes. Note that the indices of the nodes are assigned in a depth first manner.

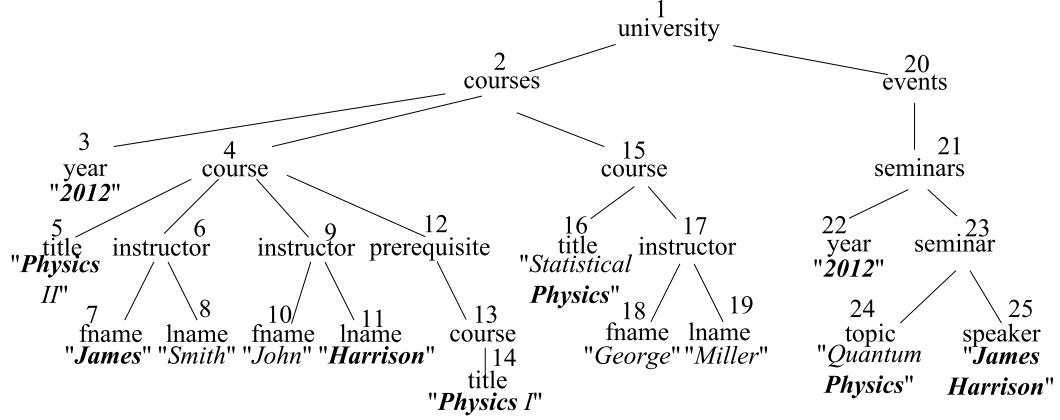


Figure 3.1 An XML tree T .

3.2 Keyword Queries

A (keyword) query Q is a set of keywords $\{k_1, k_2, \dots, k_n\}$. In the context keyword search over XML data, keyword queries are embedded to XML trees. Next, we define the *instance* of a query on a data tree.

Definition 3.2.1. Let Q be a query and T be an XML tree. An instance of Q on T is an embedding of Q to T (i.e., a function from Q to the nodes of T that maps every keyword k in Q to an instance of k in T).

We overload the term “query instance” and we use it to refer both to the function that maps the query keywords to the tree nodes and to the images of the query keywords under this function. Note that two query keywords can be mapped to the same tree node. We define the trees that represent query instances in Definition 3.2.2.

Definition 3.2.2. Let Q be a query, T be an XML tree, and I be an instance of Q on T . The instance tree (IT) of I is the minimum subtree S of T such that: (a) S is rooted at the root of T and comprises all the nodes of I , and (b) every node n in S is annotated by the keywords

which are mapped by I to n . The minimum connected tree (MCT) of I is the minimum subtree of S that comprises the nodes of I .

Clearly, the root of the MCT is the *Lowest Common Ancestor* (LCA) of the nodes of I in T .

Consider the XML tree of Figure 3.1 and the keyword query $Q = \{Physics, James, Harrison, 2012\}$. Figures 3.2(a) and (b) show the IT and the MCT, respectively, of the instance $\{(Physics, 24), (James, 25), (Harrison, 25), (2012, 22)\}$ of Q on T . In the figures, the annotation of the nodes is shown between square brackets by the nodes. The MCT of this IT is rooted at node 21 (the LCA of the keyword instances). The IT also contains the path from the root of T to node 21. In the following we identify an IT and an MCT by their corresponding query instance.

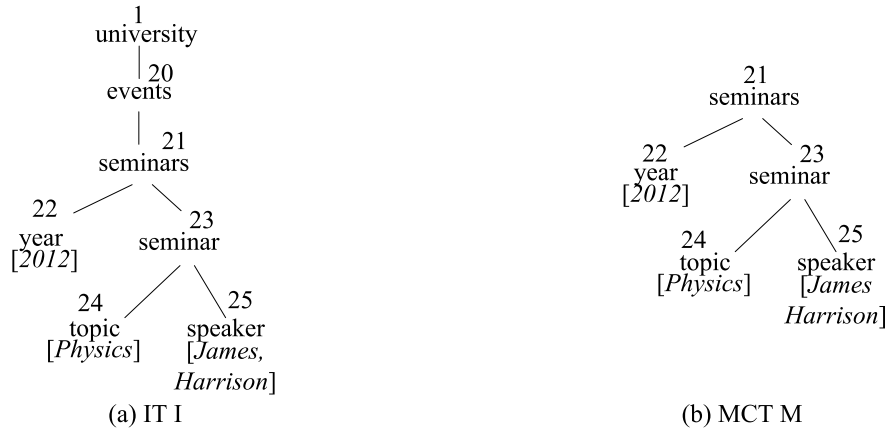


Figure 3.2 (a) An IT and (b) its MCT.

Note that, MCTs and ITs comprise besides the structural information also semantic information (the labels of the nodes). An IT of an instance of a query Q on an XML tree T is also called IT of Q on T . Several previous approaches return to the users LCAs as results. In our approach we study keyword search on XML trees by assuming that the results of keyword queries are ITs. An IT is much of a richer construct than an LCA in terms of the information it provides as it shows both: (a) how the keyword instances are combined under their LCA to form an MCT, and (b) how the LCA is linked to the root of

the XML tree.

Given a keyword query Q and an XML tree T , the set \mathcal{C} of the ITs of all the instances of Q on T is the set of the candidate results of Q on T . The answer of a keyword query Q on an XML tree T is a *subset* of \mathcal{C} . Which specific subset forms the answer of a query depends on the semantics adopted. In our approach, the answer is determined by comparing the patterns (to be defined in 4) of the ITs. For the needs of this dissertation, this comparison is realized based on different types of homomorphisms. In the next chapter, we define formally these homomorphisms and then we use them to provide ranking and filtering semantics to the queries. Other semantics will be presented in terms of ITs and compared with our approach in Section 4.5.

CHAPTER 4

SEMANTICS OF KEYWORD QUERIES OVER TREE DATA

In order to define semantics for queries we introduce patterns of ITs and homomorphisms between patterns and study their properties.

4.1 Instance Tree Patterns

Definition 4.1.1 (Instance tree pattern). *A pattern P of a query Q on an XML tree T is a tree which is isomorphic (including the annotations) to an IT of Q on T . The MCT of a pattern P refers to P without the path that links the LCA of the annotated nodes to the root of P .*

Multiple ITs of Q on T can share the same pattern. Figure 4.1 shows eight patterns (out of 15 in total) of the keyword query $Q = \{Physics, James, Harrison\}$ on the XML tree T of Figure 3.1. All patterns except pattern P_8 have one IT. Pattern P_8 has two ITs which comply with it: the IT of the query instance $\{(Physics, 5), (James, 25),$

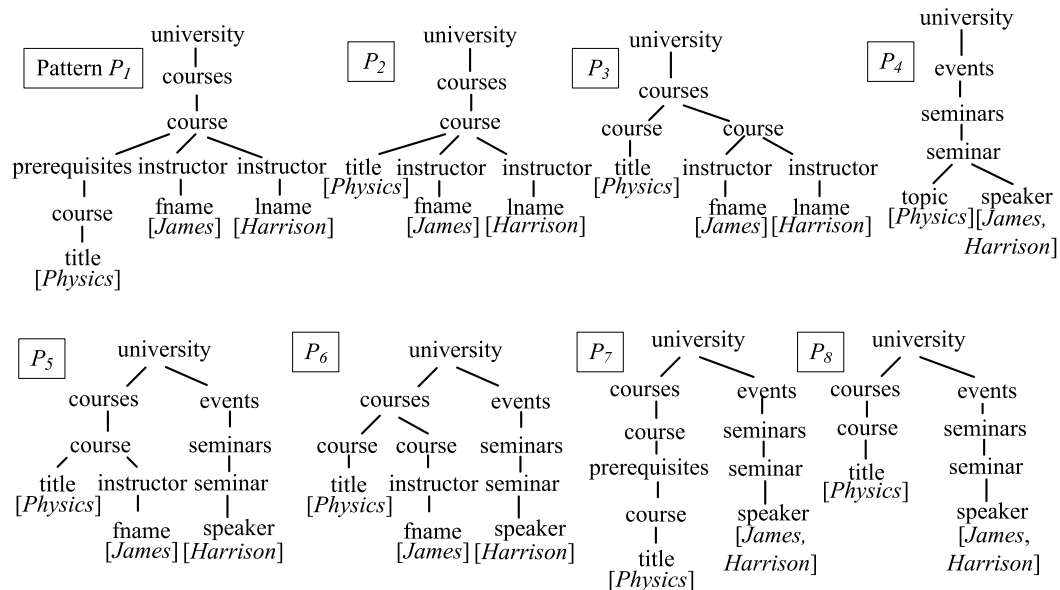


Figure 4.1 Some patterns for $Q = \{Physics, James, Harrison\}$ on the tree of Figure 3.1.

$(Harrison, 25)\}$ and the IT of the query instance $\{(Physics, 16), (James, 25), (Harrison, 25)\}$.

The function $ann(n)$ on a node n of a pattern returns the annotation of n if the node is annotated or, an empty set, otherwise. We also define a function $size(P)$ which returns the number of edges of P . The size of pattern P_3 is 8 and that of its MCT is 7. The size of pattern P_1 is 9 and that of its MCT is 7.

4.2 Pattern Homomorphism and Homomorphism Relation

Definition 4.2.1 (Pattern homomorphism). *Let S and S' be two subtrees of patterns of a query on an XML tree. A homomorphism from S to S' is a function h from the nodes of S to the nodes of S' such that:*

- (a) *for every node n in S , n and $h(n)$ have the same labels.*
- (b) *if n_2 is a child of n_1 in S , $h(n_2)$ is a child of $h(n_1)$ in S' , and*
- (c) *for every node n in S , $ann(n) \subseteq ann(h(n))$.*

Figure 4.2 shows the MCTs M , M' , and M'' of three patterns of the query $Q = \{2012, James, Harrison\}$ on an XML tree. As we can see in this figure there are homomorphisms from M to M' and from M' to M'' but not from M' to M or from M'' to M' . Observe that M' can be obtained from M (and M'' from M') by merging paths with the same sequence of labels starting from the same node and by unioning their annotations. For instance, M'' can

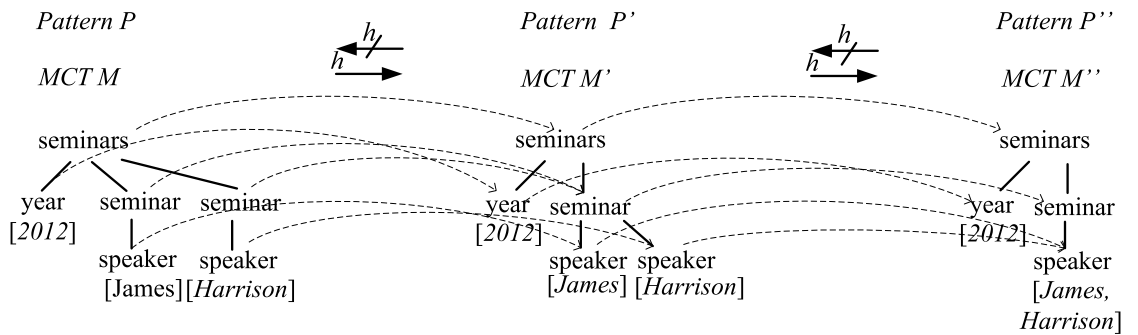


Figure 4.2 Pattern MCTs M , M' and M'' and homomorphisms between them.

be obtained from M' by merging the paths `seminars/seminar/speaker` from the node `speaker` and by unioning the annotations $[James]$ and $[Harrison]$. One can see that, in general, this is also a necessary condition for the existence of a homomorphism between two patterns when those two patterns are not identical.

Clearly, if there is a homomorphism from the MCT of a pattern P to the MCT of a pattern P' , the keyword instances are more closely related in any instance of P' than in P . Thus, we consider the ITs of P' to be more relevant to the query than those of P .

Based on the existence of a homomorphism between two patterns, we can define a relation \prec_h (called *homomorphism* relation) on patterns in order to compare their relevance to the query.

Definition 4.2.2 (\prec_h relation). *Let P and P' be two patterns of a query Q on an XML tree T . $P \prec_h P'$ iff there is a homomorphism from the MCT of P' to the MCT of P but not vice versa.*

For instance, for the patterns P , P' and P'' whose MCTs M , M' and M'' , respectively, are shown in Figure 4.2, $P'' \prec_h P'$ and $P' \prec_h P$, but $P' \not\prec_h P''$ and $P \not\prec_h P'$. That is, P' is more relevant than P , and P'' is more relevant than P' . Similarly, for the patterns P_5 and P_6 of our running example shown in Figure 4.1, one can see that $P_5 \prec_h P_6$.

One can easily see that if $P \prec_h P'$, $size(M) < size(M')$ where M and M' are the MCTs of P and P' , respectively. The following property which will be used later can be easily shown for the \prec_h relation.

Proposition 4.2.1. *The relation \prec_h on the set of patterns of a query on an XML tree is a strict partial order.*

Proof. \prec_h relation satisfies all three conditions of a strict partial order.

- (a) The \prec_h relation is irreflexive: $P \prec_h P'$ iff there is a homomorphism from the MCT of P' to the MCT of P but not vice versa. Therefore, P cannot be equal to P' .

- (b) The \prec_h relation is transitive: for any three patterns P, P' and P'' with MCTs M, M' and M'' , respectively if $P \prec_h P' \prec_h P''$ then there is a homomorphism from M'' to M' and M' to M . Therefore, there is also a homomorphism from M'' to M . Since $size(M) < size(M') < size(M'')$, there is no homomorphism from M to M'' . Thus, $P \prec_h P''$.
- (c) The \prec_h relation is antisymmetric: for any two distinct patterns P and P' , if $P \prec_h P'$, there is a homomorphism from the MCT of P' to the MCT of P but not vice versa. Therefore, $P' \not\prec_h P$.

□

Even though \prec_h correctly characterizes relevance, it is not sufficient. Consider for instance the MCTs M_1 and M_2 of the patterns P_1 and P_2 , respectively, shown in Figure 4.3 (ignore the dashed arrows for the moment). Even though $P_2 \not\prec_h P_1$, P_2 is more relevant than P_1 . Indeed, P_2 relates two instructors to a course they offer while P_1 relates two instructors to a prerequisite of a course they offer. In the next section we further exploit different kinds of relations in order to better capture this relevance relationship between patterns and their ITs thereof.

4.3 Path Homomorphism and Path Homomorphism Relations

In order to define additional relations on patterns, we introduce below a new type of homomorphism.

Definition 4.3.1 (Path homomorphism). *Let p and p' be two paths of two patterns, such that p ends at a node n annotated by a keyword k . We say that there is a path homomorphism from p to p' if there is a function ph from the nodes of p to the nodes of p' such that:*

- (a) *for every node n_1 in p , n_1 and $ph(n_1)$ have the same labels.*
- (b) *if n_2 is a child of n_1 in p , $ph(n_2)$ is a child of $ph(n_1)$ in p' , and*
- (c) *$k \in ann(ph(n)) \cup label(ph(n))$.*

Figure 4.3 shows the MCTs M_2 and M_1 of the corresponding patterns P_2 and P_1 (shown in Figure 4.1) for the query $\{Physics, James, Harrison\}$ on the XML tree of Figure 3.1. For every path from the root of M_2 to a node annotated by a keyword, there is a path homomorphism to a path in M_1 (the different types of dashed arrows indicate these path homomorphisms). However, the opposite is not true that is, there is at least one path of M_1 (in fact, only the path `course/prerequisites/course/title[Physics]`) that does not have a path homomorphism to a path in M_2 .

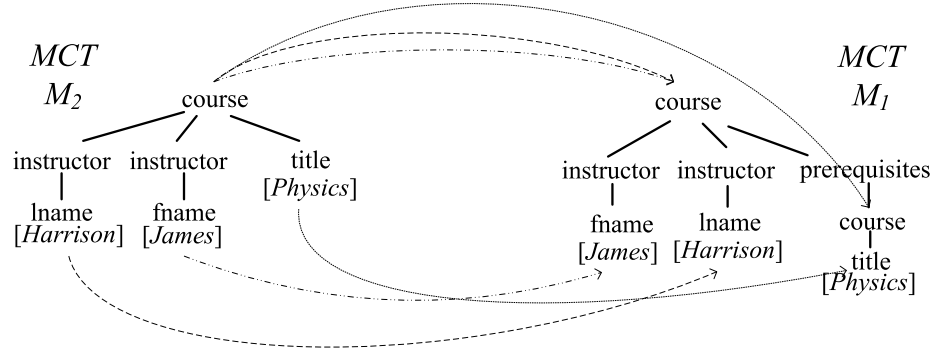


Figure 4.3 Pattern MCTs M_2 and M_1 and three path homomorphisms from the paths of M_2 to paths of M_1 .

Our intuition is that if P and P' are two patterns of a query on an XML tree, and every path from the root of the MCT of P to a keyword annotated node has a path homomorphism to a path in the MCT of P' , then the keyword instances in P are more meaningfully related than in P' because every sequence of labels from the LCA to a keyword instance in P also appears in P' .

In order to compare the relevance of query patterns, we use now path homomorphisms to define a relation \prec_{aph} (called *all_path_homomorphism* relation) on patterns.

Definition 4.3.2 (\prec_{aph} relation). *Let P and P' be two patterns of a query Q on an XML tree T . $P \prec_{aph} P'$ iff the following two conditions hold:*

- (a) *for every path p from the root of the MCT of P to a node annotated by a keyword, there is a path homomorphism from p to a path of the MCT of P' .*
- (b) *Property (a) does not hold in the opposite direction that is, from P' to P .*

As an example, observe that for the pattern MCTs M_2 and M_1 of Figure 4.3, $P_2 \prec_{aph} P_1$. That is, the \prec_{aph} relation correctly characterizes P_2 as more relevant than P_1 .

Consider also another example which involves mapping a keyword to a label or a value of a node: Figure 4.4 shows the MCTs M_a and M_b of two query patterns P_a and P_b , respectively. As it can be seen from the annotations, the keyword query is $\{Seminar, Abstract, Mathematics\}$. $P_a \prec_{aph} P_b$ since the three paths `seminar[seminar]`, `seminar/abstract[abstract]` and `seminar/abstract[mathematics]` in P_a have a path homomorphism to a path in P_b but the path `seminar/title[abstract]` of P_b does not have a path homomorphism to path in P_a . Here again \prec_{aph} correctly favors P_a over P_b .

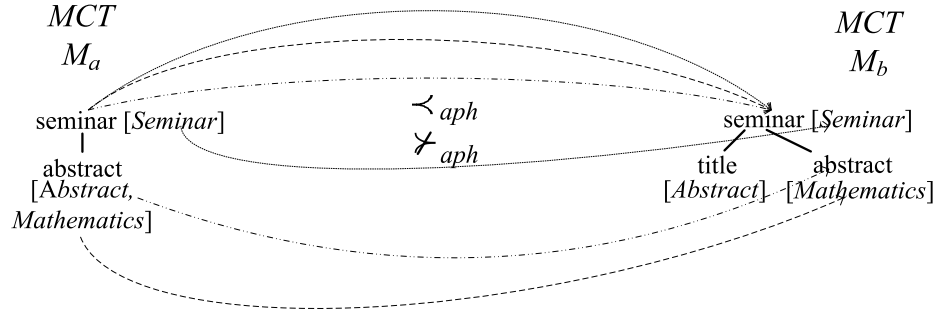


Figure 4.4 Pattern MCTs M_a and M_b and three path homomorphism from the paths of M_a to paths of M_b .

We now show a property of relation \prec_{aph} .

Proposition 4.3.1. *The relation \prec_{aph} on the set of patterns of a query on an XML tree is a strict partial order.*

Proof. \prec_{aph} relation satisfies all three conditions of a strict partial order. In the following, we refer to a root-to-annotated-node path of an MCT as a path for simplicity.

- (a) The \prec_{aph} relation is irreflexive: According to Definition 4.3.2, $P \prec_{aph} P'$ iff for every path p from the root of the MCT of P to a node annotated by a keyword, there is a path homomorphism from p to a path of the MCT of P' but not vice versa. Therefore, P cannot be equal to P' .

- (b) The \prec_{aph} relation is transitive: for any three patterns P, P' and P'' with MCTs M, M' and M'' , respectively, if $P \prec_{aph} P' \prec_{aph} P''$ then property (a) of Definition 4.3.2 holds from M to M' and from M' to M'' . Therefore, this property also holds from M to M'' . Since property (a) of Definition 4.3.2 does not hold from M' to M and M'' to M' , then there is at least one path in M'' that cannot be mapped with a path homomorphism to a path in M . Thus, property (a) of Definition 4.3.2 does not hold from M'' to M . As a consequence, $P \prec_{aph} P''$.
- (c) The \prec_{aph} relation is antisymmetric: for any two distinct patterns P and P' , if $P \prec_{aph} P'$, then for every path p from the root of the MCT of P to a node annotated by a keyword, there is a path homomorphism from p to a path of the MCT of P' but not vice versa. Therefore, $P' \not\prec_{aph} P$.

□

We next examine how \prec_h and \prec_{aph} are related. In Figure 4.2, one can see that besides homomorphisms from M to M' , and M' to M'' , for every path p from the root to an annotated node of an MCT there is a path homomorphism to a path in any one of the other MCTs. This is expected due to the following proposition.

Proposition 4.3.2. *Let M and M' be two pattern MCTs of a query on an XML tree. If there is a homomorphism from M to M' , then: (a) for every path p from the root of M to a node annotated by a keyword, there is a path homomorphism from p to a path of M' , and (b) for every path p' from the root of M' to a node annotated by a keyword, there is a path homomorphism from p' to a path of M .*

Nevertheless, if for every path p from the root of M to a node annotated by a keyword, there is a path homomorphism from p to a path of M' , then there is not necessarily a homomorphism from M to M' or from M' to M .

Proof. This proposition can be derived using the properties of the pattern homomorphism.

We mentioned that if there is a pattern homomorphism from an MCT M to another MCT

M' , M can be transformed into M' by merging some paths on the nodes with the same labels and unioning their annotations. This means that M and M' contains the same root-to-annotated-node paths. So, all paths of M can be mapped to a path in M' and vice versa. However, such a reasoning is not true for path homomorphisms. \square

As a consequence of Proposition 4.3.2 and Definition 4.3.1, if $P \prec_h P'$, then $P \not\prec_{aph} P'$ and $P' \not\prec_{aph} P$.

For two patterns P and P' with MCTs M and M' , respectively, if $P \prec_{aph} P'$, $size(M)$ does not have to be smaller than $size(M')$. Figure 4.5 shows an example for this case where $size(MCT(P)) = 5$ and $size(MCT(P')) = 4$.

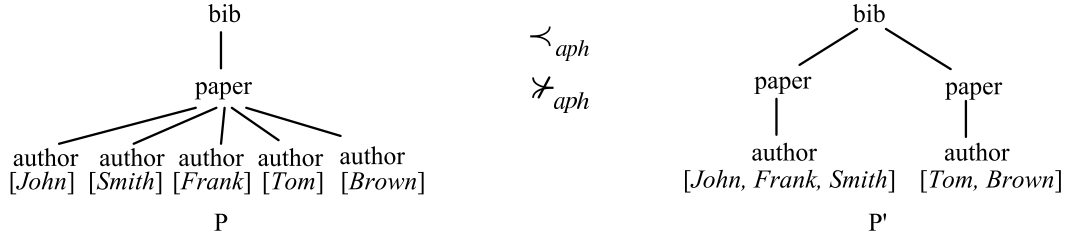


Figure 4.5 Two patterns P and P' that are related with \prec_{aph} .

Often, it is the case that an XML tree integrates data for entity types which are unrelated. For instance, the University XML tree of Figure 3.1 involves courses and seminars. Instances of these entity types are not related other than that they share the root of the XML tree as the only common ancestor. In such a context, the \prec_{aph} relation does not help us compare effectively the relevance of patterns that involve labels from different entity types with patterns that involve labels from the same entity type.

Consider, for instance, the patterns P_4 and P_5 of Figure 4.6 for the query $\{Physics, James, Harrison\}$ on Figure 3.1. These patterns are not related with respect to \prec_{aph} . However, P_4 is more relevant than P_5 since it meaningfully brings together a speaker of a seminar with the topic of the seminar. In contrast, P_5 involves both a course and a seminar and brings together the speaker of a seminar with the instructor and title of a course.

Our intuition is that if two patterns share the same root-to-leaf path (including the

annotations) the pattern P_1 whose MCT root R_1 is a descendant of the MCT root R_2 of the other pattern P_2 in this path (that is, R_1 is deeper than R_2 in the common path) more meaningfully relates the keyword instances than P_2 .

In order to enable relevance comparisons between patterns that involve unrelated parts of an XML tree, we define below a relation \prec_{pph} (called *partial_path_homomorphism* relation) on query patterns.

Definition 4.3.3 (\prec_{pph} relation). *Let P and P' be two patterns of a query Q on an XML tree T , and p be a path from the root of P to an annotated node of P . $P \prec_{pph} P'$ iff there is a path homomorphism ph of p to a path in P' such that:*

- (a) *the root of P is mapped by ph to the root of P' .*
- (b) *the root of the MCT of P is mapped by ph to a node which is a descendant (not self) of the root of the MCT of P' , and*
- (c) *$P' \not\prec_h P$ and $P' \not\prec_{aph} P$.*

Condition (c) is included for the purpose of guaranteeing the acyclicity of the relations \prec_{aph} and \prec_{pph} between two patterns.

Consider again the patterns P_4 and P_5 of Figure 4.6. As shown in the figure, there is a path homomorphism from the path `university/events/seminars/seminar/speaker[Harrison]` of P_4 to the same path of P_5 and the image of the root of the MCT of P_4 (`seminar`) under this homomorphism is a descendant of the root of the MCT of P_5 (`university`). Therefore, $P_4 \prec_{pph} P_5$. That is, the \prec_{pph} relation correctly finds P_4 to be more relevant than P_5 .

Because the roots of the MCTs of two patterns related through a \prec_{pph} relation are required to have a descendant relationship, it is easy to see that \prec_{pph} relation has the following property.

Proposition 4.3.3. *The relation \prec_{pph} is acyclic.*

Proof. Property (b) of Definition 4.3.3 requires the MCTs of two patterns related through

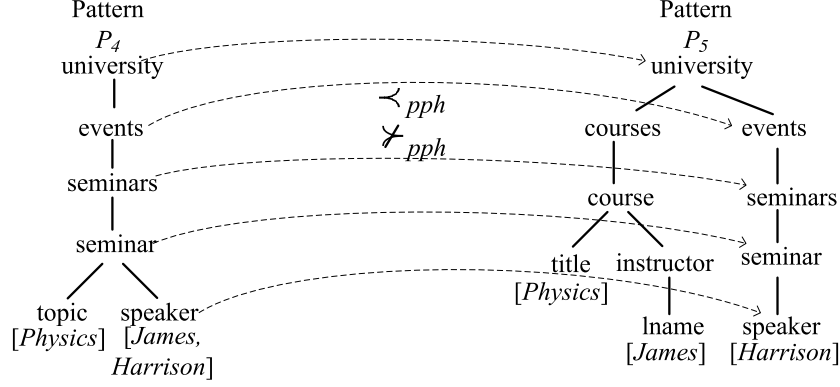


Figure 4.6 A path homomorphism from a path of pattern P_4 to a path of P_5 .

a \prec_{pph} relation to have a descendant relationship. Because of this property all of the following statements hold and hence \prec_{pph} is acyclic:

- (a) For any pattern P , clearly $P \not\prec_{pph} P$ because property (b) of Definition 4.3.3 does not hold.
- (b) For any chain of patterns P_1, \dots, P_n where $n \geq 2$, s.t. $P_1 \prec_{pph} \dots \prec_{pph} P_n$, $P_j \not\prec_{pph} P_i$ holds where $j \geq i$ because the root of MCT of P_j is an ancestor of the root of the MCT of P_i .

□

4.4 XReason Semantics

We use homomorphism relations to define filtering and ranking semantics to keyword queries called *XReason* semantics. We first define a precedence relation, \prec , on patterns which combines the three homomorphism relations¹.

Definition 4.4.1. Let P and P' be two patterns of a query Q in an XML tree T . $P \prec P'$ iff $P \prec_h P'$ or $P \prec_{aph} P'$ or $P \prec_{pph} P'$.

Based on the previous discussion one can see that with the exceptions of some

¹The homomorphism relations can also be related to the concept of preference relation in measurement theory.

impractical cases the relation \prec on the set of patterns of a query on an XML tree is acyclic.

Given the relation \prec on the set of patterns of a query Q on an XML tree T , consider a directed graph G_{\prec} such that: (a) the nodes of G_{\prec} are the patterns of Q on T , and (b) there is an edge in G_{\prec} from node P to node P' iff $P \prec P'$. Clearly, since \prec is acyclic, G_{\prec} is *acyclic*. Figure 4.7(a) shows the graph G_{\prec} for the relation \prec on the set of patterns of query $Q = \{Physics, James, Harrison\}$ on the XML tree of Figure 3.1. There are 15 such patterns and eight of them (patterns P_1 - P_8) are shown in detail in Figure 4.1. The edges are labeled by letters h , a and/or p to indicate which of the relations, respectively, \prec_h , \prec_{aph} and \prec_{pph} relate its nodes. Transitive a -edges which are not p -edges are omitted to reduce the clutter. Since G_{\prec} is acyclic, it has at least one source node (i.e., a node without incoming edges). The one of Figure 4.7 has two source nodes (pattern P_2 and P_4).

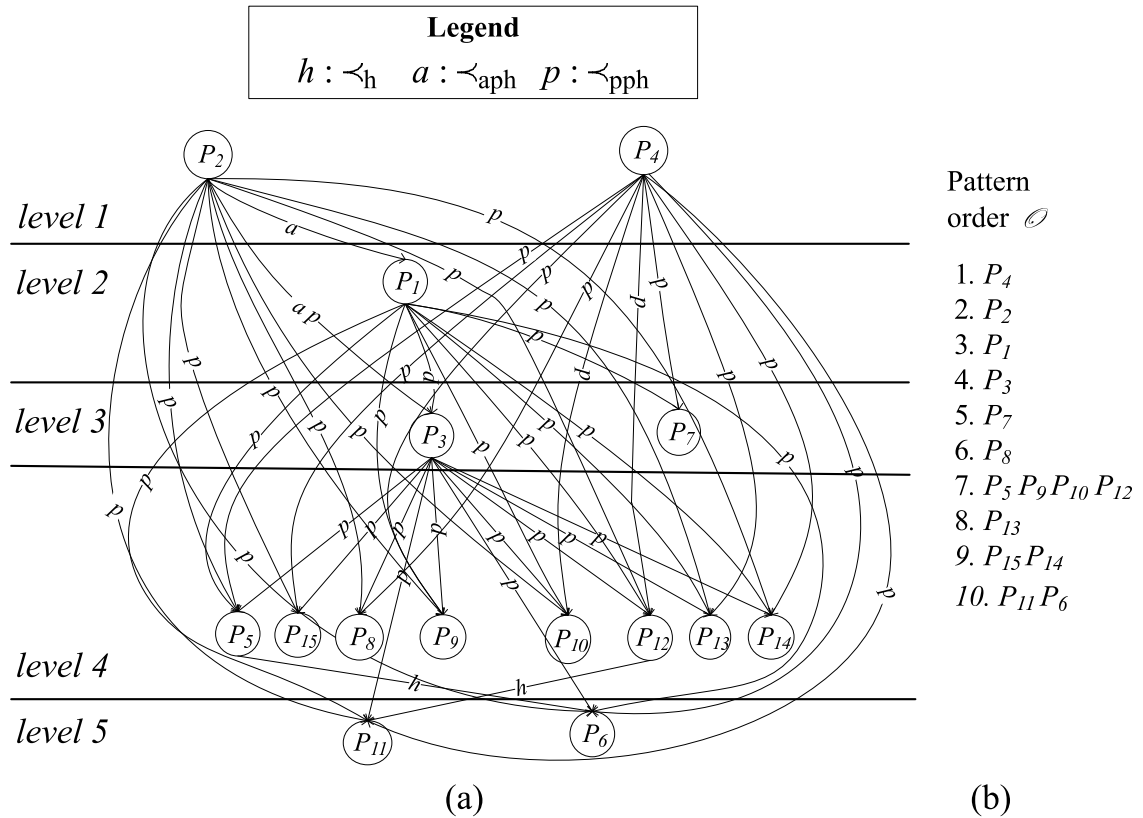


Figure 4.7 (a) The graph G_{\prec} , (b) Pattern order \mathcal{O} .

In the graph of Figure 4.7(a), observe that all the nodes (patterns) can be partitioned

in levels (separated by lines in the figure) based on their maximum distance ($GLevel$) from a source node. For instance, in level 3 there are patterns P_3 and P_7 . The patterns in one level can also be further distinguished based on the depth of their MCT root in the pattern ($MCTDepth$) and the size of their MCT ($MCTSize$). We create an order \mathcal{O} for the patterns in G_{\prec} which ranks them in: (a) ascending order of $GLevel$, (b) descending order of $MCTDepth$, and (c) ascending order of $MCTSize$. Note that two patterns might be placed at the same rank in \mathcal{O} . The order \mathcal{O} does not distinguish between these patterns. In our running example, one can see that the fifteen patterns of Figure 4.7(a) are ordered with respect to \mathcal{O} as shown in Figure 4.7(b).

Definition 4.4.2 (Ranking $XReason$ semantics). *According to the ranking $XReason$ semantics an answer of a query Q on an XML tree T is a list of the ITs of Q on T ranked in an order which complies with the order \mathcal{O} of their patterns.*

In our running example, we have 15 patterns and 16 ITs. An ordering of these ITs which complies with the order \mathcal{O} of patterns shown in Figure 4.7(b) is an answer of the query $Q = \{Physics, James, Harrison\}$ on the XML tree of Figure 3.1.

We use the patterns at the top- k levels of G_{\prec} to define filtering semantics.

Definition 4.4.3 (Filtering $XReason$ semantics). *According to the filtering $XReason$ semantics the answer of Q on T is the set of ITs of the patterns of Q on T with the smallest k $GLevel$ values.*

Parameter k is user defined. Usually, k is chosen to be equal to one (i.e., we choose the top level patterns) in which case the answer of Q on T is the set of ITs whose patterns are source nodes in the G_{\prec} graph.

Based on the previous definition and for $k=1$, the answer of query $Q = \{Physics, James, Harrison\}$ on the XML tree of Figure 3.1 is the set of ITs which comply with patterns P_2 or P_4 (the two source nodes in graph G_{\prec} of Figure 4.7(a)). There are only two ITs which comply with these patterns—one for each pattern: the IT of P_2 with leaf nodes

5, 7 and 11 whose MCT is rooted at `course` and the IT of P_4 with leaf nodes 24 and 25 whose MCT is rooted at `seminar`.

It is interesting to note that according to *XReason* semantics, an IT might more meaningfully relate its keyword instances than another IT even though these ITs do not have any location proximity that is, they do not share any node and their LCAs are not in ancestor-descendant-or-self relationship. This feature departs from previous traditional filtering semantics (e.g., *ELCA* [32, 89], *SLCA* [18, 35, 88], *MLCA* [48]) where the location proximity is required in order to privilege one IT over another. In the next section we analyze the filtering *XReason* semantics in relation to previous semantics.

4.5 Analysis of *XReason*

In this section, we compare the *XReason* filtering semantics with different previous filtering semantics from the literature. For determining the query answer with *XReason*, only ITs of the top level patterns in G_{\prec} are retained.

Many approaches to keyword search in the literature return LCA nodes as answers to keyword queries. In this dissertation, the answer of a keyword query is a set of ITs (see Definition 3.2.2). The ITs more precisely capture the subtleties of the different semantics than the LCAs since the same LCA can be the root of multiple MCTs of different ITs for a query. In order to set up a common ground for comparison, we define the previous approaches in terms of ITs. We go through these approaches with an example but we provide before a summary of their formal definitions. Let $Q = \{k_1, k_2, \dots, k_n\}$ denote a keyword query, and T denote an XML tree. Let also $LCAs_{set}(Q, T)$ be the set of LCAs of Q on T , and $IT_{set}(Q, T)$ be the set of ITs of the instances of Q on T . Table 4.1 provides formal definitions of the previous semantics in terms of ITs.

We first examine relationships between semantics in terms of result containment. This is important to understand the overall view of different XML keyword search semantics. Then, we show cases where the other approaches miss meaningful answers or return

meaningless answers while *XReason* does not. This demonstration proves also that *XReason* is different than all the previous approaches.

4.5.1 Containment Relationships between Keyword Search Semantics

If the answer of a query according to semantics A is a subset of the answer of this query according to semantics B for any query and on any XML tree, we say that B contains A and we write $A \subseteq B$. Containment relationships between the different approaches based on the definitions of Table 4.1 are shown in Figure 4.8. Containment relationships between some filtering approaches are also provided in [58]. However, the semantics defined in [58] are based on LCAs while the semantics we defined in this dissertation are based on ITs.

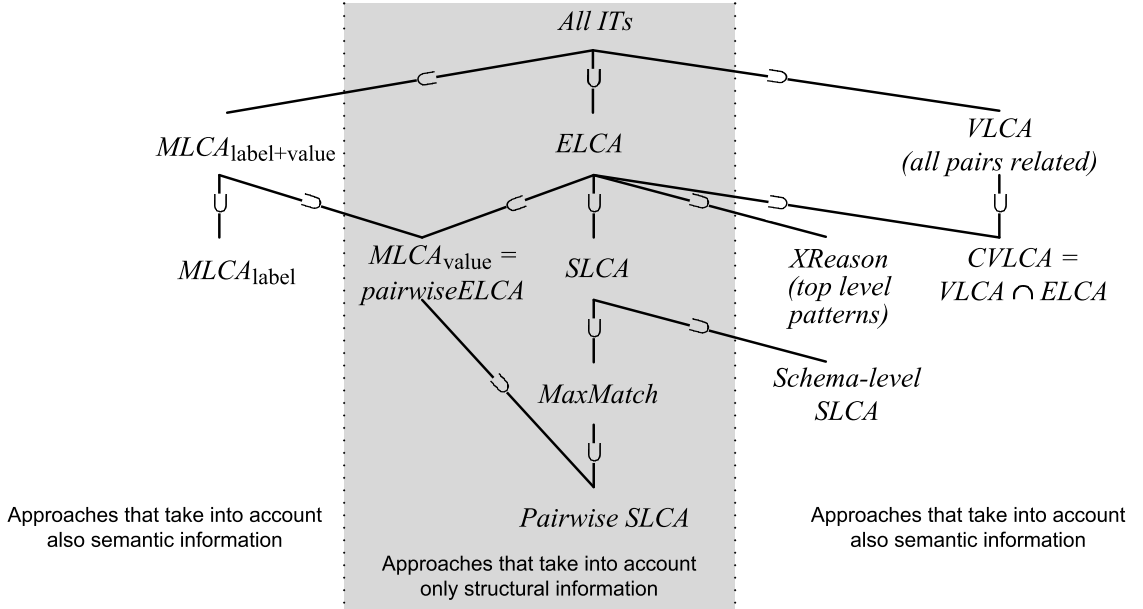


Figure 4.8 Containment relationships between different filtering semantics.

In the following, we provide proofs for the containment relationships depicted in Figure 4.8.

Proposition 4.5.1. $SLCA \subseteq ELCA$.

Proof. Let $t \in SLCAset(Q, T)$. Then, $\nexists t' \in ITset(Q, T)$ s.t. $root(MCT(t'))$ is a descendant of $root(MCT(t))$. Therefore, $t \in ELCAset(Q, T)$. \square

Table 4.1 Formal Definitions of Different Previous Filtering Semantics in Terms of ITs

Semantics	Definition of the answer of Q on XML tree T
<i>SLCA</i> [18, 35, 88]	$\{t \mid t \in ITset(Q, T), n = root(MCT(t)), \text{ and } \nexists t', n' (t' \in ITset(Q, T), n' = root(MCT(t')) \text{ and } n' > n)\}$
<i>ELCA</i> [32, 89]	$\{t \mid t \in ITset(Q, T), n = root(MCT(t)), \text{ and } \nexists n' (n' \text{ is a node in } MCT(t), n \neq n' \text{ and } n' \in LCAsset(Q, T))\}$
<i>VLCA</i> [21, 43] (all pairs related)	$\{t \mid t \in ITset(Q, T) \text{ and } \forall \text{ pair of annotated nodes } n_i, n_j \text{ in } t, \nexists \text{ distinct nodes } n_k, n_l \text{ in the path between } n_i \text{ and } n_j \text{ s.t. } label(n_k) = label(n_l) \text{ unless } n_k = n_i \text{ and } n_l = n_j\}$
<i>CVLCA</i> [43]	$VLCAset(Q, T) \cap ELCAset(Q, T)$ where $VLCAset(Q, T)$ (resp. $ELCAset(Q, T)$) is the answer of Q on T according to <i>VLCA</i> (resp. <i>ELCA</i>) semantics
$MLCA_{label}$ [48, 79]	$\{t \mid t \in ITset(Q, T) \text{ and } \forall \text{ pair of annotated nodes } n_i, n_j \text{ in } MCT(t) \text{ of two distinct keywords, } \nexists \text{ node } n \text{ in } T \text{ s.t. } label(n) = label(n_j) \text{ and } LCA(n_i, n) > LCA(n_i, n_j) \text{ in } T\}$
$MLCA_{value}$ [48] = <i>pairwiseELCA</i>	$\{t \mid t \in ITset(Q, T) \text{ and } \forall \text{ pair of nodes } n_i \text{ and } n_j \text{ in } MCT(t) \text{ annotated by the keywords } k_i \text{ and } k_j, \text{ respectively, } \nexists \text{ instance } n'_j \text{ of } k_j \text{ in } T \text{ s.t. } LCA(n_i, n'_j) > LCA(n_i, n_j) \text{ in } T\}$
$MLCA_{label+value}$ [48]	$\{t \mid t \in ITset(Q, T) \text{ and } \forall \text{ pair of nodes } n_i \text{ and } n_j \text{ in } MCT(t) \text{ annotated by the keywords respectively, } k_i \text{ and } k_j, \nexists \text{ instance } n'_j \text{ of } k_j \text{ in } T \text{ s.t. } label(n'_j) = label(n_j) \text{ and } LCA(n_i, n'_j) > LCA(n_i, n_j) \text{ in } T\}$
<i>MaxMatch</i> [55]	$\{t \mid t \in SLCAset(Q, T) \text{ and } \forall \text{ node } n \text{ in } MCT(t), \nexists \text{ sibling node } n' \text{ of } n \text{ in } T \text{ s.t. the set of keywords occurring in the subtree rooted at } n \text{ in } T \text{ is a proper subset of the set of keywords occurring in the subtree rooted at } n' \text{ in } T\}$
<i>pairwiseSLCA</i>	$\{t \mid t \in ITset(Q, T) \text{ and } \forall \text{ pair of nodes } n_i \text{ and } n_j \text{ annotated by keywords } k_i \text{ and } k_j, \text{ respectively in } MCT(t), \nexists \text{ instances } n'_i \text{ and } n'_j \text{ of the same keywords in } T \text{ s.t. } LCA(n'_i, n'_j) > LCA(n_i, n_j) \text{ in } T\}$
<i>Schema-level SLCA</i> [42]	$\{t \mid t \in ITset(Q, T) \text{ and } \nexists t' \in ITset(Q, T) \text{ s.t. the root-to-LCA label path of } t \text{ is a proper prefix of the root-to-LCA label path of } t'\}$

Proposition 4.5.2. $MaxMatch \subseteq SLCA$.

Proof. By the definition of *MaxMatch* (see Table 1), if $t \in MaxMatchset(Q, T)$, $t \in SLCAset(Q, T)$. □

Proposition 4.5.3. $pairwiseSLCA \subseteq MaxMatch$.

Proof. Let $t \in ITset(Q, T)$ and $t \notin MaxMatchset(Q, T)$. Then, there exist two sibling nodes n and n' in T , n appearing in $MCT(t)$, s.t., the set S of keywords occurring in the subtree rooted at n in T is a proper subset of the set S' of keywords occurring in the subtree rooted

at n' in T . Thus, there exist two keywords $k_1, k_2 \in Q$, s.t., $k_1 \in S \cap S'$ and $k_2 \in S' - S$. Let n_1 and n'_1 be two instances of k_1 in the subtrees rooted at n and n' , respectively, s.t. k_1 annotates n_1 , let also n_2 be the node annotated by k_2 in t , and n'_2 be an instance of k_2 in the subtree rooted at n' . Then, $LCA(n_1, n_2) < LCA(n'_1, n'_2)$, i.e., $t \notin pairwiseSLCAset(Q, T)$. We conclude that $pairwiseSLCA \subseteq MaxMatch$. \square

Proposition 4.5.4. $CVLCA \subseteq VLCA$.

Proof. By the definition of CVLCA (see Table 1), if $t \in CVLCAset(Q, T)$, $t \in VLCAset(Q, T)$. \square

Proposition 4.5.5. $CVLCA \subseteq ELCA$.

Proof. By the definition of CVLCA (see Table 1), if $t \in CVLCAset(Q, T)$, $t \in ELCAset(Q, T)$. \square

Proposition 4.5.6. $MLCA_{value} \subseteq ELCA$.

Proof. Let $t \in ITset(Q, T)$ and $t \notin ELCAset(Q, T)$. Then, $\exists t' \in ITset(Q, T)$, s.t., $root(MCT(t'))$ occurs in $MCT(t)$ and $root(MCT(t')) \neq root(MCT(t))$. Thus, there exist two keywords k_1 and k_2 in Q s.t., (a) k_1 annotates a node n_1 in t , and n_1 is a descendant-or-self of $root(MCT(t'))$, (b) k_2 annotates a node n_2 in t and a node n'_2 in t' , and (c) n_2 is not in t' . Then, $LCA(n_1, n_2) < LCA(n_1, n'_2)$, i.e., $t \notin MLCA_{value}set(Q, T)$. We conclude that $MLCA_{value} \subseteq ELCA$. \square

Proposition 4.5.7. $XReason \subseteq ELCA$.

Proof. Let $t \in ITset(Q, T)$ and $t \notin ELCAset(Q, T)$. Then, $\exists t' \in ITset(Q, T)$, s.t., $root(MCT(t'))$ occurs in $MCT(t)$ and $root(MCT(t')) \neq root(MCT(t))$. Therefore, t and t' share a common root-to-annotated-node path and $root(MCT(t)) < root(MCT(t'))$. As a consequence, the pattern P'_t of t' precedes the pattern P_t of t with respect to \prec_{pph} ($P'_t \prec_{pph} P_t$) and thus, t (and all the ITs of P_t) is eliminated from the answer of Q on T according to $XReason$. Thus, $t \notin XReasonset(Q, T)$. We conclude that $XReason \subseteq ELCA$. \square

Proposition 4.5.8. *Schema-level SLCA \subseteq SLCA.*

Proof. Let $t \in ITset(Q, T)$ and $t \notin SLCAset(Q, T)$. Then, $\exists t' \in ITset(Q, T)$, s.t., $root(MCT(t)) < MCT(root(t'))$. Thus, the root-to-LCA label path of t is a proper prefix of the root-to-LCA label path of t' and $t \notin Schema-levelSLCAset(Q, T)$. We conclude that *Schema-level SLCA \subseteq SLCA*. \square

Proposition 4.5.9. *$MLCA_{label} \subseteq MLCA_{label+value}$.*

Proof. By the definition of $MLCA_{label}$ (see Table 1), if $t \in MLCA_{label}set(Q, T)$, $t \in MLCA_{label+value}set(Q, T)$. \square

Proposition 4.5.10. *$MLCA_{value} \subseteq MLCA_{label+value}$.*

Proof. By the definition of $MLCA_{value}$ (see Table 1), if $t \in MLCA_{value}set(Q, T)$, $t \in MLCA_{label+value}set(Q, T)$. \square

Proposition 4.5.11. *$pairwiseSLCA \subseteq MLCA_{value}$.*

Proof. Let $t \in ITset(Q, T)$ and $t \notin MLCA_{value}set(Q, T)$. Then, for a pair of nodes n_i and n_j in $MCT(T)$ annotated by keywords k_i and k_j , respectively, there exists an instance n'_j of k_j in T s.t. $LCA(n_i, n_j) < LCA(n_i, n'_j)$ in T . Then, by the definition of *pairwiseSLCA* (see Table 1), $t \notin pairwiseSLCA$. We conclude that *$pairwiseSLCA \subseteq MLCA_{value}$* . \square

4.5.2 Comparison of Filtering Semantics

Consider the query, $Q = \{Physics, James, Harrison\}$ on the XML tree of Figure 4.9. With this query, the user requests information about a course or seminar on *physics* which is offered by *James Harrison* or by *James* and *Harrison*. The relevant ITs to Q are, $IT_1 = \{(Physics, 4), (James, 6), (Harrison, 10)\}$, $IT_2 = \{(Physics, 13), (James, 16), (Harrison, 15)\}$ and $IT_3 = \{(Physics, 40), (James, 41), (Harrison, 41)\}$. The LCAs of their keyword instances are (3, course), (12, course) and (39, seminar) and are labeled l_1 , l_2 and l_3 , respectively, in Figure 4.9.

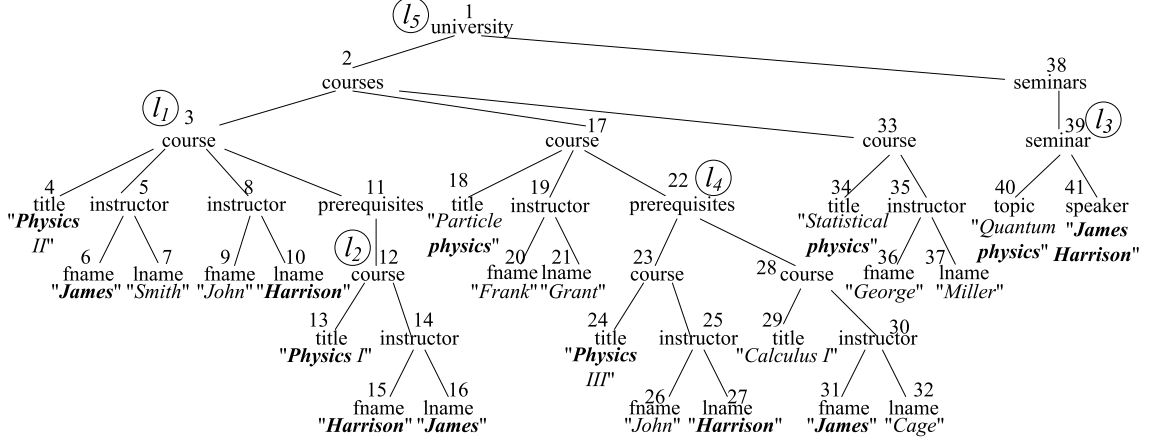


Figure 4.9 An XML tree.

SLCA [18, 35, 88] eliminates an IT whose LCA is an ancestor of another IT's LCA. Therefore, it fails to return IT_1 because there is another IT, IT_2 , whose LCA (node 12) is a descendant of IT_1 's LCA (node 3). Missing correct results reduces the recall of the approach. Moreover, *SLCA* returns $IT_4 = \{(Physics, 24), (James, 31), (Harrison, 27)\}$ with LCA (22, prerequisites) which is an irrelevant result since it involves two different courses. Irrelevant results affect the precision of *SLCA*.

ELCA [32, 89] returns ITs whose MCT does not comprise an LCA of the query keywords other than the MCT root. For this reason IT_2 is a result for *ELCA*. IT_1 is also a result of *ELCA* even though it is not a result of *SLCA* since its MCT does not comprise any LCA of the query keywords other than (3, course). This is an improvement of *ELCA* over *SLCA*. However, it fails to eliminate wrong results. For instance, *ELCA* returns IT_4 as there does not exist another IT with a descendant LCA. In some cases, *ELCA* has no way of eliminating an irrelevant answer. This weakness affects its precision.

The *VLCA* semantics [21, 43] takes into account also the labels of the nodes in the XML tree. It eliminates an IT if for some pair of annotated nodes, the path between them comprises two distinct nodes with the same label and at least one of them is an internal node in the path. *VLCA* fails to return IT_1 because nodes 9 and 10 have the same label, instructor. Even though, *VLCA* is intuitive in specific cases, it is prone to

missing relevant results in the general case. In addition, it is not able to eliminate irrelevant results when they do not contain duplicate labels. For instance, it fails to eliminate $IT_5 = \{(Physics, 34), (James, 41), (Harrison, 41)\}$ (whose LCA (1, university) is labeled as l_5 in Figure 4.9) which is irrelevant as it links information from a course and a seminar through the root of the XML tree.

CVLCA [43] returns an answer which is a subset of *VLCAset* by enforcing a stricter rule. If an IT is in the *CVLCAset*: (a) the IT is in *VLCAset*, and (b) its MCT does not comprise an LCA of the query keyword instances other than the MCT root. Condition (b) amounts to forcing the IT to be part of the *ELCAset*. Therefore, an IT is returned by *CVLCA* if and only if it is in the intersection of *VLCAset* and *ELCAset*. Since *CVLCA* is a subset of *VLCAset* and *ELCAset*, it inherits the recall problems from both approaches. For instance, it fails to return IT_1 which as mentioned above does not belong to *VLCAset*.

The *MLCA* semantics [48] requires all keyword instances in a result IT to be pairwise meaningfully related. However, the original definition of the meaningfulness relationship in [48] does not distinguish between a keyword matching the content of a node and the label of a node. Therefore, there are three ways to interpret this relationship in the XML keyword search context, which lead to three alternative definitions for *MLCA* semantics, named here $MLCA_{label}$, $MLCA_{value}$ and $MLCA_{label+value}$. $MLCA_{label}$ and $MLCA_{label+value}$ take the labels of the nodes in the XML tree into account. According to $MLCA_{label}$ two keyword instances n_i and n_j are meaningfully related if there exists no other node n_k in T , with the same label as n_i , such that $LCA(n_j, n_k) > LCA(n_i, n_j)$ in T . In the example of Figure 4.9, $MLCA_{label}$ misses IT_1 as it does not meaningfully relate nodes (6, fname) and (10, lname). Also, it fails to eliminate IT_5 as all pairs of keyword instances in IT_5 are meaningfully related. $MLCA_{value}$ is purely structural that is, it is independent of the node labels in the XML tree. According to $MLCA_{value}$, two instances n_i and n_j of the keywords k_i and k_j , respectively, are meaningfully related if there exists no other instance n_k of k_i in T such that $LCA(n_j, n_k) > LCA(n_i, n_j)$ in T . One can see that $MLCA_{value}$ is

equivalent to *pairwiseELCA*. The *pairwiseELCA* semantics returns an IT if the LCA of any two annotated instances of two distinct keywords in it is also an ELCA of these two keywords in T . $MLCA_{value}$ fails to eliminate the irrelevant result IT_4 . Finally, according to $MLCA_{label+value}$, two instances n_i and n_j of the keywords k_i and k_j , respectively, are meaningfully related if there exists no other instance n_k of k_i in T such that $label(n_k) = label(n_i)$ and $LCA(n_j, n_k) > LCA(n_i, n_j)$ in T . In other words, $MLCA_{label+value}$ defines the meaningfulness relationship by imposing the conditions of both $MLCA_{label}$ and $MLCA_{value}$. Since $MLCA_{label+value} = MLCA_{label} \cup MLCA_{value}$, it inherits the precision problems of $MLCA_{label}$ and $MLCA_{value}$.

MaxMatch [55] refines *SLCA* by excluding ITs accepted by *SLCA* when they involve “disqualified” keyword instances in the XML tree. *MaxMatch* is a purely structural semantics since it uses structural criteria to identify disqualified nodes. The formal definition is shown in Table 4.3. The query answer of *MaxMatch* is a subset of that of *SLCA*. One can see that *MaxMatch* is less restrictive than *pairwiseSLCA* which accepts an IT if the LCA of any two annotated instances of two distinct keywords in it is also an *SLCA* of these two keywords in the XML tree. Since, *MaxMatch* is contained in *SLCA*, it inherits the bad recall of *SLCA*. For instance, in our running example, it misses the relevant IT_1 . Moreover, it fails to eliminate IT_4 which is irrelevant since, as mentioned earlier, it groups together two distinct courses.

Schema-level SLCA [42] also refines *SLCA* by excluding ITs accepted by *SLCA* leveraging both structural and semantic information. It excludes an IT accepted by *SLCA* if its root-to-LCA label path is a proper prefix of that of another IT. By definition, *Schema-level SLCA* is contained in *SLCA* and as such it demonstrates the poor recall performance of *SLCA* but it can even worsen it by excluding relevant ITs retained by *SLCA*. Figure 1.1 (and not Figure 4.9) shows such an example for query $\{Physics, James, Harrison\}$ where the relevant IT whose LCA is node 3 is rejected even though it is accepted by *SLCA*.

Our approach, *XReason*, successfully returns all relevant ITs, and eliminates the

irrelevant ones. Results IT_1 , IT_2 and IT_3 conform to top level patterns in the G_{\prec} graph and they are retained. The irrelevant IT_4 is eliminated because the pattern P_1 of IT_1 precedes the pattern P_4 of IT_4 with respect to the \prec_{aph} relation ($P_1 \prec_{aph} P_4$). The pattern P_5 of IT_5 is preceded by the pattern P_3 of IT_3 with respect to the \prec_{pph} relation ($P_3 \prec_{pph} P_5$). Thus, the IT_5 is eliminated. All other irrelevant ITs are eliminated using \prec_{pph} .

The \prec_{pph} relation is particularly useful for eliminating irrelevant ITs that link keyword instances through the root of the XML tree and are almost in all cases meaningless. In Section 4.6.3 we present a heuristic extension of our algorithms which eliminates patterns connecting keyword instances through the root of the XML tree in order to improve performance.

4.6 Algorithms

Our implementation of *XReason* comprises two components. The first one uses a stack-based algorithm to generate the query patterns and their associated ITs. The second one constructs the precedence graph G_{\prec} based on the \prec_h , \prec_{aph} and \prec_{pph} relations and ranks the patterns and their respective ITs. We have maintained these processes separate in our system in order to be able to modify the semantics of query answers (pattern graph construction and pattern ranking) but also to include additional metrics in producing a ranking for the patterns, if desired. We also present in this section an improvement of the pattern generation algorithm which avoids generating patterns that are not meaningful and would be placed in low ranks based on the homomorphism relations, thereby substantially improving the performance of the system. Aggeliki Dimitriou from NTUA also contributed to the design of the algorithms.

4.6.1 Pattern Generation

The algorithm that extracts the query patterns is named *PatternStack* and is outlined in Algorithm 1. *PatternStack* takes as input the keyword query and the inverted lists of the keyword instances (XML tree nodes) for the query keywords. It returns the patterns of the query answers associated with their ITs. The helper functions *pop*, *push*, *constructNewPatterns* and *extendToParent* of the *PatternStack* algorithm are outlined in Algorithm 2.

PatternStack does not wait to extract patterns until after all the result ITs of the query are computed. Instead, it follows a *dynamic* approach: it incrementally computes the patterns on the fly while computing the result ITs and links the ITs to their respective patterns. A notable feature of *PatternStack* is that it does not require auxiliary structures for computing patterns and ITs.

Every entry in an input inverted list consists of the Dewey code of a keyword instance and the label path from the root of the XML tree to this instance. In order to reduce space consumption, the labels are numerically encoded. The inverted lists are produced with a single pass of the XML document. Patterns in *PatternStack* are tree structures. Every node in a pattern is associated with the set of keywords which annotate this node and its descendants in the pattern. This keyword set is encoded as a bitmap over the list of all keywords. During the pattern construction phase, *PatternStack* constructs patterns which do not involve all the keywords and are called partial patterns. These patterns are represented similarly to complete patterns. Partial patterns are progressively augmented into complete patterns. Partial and complete patterns are identified by ids (*pids*). Figure 4.10 shows (among other concepts which will be explained later) how patterns are represented by *PatternStack*.

PatternStack processes nodes from the keyword inverted lists in document order. The algorithm uses a stack to progressively construct the patterns of a query on an XML tree in a bottom-up way. Each stack entry corresponds to a node of the XML tree and is associated with the set of all the MCTs that involve its descendant keyword instances.

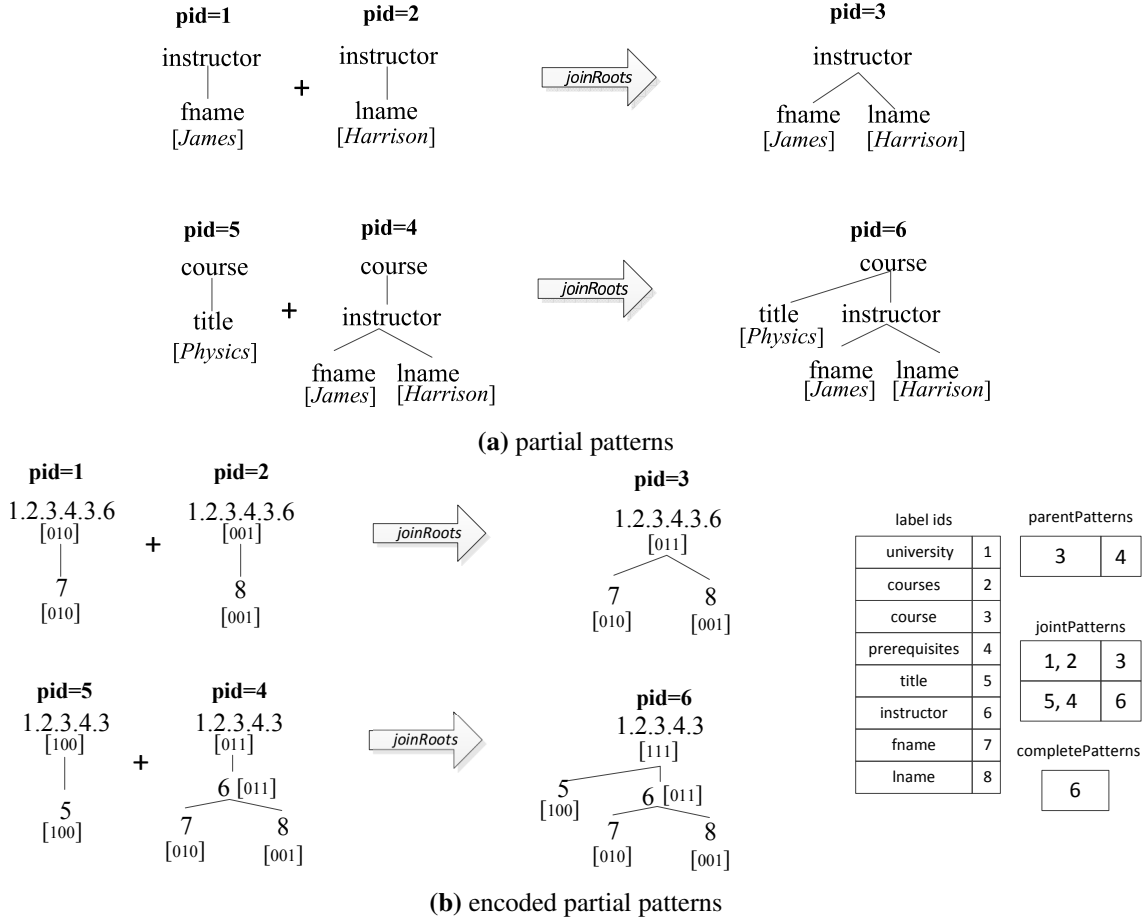


Figure 4.10 PatternStack pattern encoding and combination for $Q = \{Physics, James, Harrison\}$ on the data tree of Figure 4.9.

These MCTs can be complete (i.e., they involve instances of all the query keywords) or partial (i.e., they involve instances of a proper subset of the query keywords). They are represented in the stack entry by their corresponding partial or complete patterns. For each pattern only an id (*pid*) is stored.

In order for a node n to be pushed into a stack, the top stack node should be the parent of n . This is guaranteed by appropriate pops of non-ancestor nodes and pushes of all the ancestors of n (Algorithm 1, lines 8-12). For each new node, a partial pattern MCT is constructed (Algorithm 2, line 13). If this pattern has been constructed before, it appears in the list *patterns* and its *pid* is known. Otherwise, it is added to *patterns* and gets a new *pid* (Algorithm 2, line 14). Then, it is combined with all the existing patterns of the top stack

node to build new patterns. The resulting new pattern set is unioned with the old pattern set of the top stack node (Algorithm 2, lines 15-16).

Algorithm 1: PatternStack algorithm.

```

1 PatternStack(IN:  $k_1, \dots, k_n$ : keyword query, invL: inverted lists)
2   pattern[] patterns /* Array of patterns. The array indexes
   are pattern ids */
3   int[] completePatterns /* Array of complete patterns' ids */
4   int[][] jointPatterns /* Mappings from pairs of patterns to
   their joint patterns */
5   int[][] parentPatterns /* Mappings from patterns to their
   parent patterns */
6   s = new Stack()
7   while currentNode = getNextNodeFromInvertedLists() do
8     while s.topNode is not ancestor of currentNode do
9       pop(s)
10    while s.topNode is not parent of currentNode do
11      push(s, ancestor of currentNode at
12        s.topNode.depth+1, "")
13    push(s, currentNode, keyword)
14  while s is not empty do
15    pop(s)

```

During a pop action, all complete patterns are removed from the top stack entry (Algorithm 2, lines 3-5). The Dewey code of the top stack node is associated with all complete patterns, since it is the root of the MCTs of the ITs corresponding to these complete patterns (Algorithm 2, line 6). The remaining (partial) patterns are extended to the parent of the top node (Algorithm 2, line 8 and 32-41). Note that only the root node of a pattern MCT is associated with a path of label ids, while the rest of the nodes are associated only with their own label id (Figure 4.10b). This scheme is implemented in lines 37-38 of Algorithm 2. The annotation of the former root is also propagated to the new one (Algorithm 2, line 37). The top entry is popped (Algorithm 2, line 9) and its extended patterns are combined with the patterns of the parent stack entry to produce new ones (Algorithm 2, line 10). The patterns are combined through an operation called *joinRoots* (Algorithm 2,

Algorithm 2: PatternStack algorithm.

```

1  pop(Stack s)
2      for tempId = s.top.patterns.next() do
3          if temp is complete then
4              s.top.removePatternId(tempId)
5              completePatterns.add(tempId)
6              patterns[tempId].addLCA(s.top.dewey())
7          else
8              childPatterns.add(extendToParent(tempId))
9      s.pop()
10     newPatternIds = constructNewPatterns(s.top.patterns, childPatterns)
11     s.top.addPatternIds(newPatternIds)

12 push(Stack s, Node n, String keyword)
13     newP = new Pattern(n.labelId, flags.set(id(keyword)))
14     newPid = addToPatternsIfNotExists(newP)
15     newPatternIds = constructNewPatterns(s.top.patterns, array(newPatternId))
16     s.top.unionPatternIds(newPatternIds)

17 int[] constructNewPatterns(int[] currentPatternIdsA, int[] currentPatternIdsB)
18     foreach currentPatternIdsA as idA do
19         foreach currentPatternIdsB as idB do
20             if patterns[idA].keywordFlags AND patterns[idB].keywordFlags == 0
21                 then
22                     if jointPatterns[min(idA, idB), max(idA, idB)] is set then
23                         newPatterns.add(jointPatterns[min(idA, idB),
24                             max(idA, idB)])
25                     else
26                         newP = new Pattern(joinRoots(patterns[idA], patterns[idB]))
27                         newP.keywordFlags = patterns[idA].kwFlags OR
28                             patterns[idB].kwFlags
29                         newPid = addToPatternsIfNotExists(newP)
30                         newPatterns.add(newPid)
31                         jointPatterns[min(idA, idB), max(idA, idB)] = newPid
32     return newPatterns

33 int extendToParent(int childPid)
34     if parentPatterns[childPid] is set then
35         return parentPatterns[childPid]
36     else
37         childP = copyOf(patterns[childPid])
38         parentP = new Pattern(parentLabel(childP.label), childP.kwFlags)
39         childP.label = tail(childP.label)
40         parentP.addChild(childP)
41         parentPid = patterns.add(parentP)
42         parentPatterns[childPid] = parentPid

```

line 25) which simply merges their roots. The resulting pattern is called *jointPattern*. Figure 4.10 shows examples of *joinRoots* operations on partial patterns. Finally, the extended patterns together with newly combined ones are added to the parent stack entry.

Algorithm PatternStack considers any alternative way of combining partial patterns into a complete pattern only once. It stores the patterns in the *patterns* array and keeps only their pids in the stack entries. Every time two patterns are combined to produce a new one, the pids of the combined patterns and that of the *jointPattern* are kept in the table *jointPatterns* (Algorithm 1, line 4). Another table (*parentPatterns* in line 5 of Algorithm 1) associates each pattern MCT *childP* with the pattern *parentP* produced when *childP* is augmented with an edge to the parent node of its root (Algorithm 2, lines 32-41). These two tables are consulted before any pattern is constructed (Algorithm 2, lines 21 and 33) in order to avoid extending a pattern which has already been extended or combining two patterns that have already been combined. Function *addToPatternsIfNotExists()* (Algorithm 2, lines 14 and 28) checks if a newly constructed pattern has been constructed before. This function compares the new pattern only with stored patterns having: (a) the same root label path and (b) the same root annotation. The comparison is performed exploiting a unique string representation of each pattern, which uses the label encodings and the annotation bitmaps of the pattern nodes. Function *checkIfExistsOrAddToPatterns()* returns the existing pid or the new pid assigned to the new pattern added to the array *patterns*.

The example of Figure 4.10 shows how PatternStack combines and extends patterns. Patterns P1 and P2 are combined to construct P3. Pattern P3 is extended to produce P4. Pattern P4 combined with P5 produces P6. The figure shows also the encoded labels and label paths as well as the bitmap keyword annotations.

The previous discussion in this section proves the following proposition.

Proposition 4.6.1. *The PatternStack algorithm correctly computes all the patterns of a keyword query on the set of inverted lists of an XML tree.*

Let d be the depth of the data tree and $|S|$ be the total number of nodes in the

inverted lists. Each insertion of a node from the inverted lists may require at most d pops from and d pushes onto the stack. Let p be the number of partial patterns a stack entry can contain. When a node is pushed onto the stack, it may be combined with at most p patterns of the parent stack entry. This takes $O(p)$ time. When a node is popped from the stack, all its partial patterns are extended with an edge to their parent node (the new top entry in the stack). Assuming all the patterns of the popped node are partial, this takes $O(p)$. They are also combined with the partial patterns of the parent node to produce new patterns which takes $O(p^2)$. The whole process takes $O(|S|dp^2)$.

4.6.2 Graph Construction and Ranking

The second component of our system constructs the precedence graph G_{\prec} and ranks the patterns. This is implemented by algorithm *PatternGraph*. Algorithm *PatternGraph* takes as input the patterns produced by *PatternStack* and incrementally constructs G_{\prec} by checking for the existence of the homomorphism relations \prec_h , \prec_{aph} and \prec_{pph} between each new pattern and the patterns in G_{\prec} . Then, it uses the graph to rank the patterns as described in Section 4.4. There are two sources of complexity in this process: (a) checking for the existence of the \prec_h , \prec_{aph} and \prec_{pph} relations between two patterns (which involves checking for the existence of the different types of homomorphisms), and (b) applying these checks to a large number of pairs of patterns.

In order to deal with (a), *PatternGraph* exploits the properties of the \prec_h , \prec_{aph} and \prec_{pph} relations as this is shown by the next four propositions.

Property 1. *Let P and P' be two patterns of a query on an XML tree. If the number of root-to-leaf paths of P is greater than the number of root-to-leaf paths of P' , then $P \not\prec_h P'$.*

The proof of Property 1 is derived from the fact that the annotation (subset of the keywords) of an annotated node n' in P' which is mapped by a homomorphism to a node n in P should be a subset of the annotation of n . Since the numbers of annotating keywords

in two patterns are equal, the number of root to leaf paths in P cannot exceed that of P' .

Property 2. *Let P and P' be two patterns of a query on an XML tree. If the height of the MCT of P is greater than that of the MCT of P' , then $P \not\prec_{aph} P'$.*

Clearly, if the height of the MCT of P is greater than that of the MCT of P' , the longest path in the MCT of P cannot be mapped by a path homomorphism to any path in the MCT of P' , and therefore, $P \not\prec_{aph} P'$.

Property 3. *Let P and P' be two patterns of a query on an XML tree. If the labels of nodes annotated by the same keyword in P and P' are not the same, then $P \not\prec_h P'$, $P \not\prec_{aph} P'$, $P' \not\prec_h P$ and $P' \not\prec_{aph} P$.*

Indeed, it is easy to see that there is no homomorphism from the MCT of P to that of P' and path homomorphisms from the paths of the MCT of P to those of the MCT of P' if the labels of their nodes annotated by the same keywords are not the same.

Property 4. *Let P and P' be two patterns of a query on an XML tree. $P \not\prec_{pph} P'$, if one of the following conditions does not hold: (a) the LCA depth of P' is smaller than the LCA depth of P , (b) the root to LCA path of P' is a prefix of the root to LCA path of P , and (c) the maximum length MCT path in P' is longer than the minimum length MCT path in P .*

The proof of Property 4 can be directly derived from the definition of path homomorphism and the \prec_{pph} relation.

Based on these properties, *PatternGraph* avoids initiating in most cases the checking for the existence of homomorphisms or path homomorphisms between patterns by storing numeric information (e.g., the number of root to leaf paths, the height of the MCT, the LCA depth) with every pattern at construction time.

To support checking for path homomorphisms when this is needed, the MCT root-to-annotated-node paths, including the annotations, are represented as strings. Then, checking for the existence of path homomorphisms reduces to string matching starting with the annotations.

In order to address the complexity related to the large number of checks between patterns (which are needed in order to determine the existence of edges between nodes), *PatternGraph* avoids constructing some paths in the graph G_{\prec} which do not affect the final ordering of the patterns. This is based on the following proposition.

Proposition 4.6.2. *Let P and P' be two nodes in the graph G_{\prec} . If there is an edge from P to P' in G_{\prec} (that is $P \prec P'$) any edge from the ancestors of P to P' does not alter the ordering of the patterns produced by G_{\prec} .*

Proof. The ordering of the patterns produced by G_{\prec} depends on their levels (*GLevel*) in the graph G_{\prec} . The *GLevel* value of a pattern is the maximum distance of the pattern from a source node in G_{\prec} . Since the transitive edges have smaller distances to the source nodes, they do not affect the *GLevel* value of a pattern. Therefore, they do not alter the ordering of the patterns produced by G_{\prec} . \square

Clearly, because of Proposition 4.6.2, transitive edges can be removed from the G_{\prec} graph.

Graph G_{\prec} is stored in the form of an adjacency list. For each pattern (node) P , a list of pointers to the parent nodes and a list of pointers to the child nodes are maintained. We also maintain the list of source nodes and the list of sink nodes in G_{\prec} . When a new pattern is considered *PatternGraph* checks the existence of homomorphisms starting with a sink node of G_{\prec} and proceeds in a bottom-up way. The sink node list supports the bottom-up computation and the source node list is used for easily detecting the minimal patterns at the end of the process.

The outline of algorithm *PatternGraph* is shown in Algorithm 3. Algorithm *PatternGraph* compares every pattern P_{new} in the input list *PList* of patterns with the patterns P_{old} in the L_{sink} list by calling procedure **Compare** (lines 6-12). If $P_{old} \prec P_{new}$, an edge from P_{old} to P_{new} is added to G_{\prec} and L_{sink} is updated if needed (lines 21-24). If $P_{new} \prec P_{old}$ or P_{new} and P_{old} are incomparable w.r.t. \prec , procedure **Compare** is recursively called for all

parents P_0 of P_{old} (lines 24-29). If $P_0 \prec P_{new}$ and $P_{new} \prec P_{old}$, the transitive edge from P_0 to P_{old} is removed (lines 28-29). If $P_{new} \prec P_{old}$ an edge from P_{new} to P_{old} is added to G_{\prec} and L_{source} is updated if needed (lines 30-33). Further optimizations based on Proposition 4.6.2 (not shown in the outline of Algorithm 3) are implemented in *PatternGraph* to avoid adding transitive edges.

The previous discussion in this section justifies the next proposition about Algorithm *PatternGraph*.

Proposition 4.6.3. *Algorithm PatternGraph correctly constructs the graph G_{\prec} given a set of patterns as input.*

Let m be the number of patterns. Procedure **Compare** compares each pattern to at most m other patterns. That is, it performs $O(m^2)$ comparisons. Comparing one pattern to another for the \prec_h relation can be done in linear time on the size of the pattern assuming every node is annotated by the keywords of its descendant nodes and sibling nodes are ordered based on the annotating keywords. Thus, this takes $O(dk)$, where d is the depth of the XML tree and k is the number of keywords. Comparing two patterns for the \prec_{pph} relation takes $O(dk^2)$ since a node annotated by keyword k_i is allowed to map to a node which is not annotated by k_i but labeled by k_i . Finally, comparing two patterns for the \prec_{aph} relation takes $O(d^2k^2)$ since, in this case, a path can even map different subpaths of another path. Therefore, the *PatternGraph* algorithm constructs the graph G_{\prec} in $O(m^2d^2k^2)$ time. In practice, by exploiting the properties of the homomorphism relations mentioned earlier the algorithm very efficiently avoids most of these comparisons and checks.

After the graph G_{\prec} is constructed, the order \mathcal{O} is extracted by using the *GLevel*, *MCTDepth* and *MCTSize* values of each pattern. *MCTDepth* and *MCTSize* are calculated and stored during the generation of the patterns. In order to calculate the *GLevel* values, the graph G_{\prec} is traversed and the *GLevel* value for each pattern is set to the maximum *GLevel* value of its parents incremented by one.

The filtering semantics for *XReason* depends on the number k of top levels in the

Algorithm 3: PatternGraph algorithm.

```

1 PatternGraph(IN:  $PList$ : list of patterns)
2   boolean[]  $visited$  /* Boolean visited flags for patterns */
3    $L_{sink} = []$  /* List of sink patterns */
4    $L_{source} = []$  /* List of source patterns */
5    $G_{\prec} = []$  /* Empty precedence graph */
6   foreach  $P$  in  $PList$  do
7     if  $G_{\prec}$  is empty then
8        $L_{sink}.add(P)$ 
9        $L_{source}.add(P)$ 
10    else
11      foreach  $P'$  in  $L_{sink}$  do
12         $Compare(P, P')$ 
13      if  $P.parents$  is empty then
14         $L_{source}.add(P)$ 
15      if  $P.children$  is empty then
16         $L_{sink}.add(P)$ 
17      Reset all visited flags
18 Compare( $P_{new}, P_{old}$ )
19   if  $P_{old}.visited = false$  then
20      $P_{old}.visited = true$ 
21     if  $P_{old} \prec P_{new}$  then
22       Add edge( $P_{old}, P_{new}$ ) to  $G_{\prec}$ 
23       if  $P_{old}$  in  $L_{sink}$  then
24          $L_{sink}.remove(P_{old})$ 
25     else
26       foreach  $P_0$  in  $P_{old}.parents$  do
27          $Compare(P_{new}, P_0)$ 
28         if  $P_0 \prec P_{new}$  and  $P_{new} \prec P_{old}$  then
29           remove edge( $P_0, P_{old}$ ) from  $G_{\prec}$ 
30       if  $P_{new} \prec P_{old}$  then
31         add edge( $P_{new}, P_{old}$ ) to  $G_{\prec}$ 
32         if  $P_{old}$  in  $L_{source}$  then
33            $L_{source}.remove(P_{old})$ 

```

graph G_{\prec} . The patterns (nodes) that satisfy this condition (and their ITs thereon) are obtained through a depth first traversal of G_{\prec} up to level k .

4.6.3 An Extension of PatternStack

In this section, we present an extension of *PatternStack*. This extension is based on the observation that the patterns in which the MCT root coincides with the pattern root are usually meaningless. Indeed, these are patterns that link the keyword instances through the root of the XML tree which suggests that these keyword instances are not meaningfully related. We name these patterns *root* patterns. *Root* patterns usually represent a large percentage of all the patterns. Similar remarks about the meaninglessness of *root* patterns have been made in the context of different semantics [79] in order to avoid their processing.

The semantics of *XReason* is expected to capture the meaninglessness of such patterns and eventually rank them in low ranks (in the case of ranking semantics), or exclude them from the query answer (in the case of filtering semantics). Both \prec_{aph} and \prec_{pph} relations (but in particular the \prec_{pph} relation) are effective in pushing down in the G_{\prec} graph the *root* patterns. Further, *MCTDepth* and *MCTSize* rank low the *root* patterns among patterns with the same *GLevel* value.

Nevertheless, even though we do not expect to have an improvement in the effectiveness of *XReason* from the pruning of *root* patterns, terminating their construction before they are even generated substantially improves the performance of *PatternStack*. Further, invoking *PatternGraph* on a much smaller number of patterns, greatly reduces the number of pattern comparisons and the execution time of that algorithm too.

It is important to note that the extended ordering \mathcal{O}' of the non-*root* patterns produced by the extended *PatternStack* complies with the original ordering \mathcal{O} of all the patterns produced by *PatternStack*. That is, the extended *PatternStack* does not alter the order of the remaining patterns. In order to support this claim we first show the following proposition.

Proposition 4.6.4. *Let P_1 and P_2 be two patterns. If $P_1 \prec P_2$ and P_1 is a root pattern then P_2 is also a root pattern.*

Proof. Let P_1 and P_2 be two patterns. If $P_1 \prec P_2$, then one of the following relations hold:

$P_1 \prec_h P_2$, $P_1 \prec_{aph} P_2$ or $P_1 \prec_{pph} P_2$. We prove the proposition above by considering these cases:

- (a) If $P_1 \prec_h P_2$ or $P_1 \prec_{aph} P_2$, and P_1 is a *root* pattern, for the needs of \prec_h or \prec_{aph} relations, the root of the MCT of P_1 has to be mapped to a node in the MCT of P_2 . Since, the label of the root of the XML tree is unique, P_2 is also a *root* pattern.
- (b) If $P_1 \prec_{pph} P_2$, the root of the MCT of P_1 is mapped to a descendant of the root of the MCT of P_2 . Therefore, P_1 cannot be a *root* pattern.

□

Proposition 4.6.4 states that if a pattern in the G_{\prec} graph is a *root* pattern, all its descendant patterns are also *root* patterns. Consider the ordering \mathcal{O} of the patterns induced by the G_{\prec} . If we eliminate all *root* patterns and their incident edges from G_{\prec} , the order between the remaining non-*root* patterns will be preserved in the ordering \mathcal{O}' of the patterns induced by the resulting graph G'_{\prec} (which is the graph produced by the extended *PatternStack*). This is formalized by the next proposition.

Proposition 4.6.5. *Let P_1 and P_2 be two non-root patterns. If P_1 precedes P_2 in the original ordering \mathcal{O} then P_1 also precedes P_2 in the extended ordering \mathcal{O}' .*

Proof. Rank of a pattern with respect to the *XReason* semantics depend on *GLevel*, *MCTDepth* and *MCTSize*, respectively. Within these metrics, only *GLevel* is affected by the organization of the graph.

Based on Proposition 4.6.4, one can see that *root* patterns are always preceded by non-*root* patterns with respect to \prec , thus they are placed farther from the source nodes compared to non-*root* patterns. So, the exclusion of *root* patterns from the G_{\prec} graph does not affect the *GLevel* values of non-*root* patterns. Therefore, in the extended ordering \mathcal{O}' , the original order of non-*root* patterns is preserved. □

Depending on Proposition 4.6.5, the extended *PatternStack* algorithm can be safely used to improve the performance of *PatternStack*.

The modification of PatternStack for the implementation of the extension under discussion is confined to the pop() procedure of Algorithm 2. The extended PatternStack treats in a different way the nodes that are children of the document root. The patterns that are rooted on them are not extended to their parent (i.e., the document root) in order to construct new patterns. Thus, if the stack contains exactly two entries (i.e., the document root and one of its children), lines 6, 8 and 9 are not executed. This way, the children of the root node may contribute only complete patterns, while their partial ones are not further processed.

4.7 Experimental Evaluation

We performed experiments to measure the efficiency and effectiveness of *XReason* as a filtering and ranking system. We compare the quality of our results to that of previous approaches.

In contrast to the IR domain [20], there is no standard benchmark to evaluate the effectiveness of keyword search on data-oriented XML [79].

We use Mondial, SIGMOD, EBAY, NASA and DBLP datasets for the experiments which are obtained from the UW XML Data Repository². Statistics for these datasets are depicted in Table 4.2. We do not distinguish between elements and attributes, and we represent attributes in these datasets as elements. For the effectiveness experiments, we use Mondial and SIGMOD datasets which are often used for this purpose in XML keyword search research [6, 53]. We used NASA and DBLP datasets which are larger for the scalability experiments. NASA and DBLP datasets show different characteristics: DBLP is a large but shallow dataset whereas NASA is a relatively smaller but deep dataset. Because of these different characteristics, we can measure the efficiency of our algorithms in a representative environment. The experiments were conducted on a 2.9 GHz Intel Core

²<http://www.cs.washington.edu/research/xmldatasets/>

Table 4.2 Mondial, SIGMOD, DBLP and NASA Dataset Statistics

	Mondial	SIGMOD	DBLP	NASA
Size	1 MB	467 KB	127 MB	23 MB
# nodes	69,846	15,263	3,736,406	523,963
# distinct tags	50	12	50	70
# distinct label paths	119	12	145	111
Average depth	3.00	4.60	1.93	4.56
Maximum depth	5	6	5	7

i7 machine with 3 GB memory running Ubuntu.

We first introduce the metrics we use for the experimental evaluation; then, we present our results on effectiveness for both filtering and ranking semantics, and finally we present our efficiency experiments.

4.7.1 Metrics

Since the ranking approaches we consider may view a number of results as equivalent (i.e., having the same rank) we extend below the metrics that are usually used to measure the quality of ranking. In order to determine the ground truth for the effectiveness experiments, we employed five expert users who are not involved in this project which characterized the query patterns as relevant or irrelevant to the query. The relevancy of each pattern (relevant or irrelevant) was determined by the majority of the characterizations of the expert users.

Filtering Experiments. Since *XReason* works with patterns, if a pattern is among those with the smallest k *GLevel* values, all candidate results that conform to it are regarded as relevant and are returned to the user. We use *precision* and *recall* to measure the effectiveness of filtering semantics. *Precision* is the ratio of the number of relevant results in the result set of the system to the total number of results returned by the system. *Recall* is the ratio of the number of relevant results in the result set of the system to the total number of relevant results.

Ranking Experiments. For the ranking experiments, we employ two metrics: Mean Av-

erage Precision (MAP) and precision@N.

MAP is the mean average precision of a set of queries with average precision of a query being the average of precision scores after each relevant result of the query is retrieved. As a ranking effectiveness metric, MAP takes the order of the results into account.

We extend MAP so that it takes into account equivalence classes of results. An equivalence class in a ranked list is a set of all the results which have the same rank. Different orderings of these results in the ranked list would affect the value of MAP [69]. For this reason, we define and compute *worst* and *best* versions for MAP. In the *worst* (resp. *best*) version, the ranked list is assumed to have the correct results ranked at the end (resp. beginning) of each equivalence class. This extension allows us to compute upper and lower bounds for the ranking metrics between which the scores of all the possible rankings lie. We denote these metrics as MAP_{worst} and MAP_{best} . We also computed the *expected* MAP value by averaging over all the queries, the average AP of the possible rankings of the results of every query. This latter metric is denoted MAP_{exp} . Clearly, $MAP_{worst} \leq MAP_{exp} \leq MAP_{best}$. When the possible rankings was too numerous to be computed exhaustively, we used sampling to compute MAP_{exp} .

In order to assess the effect of answer set size on precision in ranking experiments, we also measure precision with a cutoff point for the number N of results which is called precision@N ($P@N$). Similarly to MAP, we consider two versions of $P@N$: $P@N_{worst}$ and $P@N_{best}$, and also compute the expected $P@N$ value.

4.7.2 Effectiveness of Filtering Semantics

For the filtering experiments, we compare *XReason* with $k = 1$ (that is, we consider only the patterns with the smallest *GLevel* value) with three well-known baseline approaches: SLCA [18, 35, 88], ELCA [32, 89] and VLCA [21, 43]. We also compare with two more recent approaches, XReal [6, 7] and the Coherency Ranking (CR) approach [79].

In order to allow the comparison of *XReason*, which returns ITs and not simply

LCAs with the other approaches, we use the definitions of SLCA, ELCA and VLCA semantics in terms of ITs provided in Table 4.1. XReal infers promising result node types (label paths from the root) and ranks and returns the nodes that match these node types. In order to compare XReal with *XReason* we adjusted XReal in Table 4.3 so that it returns ITs and we named this new approach *ITReal*. For a query Q on an XML tree T , $ITset(Q, T)$ denotes the set of ITs of the instances of Q on T . $XReal_{Nodes}$ denotes the set of nodes in T that match the node type inferred by XReal.

Table 4.3 Definitions of XReal Semantics in Terms of ITs

Approach	Definition of answer of Q on T
<i>ITReal</i>	$\{t \mid t \in ITset(Q, T), n = root(MCT(t)), \text{ and } \exists n' (n' \in XReal_{Nodes} \text{ and } n' < n)\}$

CR does not need an adjustment as it returns subtrees similar to the ITs of *XReason*. CR can be directly compared to *XReason* because, like *XReason*, it partitions the results into patterns. It also characterizes the relevance of the patterns, not of individual results. Since CR excludes root patterns, we consider in the effectiveness experiments non-root patterns. As in [79], the factor $f(n)$ of NTC is set to $n^2/(n-1)^2$ and the threshold of NTC is set to zero.

We run 20 queries on Mondial and SIGMOD datasets shown in Tables 4.4 and 4.5, respectively. Most of the queries are chosen from previous papers: M1-M7 and S1-S8 from [57], M8-M12 and S9-S11 from [53], M13 from [54], M14-M16 from [55], and S12-S15 from [45]. Tables 4.4 and 4.5 also show the total number of patterns (results) and the number of relevant patterns (results) for each query. Precision scores of the different approaches for all the queries on the Mondial and SIGMOD datasets are presented in Figures 4.11 and 4.12, respectively. Recall scores are only shown for the queries on Mondial in Figure 4.13. The recall scores for the queries on SIGMOD are all equal to 1.0 for all approaches with the exception of *XReason* on query S2 which is 0.95.

Table 4.6 provides average precision and recall scores for all approaches on both datasets. All approaches show good recall with CR and *ITReal* topping the list and *XReason*

Table 4.4 Queries Used in the Experiments on the Mondial Dataset

Query ID	Keywords	# relevant patterns	# patterns	# relevant results	# results
M1	<i>torneaelv, country, province</i>	1	1	1	1
M2	<i>roman, catholic, percentage, united, states</i>	1	8	2	36
M3	<i>population, 87, albania, city</i>	1	7	12	504
M4	<i>organization, name, members</i>	2	2	10,111	10,111
M5	<i>country, government, republic</i>	2	17	115	3,579
M6	<i>country, ethnicgroups, german</i>	2	14	19	3,066
M7	<i>city, washington, province</i>	3	37	6	149,352
M8	<i>france, territory</i>	1	1	3	3
M9	<i>lake, located</i>	2	3	97	124
M10	<i>singapore, country</i>	3	4	4	6
M11	<i>religions, christian, muslim</i>	2	6	86	111
M12	<i>province, houston, dallas</i>	1	6	1	294
M13	<i>belarus, population</i>	2	2	4	4
M14	<i>united, states, birmingham, population</i>	2	12	6	3,198
M15	<i>ethnicgroups, chinese, indian, capital</i>	2	6	16	183
M16	<i>country, muslim</i>	2	8	101	2,209
M17	<i>international, monetary, fund, established</i>	1	1	1	1
M18	<i>government, democracy, muslim</i>	2	2	17	17
M19	<i>jewish, percentage</i>	2	9	17	109
M20	<i>japan, tokyo, population</i>	2	10	4	244

and ELCA doing almost equally well. However, in terms of precision ITReal and CR display the worst scores closely followed by ELCA and SLCA. Both ITReal and CR, as *XReason*, are also ranking systems and they can use their ranking capacity to reduce the

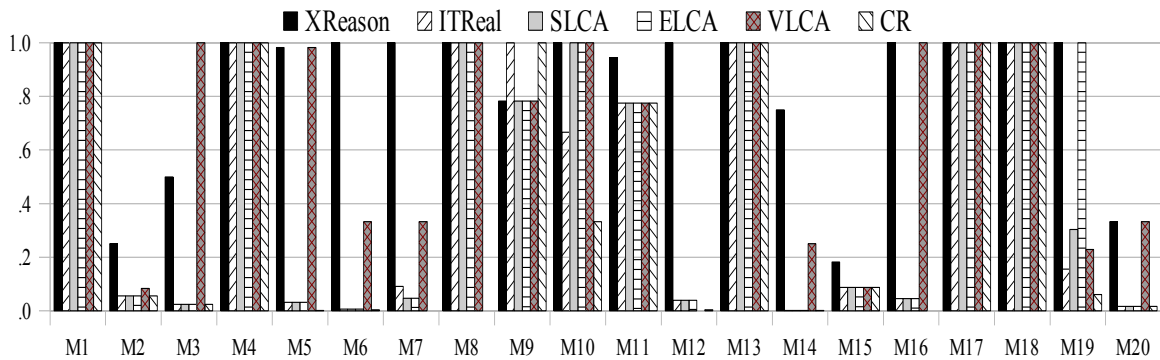
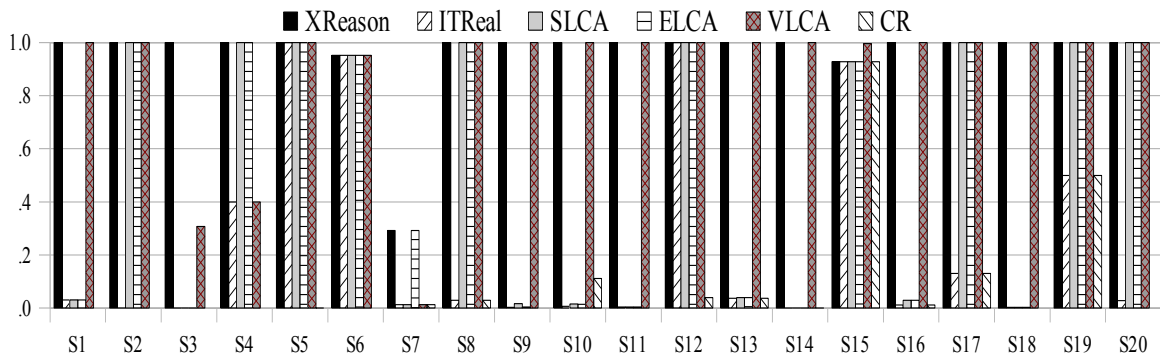
**Figure 4.11** Precision scores for the queries of Table 4.4 on the Mondial dataset.

Table 4.5 Queries Used in the Experiments on the SIGMOD Dataset

Query ID	Keywords	# relevant patterns	# patterns	# relevant results	# results
S1	<i>author, position, 01, harry, article</i>	1	217	2	74,392,500
S2	<i>jim, gray, title, initpage, endpage</i>	2	67	19	8,976,784
S3	<i>initpage, 3, endpage, 7</i>	1	65	4	56,210
S4	<i>author, nicolas</i>	1	3	2	197
S5	<i>article, title, author</i>	2	10	3,738	11,309,368
S6	<i>initpage, 7, article, endpage</i>	2	55	20	1,940,838
S7	<i>volume, 11, article</i>	1	9	12	934
S8	<i>asuman, pinar, article</i>	1	5	4	135
S9	<i>directions, database, research</i>	1	5	1	366
S10	<i>jennifer, widom, jeffrey, d, ullman</i>	1	47	2	294
S11	<i>relational, model, author, date</i>	1	10	1	238
S12	<i>karen, title</i>	1	2	2	51
S13	<i>anthony, data</i>	1	2	2	55
S14	<i>article, data, john</i>	1	5	3	7,801
S15	<i>database, volume, number</i>	1	3	347	374
S16	<i>divesh, srivastava, database</i>	1	5	2	174
S17	<i>michael, stonebraker, postgres</i>	1	6	3	23
S18	<i>database, systems, security</i>	1	5	1	417
S19	<i>christos, faloutsos, signature, files</i>	1	2	1	2
S20	<i>efficient, maintenance, materialized, views, subrahmanian</i>	1	24	1	36

**Figure 4.12** Precision scores for the queries of Table 4.5 on the SIGMOD dataset.

negative effect of a large size answer set on precision. For this reason, in the next section we also measure P@N. As we can see, *XReason* largely outperforms all the other approaches in terms of precision and shows almost perfect recall on both datasets.

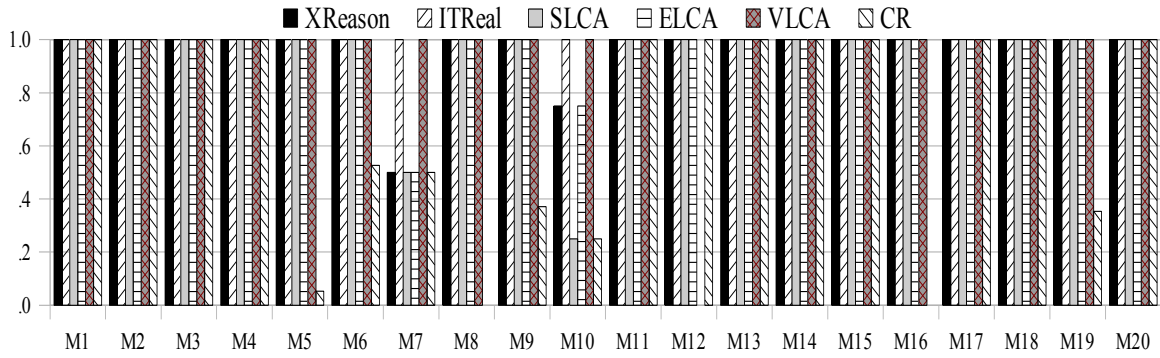


Figure 4.13 Recall scores for the queries of Table 4.4 on the Mondial dataset.

Table 4.6 Average Precision and Recall Scores for the Queries of Table 4.4 and 4.5

Dataset	Metric	<i>XReason</i>	<i>ITReal</i>	<i>SLCA</i>	<i>ELCA</i>	<i>VLCA</i>	<i>CR</i>
<i>Mondial</i>	<i>Avg. Prec.</i>	0.84	0.45	0.46	0.50	0.66	0.47
	<i>Avg. Rec.</i>	0.96	1.00	0.94	0.96	0.95	1.00
<i>SIGMOD</i>	<i>Avg. Prec.</i>	0.96	0.25	0.50	0.52	0.88	0.10
	<i>Avg. Rec.</i>	1.00	1.00	1.00	1.00	1.00	1.00

4.7.3 Effectiveness of Ranking Semantics

In order to evaluate the effectiveness of the ranking semantics of *XReason* we computed the queries of Tables 4.4 and 4.5 under *XReason*, *ITReal* and *CR* semantics on the datasets and we measured best and worst bounds and expected values for MAP and P@N.

Table 4.7 shows the MAP scores. *XReason* outperforms *CR* and largely outperforms *ITReal*. Since *XReason* ranks the correct results almost always higher than the incorrect ones, it has almost perfect MAP scores.

Table 4.7 Best and Worst MAP Scores for the Queries of Tables 4.4 and 4.5

Dataset	Semantics	MAP_{worst}	MAP_{best}	MAP_{exp}
<i>Mondial</i>	<i>XReason</i>	0.97	0.97	0.97
	<i>ITReal</i>	0.48	0.76	0.51
	<i>CR</i>	0.71	0.71	0.71
<i>SIGMOD</i>	<i>XReason</i>	1.00	1.00	1.00
	<i>ITReal</i>	0.34	0.63	0.37
	<i>CR</i>	0.90	0.90	0.90

Best and worst P@N scores are shown in Figures 4.14 and 4.15. For a given query and a given approach, best and worst scores are shown on the same column with worst

scores superimposing best scores, i.e., if the scores are the same, only worst scores are visible. $N = 10$ for all datasets because most of the queries have few correct results. The average $P@10_{exp}$ scores are displayed in Table 4.8. For *ITReal*, limiting the result set size did not have a significant effect on the precision for most of the queries, which means that some incorrect results are ranked high in the result list. *XReason* has perfect $P@10$ scores in almost all cases.

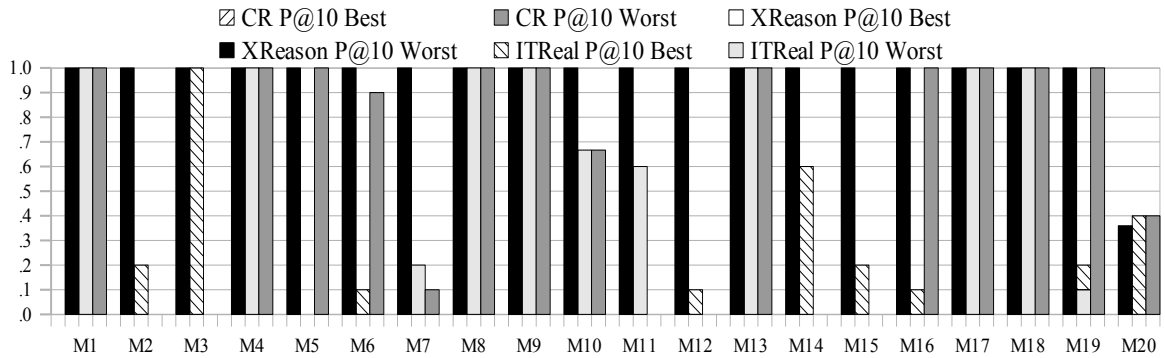


Figure 4.14 Best and worst $P@10$ scores for the queries of Table 4.4 on Mondial dataset.

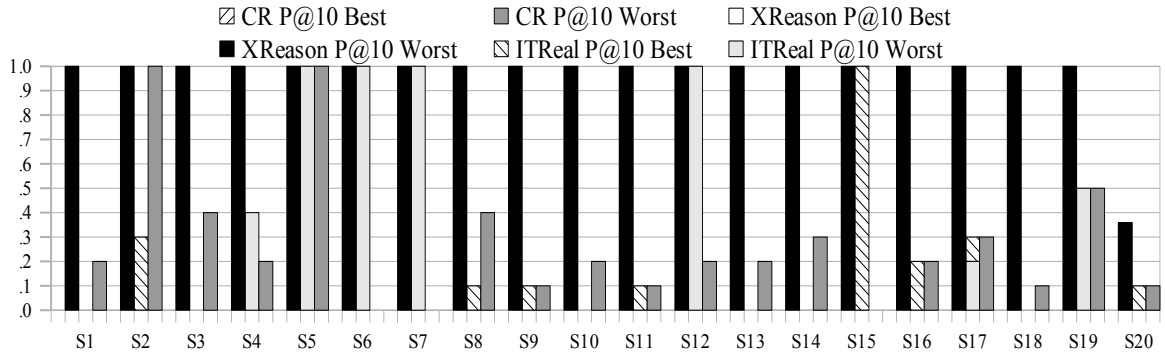


Figure 4.15 Best and worst $P@10$ scores for the queries of Table 4.5 on SIGMOD dataset.

Table 4.8 Average $P@10_{exp}$ Scores for the Queries of Tables 4.4 and 4.5

Dataset	<i>XReason</i>	<i>ITReal</i>	<i>CR</i>
<i>Mondial</i>	0.97	0.44	0.60
<i>SIGMOD</i>	1.00	0.27	0.28

4.7.4 Efficiency

In order to evaluate the efficiency of the algorithms we proposed: (a) we compared the computation time of our original algorithm to a naïve algorithm, (b) we run scalability experiments for the original and the extended algorithms, and (c) we compared the original vs. the extended algorithm.

The naïve algorithm for implementing the *XReason* semantics generates all the ITs of the query using the inverted lists of the keywords and iterates over them to extract the patterns. Then, it checks for the existence of homomorphisms between the pairs of patterns in a straightforward way and computes the query results. Figure 4.16 shows the computation times of the odd numbered queries of Tables 4.4 and 4.5 for our original algorithm and the naïve algorithm on the Mondial and SIGMOD datasets. We only report on half of the queries. Note that the y-axis is in logarithmic scale. Times exceeding 10,000 seconds

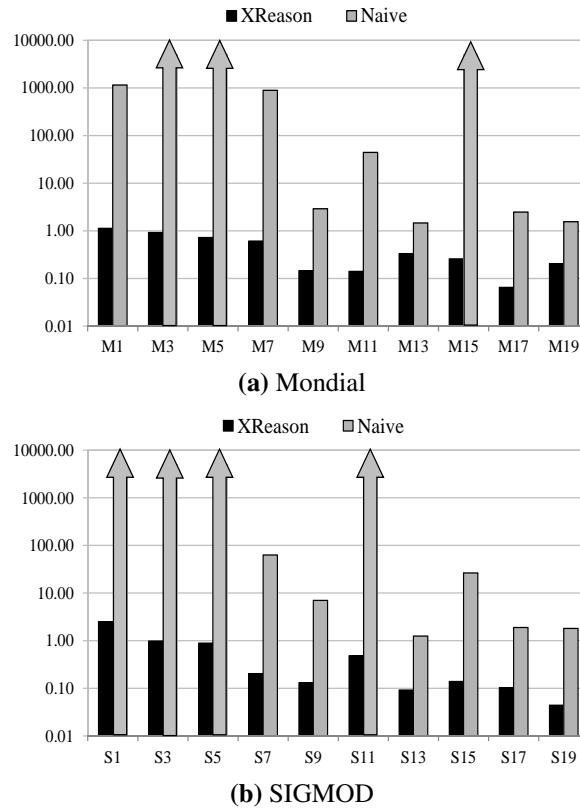


Figure 4.16 *XReason* execution times (in secs) for the queries of Tables 4.4 and 4.5.

Table 4.9 Queries Used in Scalability Experiments

Algorithm	Dataset	Keywords
<i>Original algorithm</i>	DBLP	<i>osawa, sculptured, cricket, marriages, erhebung, ilpo</i>
	NASA	<i>iccd, brightnesses, colloquium, hendry, perugia, attribute</i>
<i>Extended algorithm</i>	DBLP	<i>srinivas, elias, masao, divyakant, sums, lectures</i>
	NASA	<i>medium, dr, seen, heii, oxygen, comparing</i>

for the naïve algorithm are shown with arrows. Our algorithm is at least two orders of magnitude faster than the naïve algorithm in most of the cases. The average computation times for our algorithm over 20 queries on Mondial and SIGMOD datasets are 0.50 and 0.51 seconds, respectively. This performance is acceptable for real-time search systems even without additional optimizations.

For our scalability experiments, we used DBLP and NASA datasets. For the original algorithm, we measured the computation time in relation to the output size (the number of ITs). The original algorithm depends heavily on the output size. For the experiments, we use the most frequent four, five and six keywords to form three different queries from the randomly chosen keywords shown in Table 4.9. Table 4.9 lists the keywords in descending order of their frequencies. For each query, we truncated the keyword inverted lists at 20%, 40%, 60% and 80% of their length. The computation times are presented in Figure 4.17. Obviously, the higher the number of keywords, the higher the number of ITs returned. As we can see in Figure 4.17, the original algorithm scales well even though the number of ITs increases very fast with the percentage of the inverted lists being used.

For the scalability experiments on the extended algorithm, we measured the computation time in relation to the input size (the total number of keyword instances in the XML tree of all the query keywords). The extended algorithm depends mainly on the input size since it eliminates non-*root* patterns and returns a restricted number of ITs. We selected queries from Table 4.9 and truncated keyword inverted lists as we did in the previous scal-

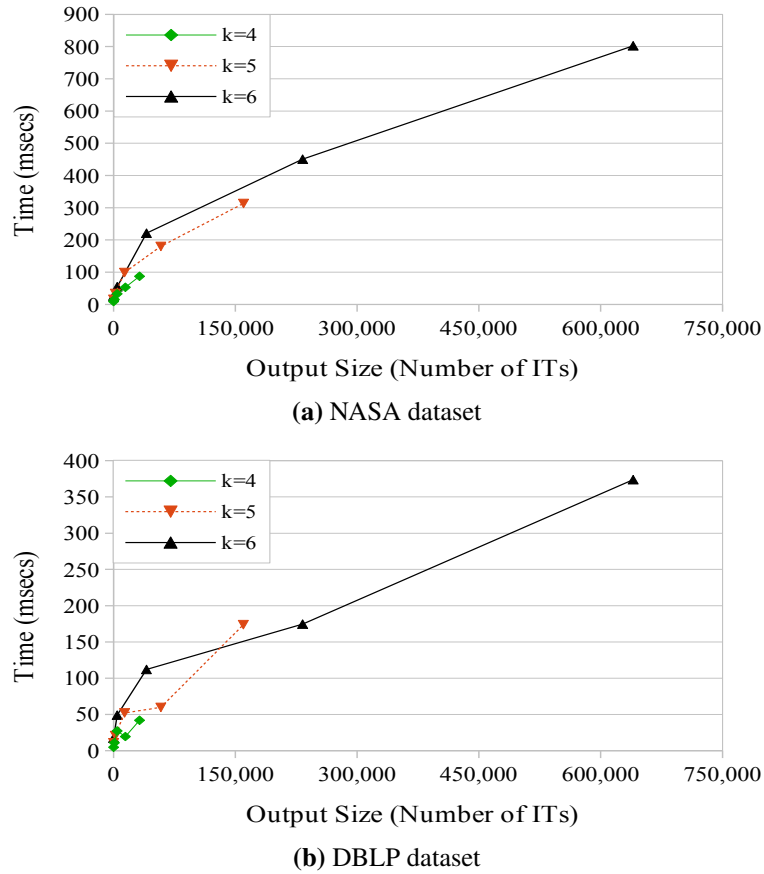


Figure 4.17 Computation time vs. output size for the original algorithm using queries with 4, 5 and 6 keywords.

ability experiment. We present the measured computation times in Figure 4.18. As shown in the figure, the extended algorithm scales smoothly (it is almost linear). This is due to the fact that the extended algorithm prunes partial patterns early on in the computation before they become complete patterns as long as they are rooted at the root of the XML tree.

For our experiments on comparing the original and the extended algorithm, we run five queries on the DBLP dataset to measure the number of ITs, the number of generated patterns, and the computation time of the two algorithms. The queries are shown in Table 4.10. We selected real-world queries in order to guarantee that they return a reasonable number of non-*root* patterns and therefore, to highlight the differences in the performance of two algorithms. The results are shown in Figure 4.19. In Figures 4.19a and 4.19b, we show the number of ITs and the number of generated patterns for each query, respectively.

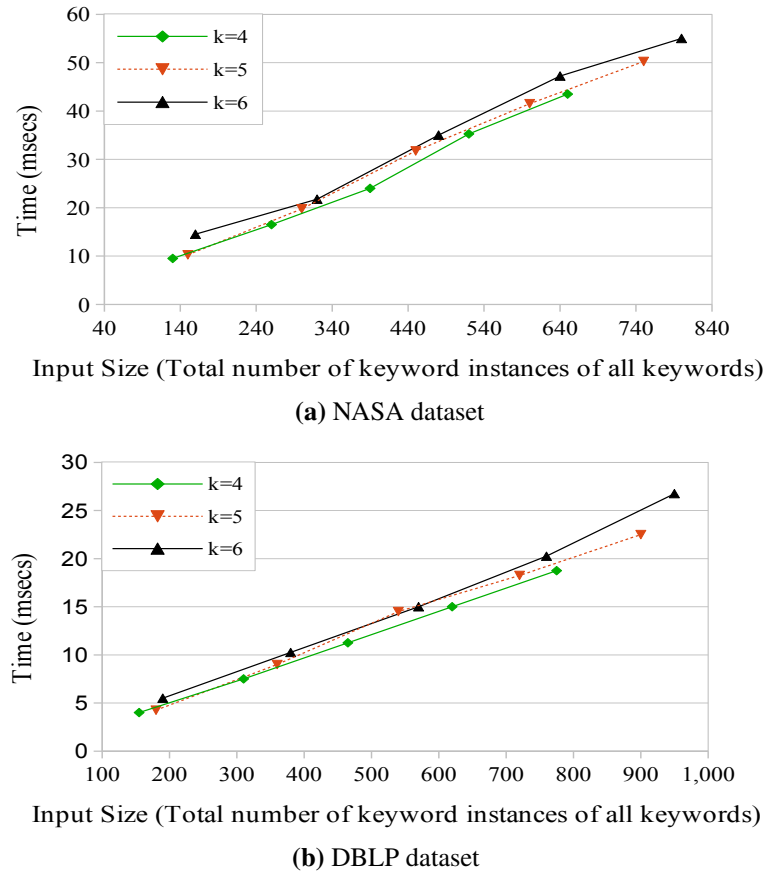


Figure 4.18 Computation time vs. input size for the extended algorithm using queries with 4, 5 and 6 keywords.

The y-axes are in logarithmic scale. The number of *root* patterns is significantly greater than the number of non-*root* patterns for all queries. The same remark applies to the number of *root* and non-*root* ITs. In Figure 4.19c, we show the computation time of both algorithms. As we can see in this figure, the extended algorithm significantly outperforms the original algorithm. These results show that the extended algorithm is a good substitute of the original one in real-world applications.

4.8 Conclusion

We have proposed *XReason*, a novel approach for providing ranking and filtering semantics to keyword queries on XML data which is based on reasoning with patterns. The patterns record the structural and semantic characteristics of the query matches. In order to rea-

Table 4.10 Queries on the DBLP Datasets Used to Compare the Performance of the Original vs. the Extended Algorithm

Query ID	Keywords
Q1	<i>xml, keyword, search</i>
Q2	<i>query, analysis</i>
Q3	<i>sequence, alignment</i>
Q4	<i>collaborative, filtering, recommendation</i>
Q5	<i>dynamic, incremental, clustering</i>

son with patterns, we introduced homomorphisms between patterns which are leveraged to define homomorphism relations on patterns. Our approach benefits from a global view of the query matches and avoids the pitfalls of previous semantics which rely on comparing query matches locally in the XML tree or rank them based simply on a scoring function. By reasoning with patterns whose number is typically very small compared to the number

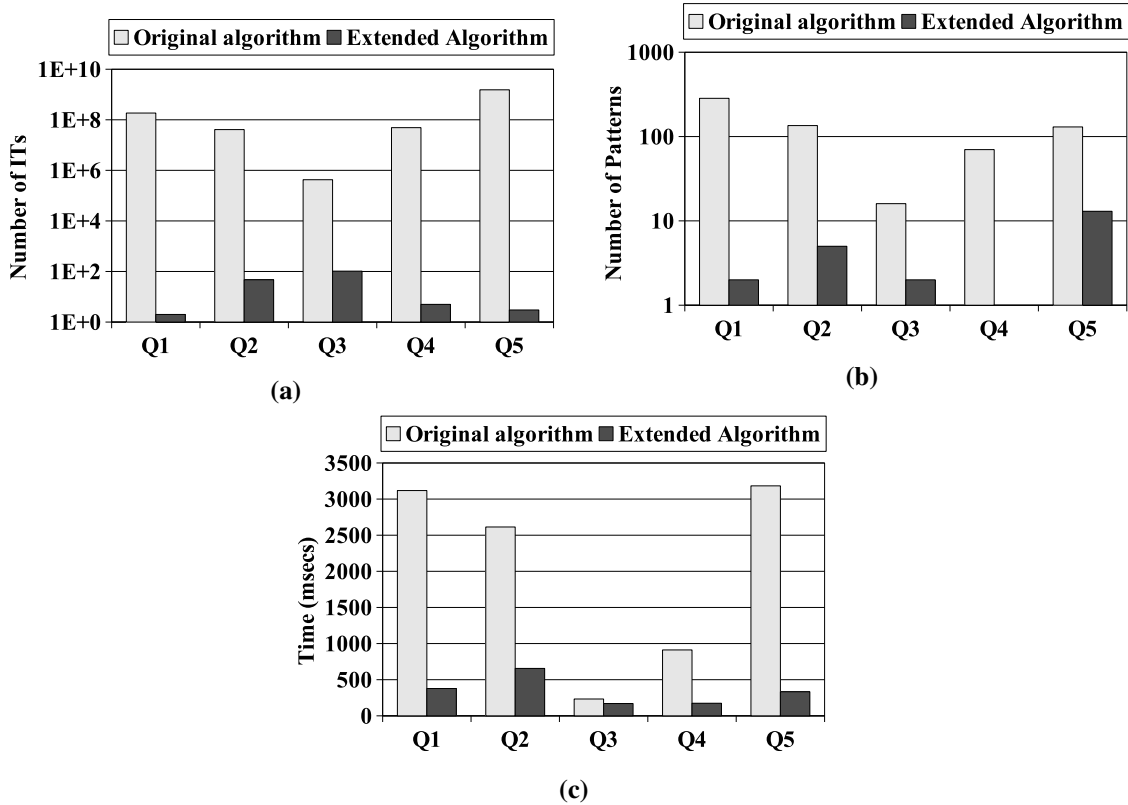


Figure 4.19 (a) number of ITs, (b) number of generated patterns, and (c) the computation time of the original and the extended algorithm on the DBLP dataset using the queries of Table 4.10.

of query matches, we make this global and multi-feature comparison feasible. We designed an efficient stack-based algorithm to implement *XReason* and we also devised a heuristic extension to improve its performance. Contrary to most previous algorithms, ours works with the keyword inverted lists and does not require any auxiliary data structures and preprocessing of the data. Our experimental studies over several real datasets show that *XReason* outperforms previous approaches in precision, precision@N, recall and mean average precision. Further, they showed that our algorithms are fast and scale smoothly and therefore, our approach is computationally feasible and can be applied in practice.

CHAPTER 5

SEARCH RESULT CLUSTERING

Usually, there is a large number of results that match a user query (candidate results) of which very few are relevant to the query. Different filtering semantics [32, 43, 48, 88, 89] have been introduced which try to exploit structural and/or semantic properties of the results and the data in order to filter out irrelevant results. In addition, ranking semantics [6, 18, 32, 65, 79] have been proposed which exploit, in general, statistical information and attempt to rank the results placing on top those that are more relevant. Both the filtering and ranking semantics are ad-hoc and, in the general case, fail to produce results of high quality [79].

An alternative view to keyword search involves clustering the results using structural and semantic information [13, 53, 57]. In this direction, we elaborate in this dissertation on a novel approach which clusters the results at different levels of granularity and then, exploits user input to navigate among the clusters in order to retrieve the relevant results.

5.1 Clustering Methodology

Our approach clusters results in three different levels. The clusters at the lowest level partition the results. Clusters in higher levels contain clusters from lower levels. In this sense, the clusters at different levels define groups of results of different granularities — coarser at the top level and more refined in the lowest level. Every cluster has a representative. The users can navigate through the system by selecting clusters initially at the top level and by drilling down to their nested clusters and finally to the results. The selection of clusters at every level is facilitated by the ranking of the relevant clusters which is used to present the results to the user.

5.1.1 First (Bottom) Level of Clustering: Patterns

In practice, multiple distinct ITs can match the keywords in the same way and share the same structural and semantic properties (that is, they share the same pattern). These individual ITs are not usually of particular interest as query results. In contrast, the users are interested on the patterns that they define. Therefore, we use patterns (defined next) for clustering the results at the first level of our clustering scheme.

Definition 5.1.1 (IT pattern). *A pattern P of a query Q on a data tree T is a tree which is isomorphic (including the annotations) to an IT of Q on T . The MCT of a pattern P refers to P without the path that links the LCA of the annotated nodes to the root of P .*

A pattern has all the information of an IT except the physical location of that one in the data tree. As an example, Figure 5.1 shows four patterns (out of 32 in total) of the keyword query $Q = \{Advanced, Database, Systems\}$ on the data tree T of Figure 1.3. Pattern P_3 is the pattern of the IT of Figure 3.2(a). Pattern P_2 has two ITs which comply with it: the IT of the query instance $\{(Advanced, 8), (Database, 8), (Systems, 8)\}$ and the IT of the query instance $\{(Advanced, 10), (Database, 10), (Systems, 10)\}$.

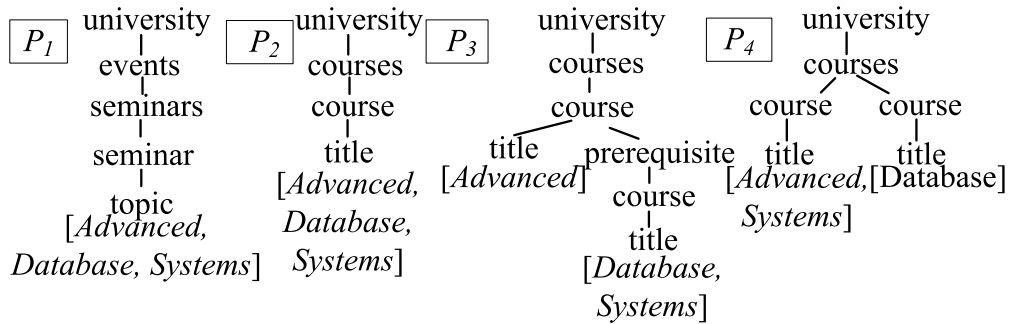


Figure 5.1 Some patterns for $Q = \{Advanced, Database, Systems\}$ on the tree of Figure 1.3.

At the first clustering level, a cluster is the set of ITs which comply with a pattern and is represented by this pattern. Patterns are used as representatives for clusters at all levels. We denote the representative of a cluster \mathcal{C} as $repr(\mathcal{C})$.

A pattern represents a possible interpretation of a query on an XML tree. The first level of clustering comprises all possible interpretations of the query on the XML tree since all the ITs of the query on the XML tree are grouped into patterns and the only information missing in a pattern is the physical location of the ITs in the XML tree.

5.1.2 Second Level of Clustering: Classes

Different patterns can be similar in the sense that they match the keywords in the same way (that is, they have the same root-to-annotated-node paths), and they have the same LCA. These patterns are semantically very close since they only differ in the way they combine keyword instances to form partial LCAs. We put such patterns in the same cluster to form the second level of clustering. These clusters of patterns are called classes.

We formally introduce classes using the concept of the \approx relation which is defined next.

Definition 5.1.2 (\approx relation). *Let P and P' be two patterns of a query on a data tree. We say that $P \approx P'$ if both of the following conditions hold:*

- (a) *the root-to-LCA-paths of P and P' are the same, and*
- (b) *for every keyword in the query, the LCA-to-keyword instance path in P and P' is the same.*

Clearly, \approx is an equivalence relation. Figure 5.2 shows two patterns, P_4 and P_5 . As the arrowed lines indicate, the paths of these patterns satisfy both conditions of Definition 5.1.2. Therefore, $P_4 \approx P_5$.

Definition 5.1.3 (Class). *Given the set of patterns of a query Q on a data tree T , a class of Q on T is an equivalence class of patterns with respect to the \approx relation.*

Figure 5.3 shows four patterns. Clearly, these patterns belong to the same class. If all the patterns of our running example are examined, one can see that patterns of Figure 5.3 form a class.

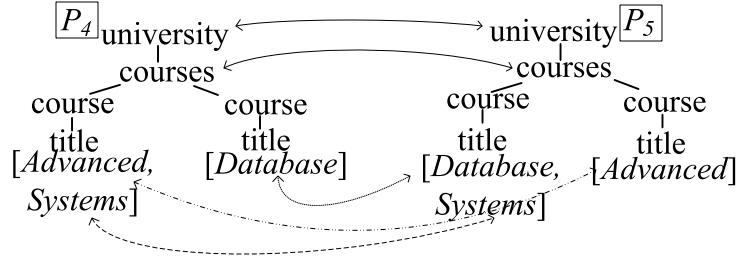


Figure 5.2 Path correspondences between two patterns, P_4 and P_5 .

We call *size* of a pattern P the number of edges in P . A pattern with the smallest size is chosen as a representative of a class (in case of a tie, a representative is chosen randomly among the patterns with the smallest size). For instance, for the class of Figure 5.3 we chose pattern P_4 as a representative of the class among the equal sized patterns P_4, P_5 and P_6 (size = 5). Pattern P_7 is of larger size (size = 7).

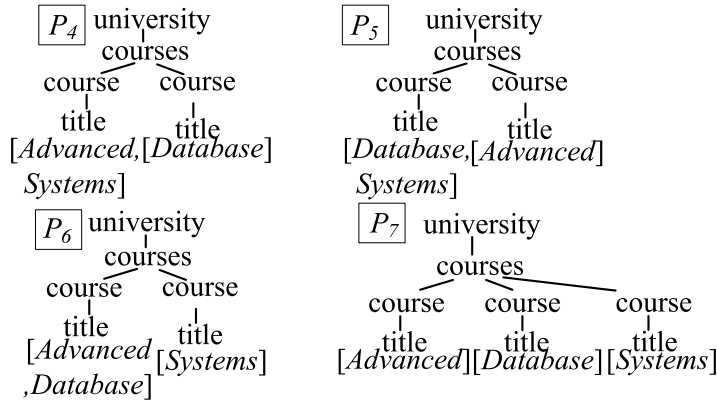


Figure 5.3 A class of patterns consisting of four patterns.

5.1.3 Third Level of Clustering: Collections

The \approx relation identifies similarities between patterns by detecting identical path subpatterns between these patterns. However, similarities between patterns can also be identified by detecting common path subpatterns in a more relaxed way in the sense that the paths of one pattern can be embedded to the paths of another pattern. Embedding a path p_1 into a path p_2 means that the edges of p_1 are mapped to sequences of edges in p_2 . That is, an edge in p_1 is viewed not as a child but as a descendant relationship between its nodes.

We capture this type of similarity between patterns using the concepts of *descendant path homomorphism* and \prec_{dph} relation. We then use the \prec_{dph} relation to cluster classes into collections which is the third level of clustering.

Definition 5.1.4 (Descendant Path Homomorphism). *Let p_1 and p_2 be two pattern paths whose last nodes have the same label and are annotated by the same keyword. There is a descendant path homomorphism from p_1 to p_2 iff there is a function dph from the nodes of p_1 to the nodes of p_2 such that:*

- (a) *for every node n in p_1 , n and $dph(n)$ have the same labels.*
- (b) *if n' is a child of n in p_1 , $dph(n')$ is a descendant of $dph(n)$ in p_2 (that is, there is a path from $dph(n)$ to $dph(n')$ in p_2).*

For instance, in Figure 5.4, the path `courses/course/title[Database]` of pattern P_4 has a descendant path homomorphism to `courses/course/prerequisite/course/title[Database]` of pattern P_8 . Similarly, the path `courses/course/title[Advanced]` of P_4 has a descendant path homomorphism to `courses/course/title[Advanced]` of P_8 since they are identical.

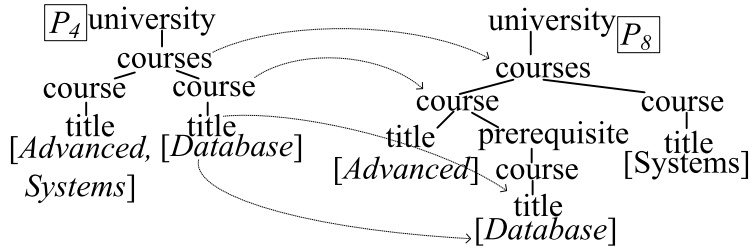


Figure 5.4 Descendant path homomorphism from a path in P_4 to a path in P_8

We use the concept of descendant path homomorphism to define the \prec_{dph} relation between patterns:

Definition 5.1.5 (\prec_{dph} relation). *Let P and P' be two patterns of a query Q on a data tree.*

$P \prec_{dph} P'$ iff

- (a) *P and P' share the same root-to-LCA path.*

(b) for every keyword k in Q , the path from the LCA to the node annotated by k in P has a descendant path homomorphism to a path in the MCT of P' .

Consider the patterns P_4 and P_8 of Figure 5.4. As one can see, $P_4 \prec_{dph} P_8$ since for each one of the keywords *advanced*, *database*, *systems*, the path from `courses` to the node annotated by this keyword has a descendant path homomorphism to a path in the MCT of P_8 .

The following proposition shows how \prec_{dph} relates to \approx .

Proposition 5.1.1. *Let P and P' be two patterns of a query Q in an XML tree T . $P \prec_{dph} P'$ and $P' \prec_{dph} P$ iff $P \approx P'$.*

Proof. We prove the proposition above by proving the following two cases: (a) For any two patterns, P and P' , if $P \prec_{dph} P'$ and $P' \prec_{dph} P$, then the root-to-LCA-path of P and P' are the same. Also, since the MCT paths of P can be mapped to those of P' with a descendant path homomorphism and vice versa, for every keyword in P and P' , the LCA-to-keyword-instance path in P and P' is the same. Therefore, $P \approx P'$. (b) If $P \approx P'$, then for every keyword in the query, the LCA-to-keyword instance path in P and P' is the same. Hence, existence of descendant path homomorphisms is trivial in both directions. Therefore, $P \prec_{dph} P'$ and $P' \prec_{dph} P$. \square

The following proposition characterizes the \prec_{dph} relation when restricted to the representatives of the classes of a query in relation to \prec_{dph} .

Proposition 5.1.2. *Let \mathcal{R} be the set of representatives of the classes of a query on an XML tree. The relation \prec_{dph} is a partial order on \mathcal{R} (i.e., it is reflexive, transitive and antisymmetric).*

Proof. \prec_{dph} relation satisfies all three conditions of a partial order on \mathcal{R} .

(a) The \prec_{dph} relation is reflexive on \mathcal{R} : for any pattern P , $P \prec_{dph} P$ is trivial since both conditions of Definition 5.1.5 are satisfied.

- (b) The \prec_{dph} relation is transitive on \mathcal{R} : for any three patterns P, P' and P'' , if $P \prec_{dph} P'$ and $P' \prec_{dph} P''$ then both properties of Definition 5.1.5 holds from P to P' and from P' to P'' . Therefore, this property also holds from P to P'' . Thus, $P \prec_{dph} P''$.
- (c) The \prec_{dph} relation is antisymmetric on \mathcal{R} : for any two distinct patterns P and P' , if $P \prec_{dph} P'$, both properties of Definition 5.1.5 holds from P to P' . For property (b) of Definition 5.1.5 to hold from P' to P , P and P' should be equal. Therefore, $P' \not\prec_{dph} P$.

□

Since \prec_{dph} is a partial order on \mathcal{R} , it has at least one minimal element in \mathcal{R} . We now use \prec_{dph} to introduce the notion of collection.

Definition 5.1.6 (Collection). *Consider the set of classes of a query on a data tree. Let \mathcal{R} be the set of class representatives. A collection is a set L of classes which contains exactly:*

- (a) *a class C whose representative is a minimal element in \mathcal{R} w.r.t. \prec_{dph} , and*
 (b) *all the classes C' such that $repr(C) \prec_{dph} repr(C')$.*

That is, for L and C , we have that $\forall C' \in L, C' \neq C, repr(C) \prec_{dph} repr(C')$ and for every class $C'' \notin L, repr(C) \not\prec_{dph} repr(C'')$ and $repr(C'') \not\prec_{dph} repr(C)$.

Clearly, $repr(C)$ is the least element in the set of representatives of classes in L with respect to \prec_{dph} . We define the representative of collection L to be the representative of class C . That is, $repr(L) = repr(C)$.

There are as many collections for Q on T as there are minimal elements in the set of class representatives \mathcal{R} w.r.t. \prec_{dph} . Note that the collections can overlap. However, they cannot overlap on a representative class (that is, a class whose representative is also the representative of a collection). Therefore, no collection can be included into another collection.

Figure 5.5 shows a collection which includes four classes. One can see that the collection groups together patterns which, even though they are structurally different, they have semantic similarities since they all represent a set of courses and match the keywords

to the titles of these courses. The edges between the classes in the figure correspond to the \prec_{dph} relations between the representatives of these classes (the edge from C_1 to C_4 is not shown as it can be derived by transitivity). Pattern P_4 is the representative of the collection since it is the least element among the representatives of the four classes w.r.t. \prec_{dph} .

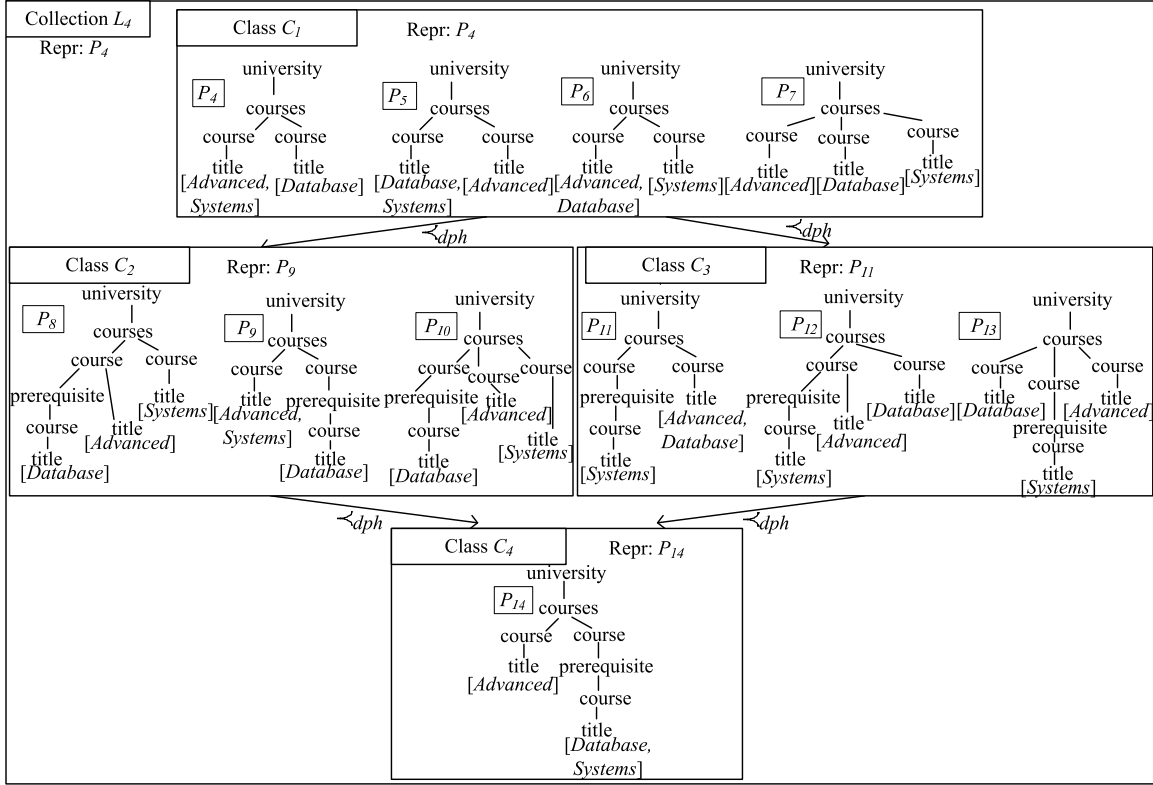


Figure 5.5 A collection of classes.

We present two other sample collections in Figure 5.6. Collection L_1 consists of a single pattern and according to our ranking, is returned as the top collection to the user. Collection L_8 consists of two classes.

5.2 Cluster Ranking and Navigation among Clusters

We now explain how the user proceeds in order to find the relevant results of a query. In order to facilitate this process, we provide techniques for ranking the clusters: all collections are ranked at the top level, classes are ranked within collections, and patterns are ranked

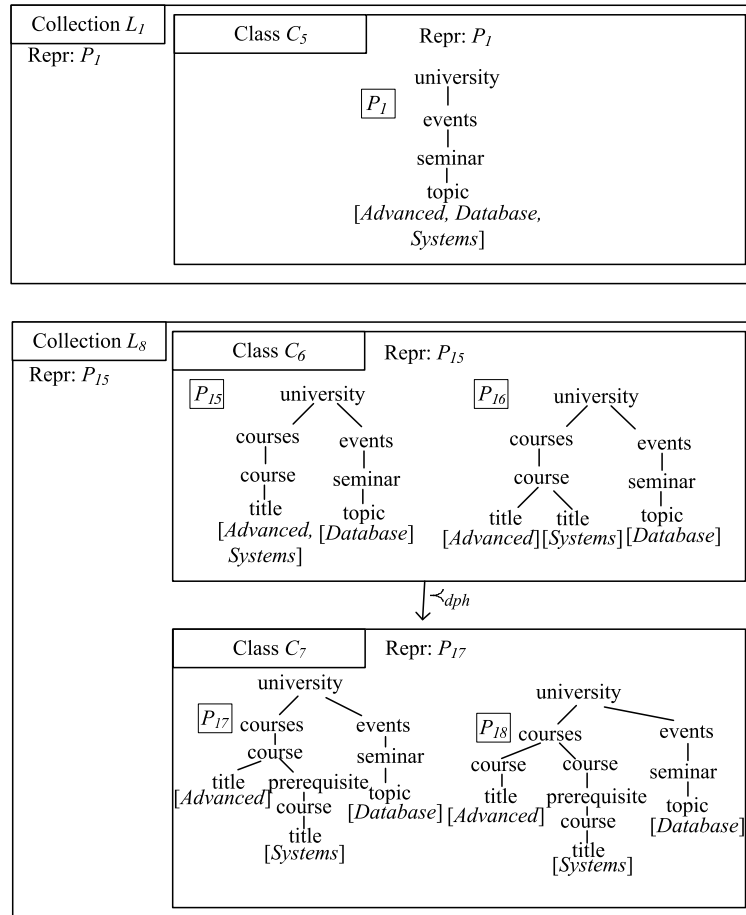


Figure 5.6 Two sample collections

within classes. The goal of the ranking is to present to the user first the clusters which are more likely to contain relevant results in order to reduce the total number of clusters examined by the user in her search for relevant results.

5.2.1 Ranking Patterns

Within a class, the patterns are ranked based on their size in ascending order. Patterns of the same size have the same rank. If the size of a pattern is smaller than the size of another pattern in the same class, it is assumed to be more relevant to the query as it more closely relates the query keyword instances.

5.2.2 Ranking Classes

The \prec_{dph} relation is used to rank the classes within a collection. As stated in Proposition 5.1.2, \prec_{dph} is a partial order. Definition 5.1.6 determines that the representative of a collection is the least element in the set of representatives of classes in the collection with respect to \prec_{dph} . Therefore, every collection is a rooted DAG where the nodes correspond to classes and the directed edges correspond to \prec_{dph} relationships between the representatives of the classes. There is an edge from class C_1 to class C_2 if $repr(C_1) \prec_{dph} repr(C_2)$. The class of the representative of the collection is the unique source node of the DAG. We rank the classes within a collection based on the maximum distance of a node (class) in the DAG from the source node of the DAG. The distance is measured in terms of the number of edges. Classes with smaller maximum distance are ranked higher. The relative order of two classes with the same maximum distance in the DAG is irrelevant. For instance, in the collection depicted in Figure 5.5, $max_dist(C_2) = max_dist(C_3) = 1$ and $max_dist(C_4) = 2$. Therefore, a possible ranking for the classes in this collection is C_1, C_2, C_3, C_4 .

5.2.3 Ranking Collections

The highest level of clustering is formed by the collections. In order to provide an ordering of collections, we define a new relation \prec_{opi} on patterns. The notation *opi* abbreviates “one-path isomorphism”, a term which the next definition justifies.

Definition 5.2.1 (\prec_{opi} relation). *Let P and P' be two patterns of a query on an XML tree. $P \prec_{opi} P'$ iff there are two paths p and p' in P and P' , respectively, from the root to a node annotated by the same keyword such that:*

- (a) *p and p' are isomorphic and*
- (b) *the LCA node of P is a descendant of the LCA node of P' in p .*

The \prec_{opi} relation relates two patterns, P and P' which share a common root-to-annotated-node path. P is assumed to be more relevant than P' since P represents a more

specific relationship of its keyword instances compared to P' : its MCT root (LCA) is located more deeply in the data tree. In Figure 5.7, $P_3 \prec_{opi} P_4$. Because of condition (b) in Definition 5.2.1, relation \prec_{opi} is *acyclic*.

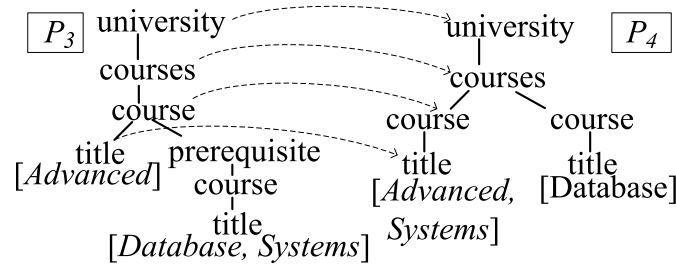


Figure 5.7 One path isomorphism between two patterns P_3 and P_4 .

As with classes, in order to rank collections, we build a graph of collections. The nodes of this graph are collections. There is an edge from collection L_1 to collection L_2 iff $repr(L_1) \prec_{opi} repr(L_2)$. Since \prec_{opi} is acyclic, this graph is a DAG. Figure 5.8 shows a graph of collections for our running example. The representatives of the collections L_1, L_2 and L_3 are the patterns P_1, P_2 and P_3 , respectively, which are shown in Figure 4.1. We use topological ordering to rank the collections, using also the maximum distance from a source node in the DAG to rank incomparable nodes. Further, collections with the same maximum distance are ranked based on (a) the length of the root-to-LCA-path of their representatives (which also reflects the depth of the LCAs in these patterns), and (b) the size of their representatives. Patterns with deeper LCAs are preferred as are patterns with smaller sizes. In both cases, they are assumed to be more relevant as they bring the keyword instances closer. Collections with the same values for all three metrics are ranked randomly. As an example, a possible ranking for the collections of our running example shown in Figure 5.8 is $L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8, L_9, L_{10}$.

5.2.4 Cluster Navigation.

The navigation of the clusters starts at the third level. The user is presented with the list of collections in ranked order. At this point, there are two possible ways a user can choose

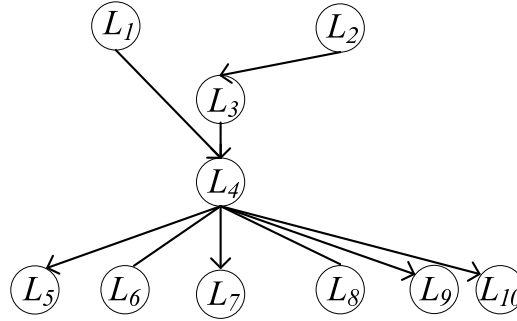


Figure 5.8 Graph of collections for our running example.

to navigate the cluster hierarchy: a depth first traversal (DFT) or a breadth first traversal (BFT). If the user chooses a DFT, she selects a relevant collection and she drills down into it. The classes of the chosen collection are shown to the user in ranked order. The user chooses a class that she finds relevant and the ranked list of the patterns of this class are presented. Finally, the user chooses a relevant pattern to retrieve its results. If more results are desired, the next pattern in the class is examined. If the current pattern is the last pattern in the class, the system backtracks to the next class/collection. The process continues until no more results are desired or the last collection in the hierarchy is examined. If the user chooses a BFT, instead of following a single path in the hierarchy, she drills down level by level. That is, the user chooses collections that are relevant. Then, the classes of these collections are presented ranked in an order which respects the order of the parent collections and the user selects relevant ones among them. At the end, the system presents the patterns of these classes in an order which complies with the order of the parent classes and relevant patterns among them and their results thereof are selected. With BFT, no backtracking is needed as the user can choose multiple clusters at any level.

If all the relevant results in the data tree are sought, both traversal techniques are equally appropriate. However, if at most k ($k \geq 1$) results are sought, DFT is more efficient since the user can avoid examining successor clusters (patterns, classes or collections) of the current cluster in the DFT order as soon as k results are retrieved. With BFT, this is not possible, and the only way to constrain the number of clusters examined is to restrict the

number of clusters selected at every level to k .

5.3 The Algorithm

Algorithm *ClusterStack* is a stack-based algorithm. It takes as input a keyword query and the inverted lists of the query keywords. It computes the results (ITs) and their patterns and concurrently produces the classes of these patterns. Subsequently, it produces and ranks the collections of classes. The computation of the results is performed in a stack-based manner. The classes generated are organized into graphs based on the \prec_{dph} relation to produce collections. These graphs are used to rank the classes within the collections. The last step involves the construction of the graph of collections based on the \prec_{opi} relation on their representatives from which the ranking of the collections is obtained. The body of the algorithm and the stack-based operations performed during the first phase of the computation are shown in Algorithm 4. Some of the helper functions used in Algorithm 4 are presented in Algorithm 5. Aggeliki Dimitriou from NTUA also contributed to the design of the algorithms.

Algorithm 4: ClusterStack

```

1 ClusterStack(IN:  $k_1, \dots, k_n$ : keyword query,  $invL$ : inverted lists)
2    $s \leftarrow \text{new Stack}()$ 
3   while  $n \leftarrow \text{getNextNodeFromInvertedLists}()$  do
4      $\text{annotatedLabelPaths.addIfNotExists}(n.kw, n.labelPath)$ 
5     while  $s.topNode$  is not ancestor of  $n$  do
6        $\text{pop}(s)$ 
7      $\text{push}(s, n)$ 
8   while  $s$  is not empty do
9      $\text{pop}(s)$ 
10   $\text{classCollections} \leftarrow \text{computeCollections}(\text{patternClasses})$ 
11   $\text{collectionsGraph} \leftarrow \text{generateOPIgraph}(\text{classCollections})$ 

```

ClusterStack uses one stack. The entries of the stack correspond to the nodes of the data tree and they are associated with a set of partial and complete patterns. Contrary to

Algorithm 5: ClusterStack

```

1  push(Stack s, Node n)
2    while s.topNode is not parent or self of n do
3       $\sqsubset$  push(s, ancestor or self of n at s.topNode.depth+1, "")
4      newP  $\leftarrow$  new Pattern(n.labelPath, flags.set(id(n.kw)))
5      newPid  $\leftarrow$  checkIfExistsOrAddToPatterns(newP)
6      newPatternIds  $\leftarrow$  composePatterns(s.top.patterns, newPatternId)
7      s.top.PatternIds  $\leftarrow$  union(s.top.PatternIds, newPatternIds)
8  pop(Stack s)
9    for curPid  $\leftarrow$  s.top.patterns.next() do
10     curP  $\leftarrow$  patterns.get(curPid)
11     if curP is complete then
12       s.top.removePatternId(curPid)
13       completePatterns.add(curPid)
14       curP.addLCA(s.top.dewey())
15       if curP.LCApath and curP.signature are not in patternClasses then
16         newclass  $\leftarrow$  new PatternClass(curP)
17          $\sqsubset$  patternClasses.add(newClass)
18         newclass.add(curPid)
19     else
20        $\sqsubset$  childPatterns.add(extendToParent(curPid))
21   s.pop()
22   newPatternIds  $\leftarrow$  composePatterns(s.top.patterns, childPatterns)
23   s.top.add(newPatternIds)
24  composePatterns(patternIdsA, patternIdsB)
25    foreach PatternIdsA as idA do
26      patA  $\leftarrow$  patterns[idA]
27      foreach currentPatternIdsB as idB do
28        patB  $\leftarrow$  patterns[idB]
29        if patA.kwFlags AND patB.kwFlags = 0 then
30          if jointPatterns[min(idA,idB),max(idA,idB)] is set then
31             $\sqsubset$  idAB  $\leftarrow$  jointPatterns[min(idA,idB),max(idA,idB)]
32          else
33            patAB  $\leftarrow$  joinRoots(patA,patB)
34            idAB  $\leftarrow$  checkIfExistsOrAddToPatterns(patAB)
35             $\sqsubset$  jointPatterns[min(idA,idB),max(idA,idB)]  $\leftarrow$  idAB
36            newPatterns.add(idAB)
37     $\sqsubset$  return newPatterns
38  joinRoots(patternA, patternB)
39    patAB  $\leftarrow$  patB.replaceRoot(patA.root)
40    patAB.kwFlags  $\leftarrow$  patA.kwFlags OR patB.kwFlags
41    patAB.size  $\leftarrow$  patA.size + patB.size

```

partial patterns, complete patterns are annotated by all the query keywords. The inverted lists of the keywords contain for each keyword instance: (a) the Dewey code [78] of the instance, and (b) its encoded root-to-instance label path in the data tree. The Dewey encoding scheme is convenient for stack based algorithms where a tree node corresponds to a stack entry and consecutive entries reflect parent-child relationships between two nodes. For instance, a stack entry corresponding to a node with Dewey code 1.2.3.4 is preceded by a stack entry corresponding to the node 1.2.3 lower in the stack. Popping of a node from the stack corresponds to the removal of the last component from a Dewey code. The label path of an instance is encoded by assigning a unique integer id to each label.

Keyword query results and their patterns are computed in the while loops of lines 3-7 and 8-9 in Algorithm 4. Each keyword instance in the inverted lists is pushed into the stack in document order. If the instance's annotated label path has not been seen before, it is stored in the `annotatedLabelPaths` array (line 4 in in Algorithm 4). In order to push an instance into the stack, the top stack entry must correspond to the parent of the instance in the data tree. This requirement entails a number of pop actions of entries corresponding to nodes that are not ancestors of the node to be pushed (lines 5-6 in Algorithm 4). Then, in the *push* procedure (lines 1-7 in Algorithm 5), for all ancestors of the node to be pushed which are not present in the stack, an empty stack entry is pushed into the stack (lines 2-3 in Algorithm 4). The push action of the new instance entails the construction of an initial pattern consisting of the instance's label path annotated by the instance's keyword (lines 4-5 in Algorithm 5). If the stack entry of the node to be pushed already exists, it may contain a number of patterns. In this case, the new one-path pattern is examined for possible combination with patterns already in the top stack entry (line 6 in Algorithm 5). If a new set of patterns is produced, it is unioned with the set of the existing ones in this stack entry (line 7 in Algorithm 5). Finally, when the last node of the inverted lists is processed, the stack is emptied through a series of pop actions (lines 8-9 in Algorithm 4).

Query results and their patterns are constructed within procedure *pop* (lines 8-23 in

Algorithm 5). If complete patterns exist in the top stack entry, they are removed from the stack, and they are added to the set of complete patterns together with the LCA nodes of their ITs (lines 12-14 in Algorithm 5). Pattern classes are generated along with patterns, according to Definition 5.1.2: a class is defined for each root-to-LCA label path and for each set of LCA-to-keyword instance label paths. *ClusterStack* encodes label paths with unique ids. These are used to build a signature for each pattern which reveals the pattern's root-to-leaf paths. If the class of a complete pattern does not exist, it is generated, and this pattern becomes its first member (lines 15-18 in Algorithm 5). The partial patterns of the top stack entry are propagated to the parent node entry with the appropriate adjustments (line 20 in Algorithm 5) and they are combined with patterns of that entry to produce new ones (lines 22-23 in Algorithm 5).

The composition of patterns (lines 24-37 in Algorithm 5) consists of merging their roots (line 49 in Algorithm 5). The composed pattern obtains the annotations of the component patterns and its size becomes the sum of the component sizes (lines 40-41 in Algorithm 5). The generation of new patterns throughout the algorithm *ClusterStack* is performed either by *composePatterns* (lines 6, 22 in Algorithm 5) or by *extendToParent* (line 20 in Algorithm 5). In order to avoid the repeated construction of the same pattern from the same component patterns, two variables, namely *jointPatterns* and *parentPatterns* are consulted before the construction of a new pattern in the two procedures, respectively.

After the pattern construction phase is completed, the processing moves on by calling procedure *computeCollections* (line 10 in Algorithm 4) which creates and ranks collections of the generated classes based on the \prec_{dph} relation. Procedure *computeCollections* is shown in Algorithm 6. Initially, the annotated label paths of each keyword are pairwise compared to identify existing descendant path homomorphisms. The outcome of this process is stored in the variable *pathHomomorphisms* (line 3) and is exploited in discovering \prec_{dph} relationships between classes. *ClusterStack* avoids the exhaustive examination of all pairs of classes for finding \prec_{dph} relationships. Instead, it prunes the search space based on

the following remark.

Algorithm 6: computeCollections procedure

```

1 computeCollections(patternClasses, annotatedLabelPaths)
2   collections  $\leftarrow \emptyset$ 
3   descPathHomomorphisms  $\leftarrow$  generateDPH(annotatedLabelPaths)
4   for each lca in classes do
5     lcaClasses  $\leftarrow$  getLcaClasses(classes, lca)
6     for each maxPatternSize observed in lcaClasses, in ascending order do
7       sizeClasses  $\leftarrow$  getSizeClasses(lcaClasses, maxPatternSize)
8       if maxPatternSize is the minimum in the sizeClasses then
9         for each cl in sizeClasses do
10           col  $\leftarrow$  new Collection(cl)
11           collections.add(col)
12       else
13         connected  $\leftarrow$  false
14         for each childClass in sizeClasses do
15           for each mps < mps(childClass) in descending size order do
16             smallerSizeClasses  $\leftarrow$  getSizeClasses(lcaClasses, mps)
17             for each parentClass in smallerSizeClasses do
18               if parentClass.hasDPHRto(childClass) then
19                 parentClass.connectTo(childClass)
20                 childClass.rank  $\leftarrow$  parentClass.rank+1
21                 connected  $\leftarrow$  true
22             if not connected then
23               col  $\leftarrow$  new Collection(childClass)
24               collections.add(col)
25   return collections

```

We first define the *maximal pattern size* of a class C . For a pattern P in a class C , let l be the root-to-LCA path in P and p_i , $i = 1, \dots, n$, the LCA-to-annotated-node path, for each one of the n annotating keywords in the pattern. Then, the maximal pattern size of C is

$$mps(C) = length(l) + \sum_{i=1}^n length(p_i)$$

The $mps(C)$ is the maximum possible size for a pattern in C and corresponds to the pattern whose LCA-to-annotated-node paths do not share any node other than the LCA. This

pattern may or may not appear in C .

Remark 5.3.1. Let $\text{repr}(C_1)$ and $\text{repr}(C_2)$ be the representative patterns of two distinct classes C_1 and C_2 of the same collection. If $\text{repr}(C_1) \prec_{dph} \text{repr}(C_2)$ then $\text{mps}(C_1) < \text{mps}(C_2)$.

Procedure *computeCollections* in Algorithm 6 partitions the classes based on their root-to-LCA paths (lines 4-5). Further, it partitions classes with the same root-to-LCA path based on their *mps* (lines 6-7). Leveraging Remark 5.3.1, *computeCollections* checks the existence of \prec_{dph} relationships only between classes with the same root-to-LCA path and different *mps*. All the classes with the same root-to-LCA path are examined in ascending *mps* order (lines 6-7). For each class with minimum *mps*, one collection is created (lines 8-11), since according to Remark 5.3.1 these classes are minimal elements w.r.t. \prec_{dph} . In order to find \prec_{dph} relationships between classes, a class is compared only with classes of smaller *mps* in descending *mps* order (lines 15-21). If such a relationship is discovered, the classes are connected with an edge (line 19) and the class with the larger *mps* is attached to the collections of the class with the smaller *mps*. The rank of the newly connected class is set equal to the rank of the parent class increased by 1 (line 20). If a class cannot be connected to any class of smaller *mps*, a new collection for this class is created (lines 22-24).

Quite similarly, procedure *generateOPIgraph* constructs the graph of collections based on the \prec_{opi} relation, and uses it to produce the ranking of collections. It proceeds by examining collections in ascending LCA depth order. Checking for ancestor-descendant relationships between the LCAs of a pair of patterns further prunes the number of comparisons needed for discovering \prec_{opi} relationships between collections. The listing of the *generateOPIgraph* is omitted for brevity.

Analysis. The time cost of *ClusterStack* involves the cost of the pattern and class generation phase and the cost of the cluster hierarchy construction phase. During the first phase which is stack based, the push and pop actions of the instances of the inverted lists determine the

time complexity. Let k be the number of keywords of a query and L_i be the inverted list of the i th keyword. The total number of keyword instances of the query is $|L| = \sum_i |L_i|, i \in [1, k]$. Each insertion of an instance from the inverted lists may require at most h pops from and h pushes onto the stack, with h denoting the height of the data tree. If there are p partial patterns in the top stack entry, the single path pattern of the instance pushed into the stack may be combined with at most p patterns, which takes $O(p)$ time. When an entry is popped from the stack, all its partial patterns are extended with an edge to their parent node, which will become the new top entry of the stack. Assuming that there are no complete patterns in the popped entry, this action needs $O(p)$ time. The partial patterns are also combined with the partial patterns of the parent node to produce new patterns and this takes $O(p^2)$. The whole stack-based process takes $O(|L|hp^2)$.

During the generation of the graphs for classes and collections, each class (collection) under consideration is connected with edges to other classes (collections). The number of classes is determined by the number of different label paths to the keyword instances and the height of the data tree. If l is the maximum number of distinct label paths per keyword, the maximum number of classes that can be produced is hl^k , i.e., the number of combinations of all label paths with all possible LCAs at all depths. The maximum number of distinct LCAs is hl , which means that *computeCollections* examines groups of at most l^{k-1} classes. The number of class comparisons in each group is maximized when there all possible max pattern sizes occur in the group. In this case, $l^{k-1}k^2l^2$ comparisons are performed for each group. Thus, the time cost of *computeCollections* is $O(hk^2l^k)$. Procedure *generateOPIgraph* shows similar time complexity in the worst case. This happens when a distinct collection is produced for each class. Since the stack-based phase dominates the time cost of *ClusterStack*, its time complexity is $O(|L|hp^2)$.

5.3.1 Running Example of ClusterStack

In this section, we present a running example of ClusterStack where the query $\{Advanced, Database, Systems\}$ is issued against the data tree T depicted in Figure 1.3. The keyword instances of the inverted lists are listed in Table 5.1. Table 5.1 also shows the label paths of the instances as well as an abbreviated representation of the annotated label paths which are used in the following discussion for the sake of space.

Table 5.1 Inverted Lists of Keywords in Query $\{Advanced, Database, Systems\}$ on the Data Tree T of Figure 1.3

Keyword	ID	Label path	Abbreviated annotated label path
Advanced	8	university.courses.course.title	u.cs.c.t [Advanced]
	10	university.courses.course.title	u.cs.c.t [Advanced]
	12	university.courses.course.title	u.cs.c.t [Advanced]
	23	university.events.seminars.seminar.topic	u.e.ss.s.to [Advanced]
Database	8	university.courses.course.title	u.cs.c.t [Database]
	10	university.courses.course.title	u.cs.c.t [Database]
	25	university.courses.course.prerequisite.course.title	u.cs.c.p.c.t [Database]
	23	university.events.seminars.seminar.topic	u.e.ss.s.to [Database]
Systems	8	university.courses.course.title	u.cs.c.t [Systems]
	10	university.courses.course.title	u.cs.c.t [Systems]
	25	university.courses.course.prerequisite.course.title	u.cs.c.p.c.t [Systems]
	23	university.events.seminars.seminar.topic	u.e.ss.s.to [Systems]

Algorithm ClusterStack uses a stack. Push actions on the stack correspond to traversing the data tree downwards and pop actions upwards. Figure 5.9 shows three states of the stack during the processing of the inverted lists shown in Table 5.1. Each entry in the stack corresponds to a node of the data tree and contains the patterns constructed in that node based on the patterns of its subtree. Consecutive entries in the stack correspond to nodes in the data tree that satisfy a parent-child relationship (e.g., in Figure 5.9 in state 4 of the stack, node 8 is a child of node 4). Popping a node from the stack results in combining its patterns with patterns of its parent which have been constructed from the subtrees of the parent already processed.

For our running example, the algorithm proceeds examining keyword instances

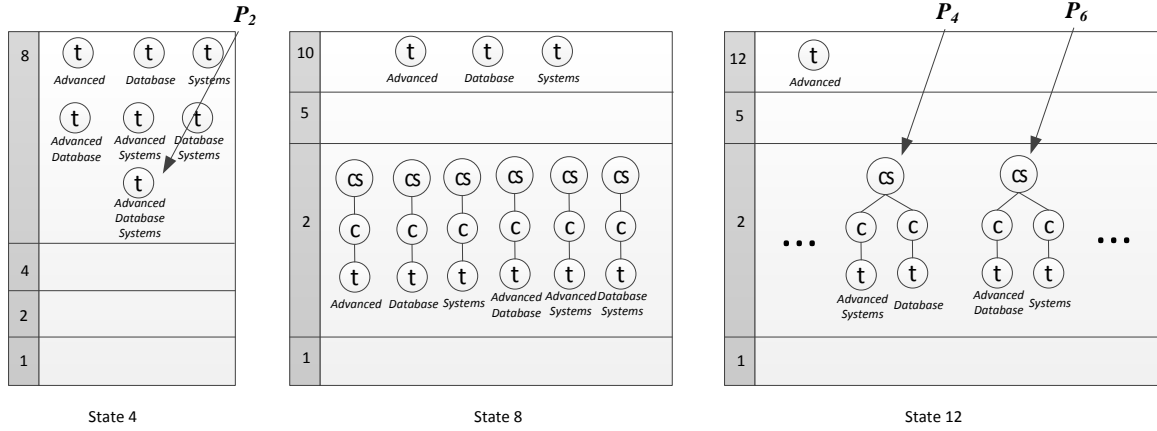


Figure 5.9 Stack states

in document order, i.e., in the order 8, 10, 12, 25 and finally 23. Once finished with node 8, ClusterStack produces the first result, which is the IT 1.2.4.8. The corresponding pattern P_2 shown in Figure 5.1 is also constructed based on the annotated label paths `u.cs.c.t [Advanced]`, `u.cs.c.t [Database]` and `u.cs.c.t [Systems]`. Figure 5.9 shows this pattern in state 4 of the stack. Note that each pattern is displayed without the path from the root of the data tree to the LCA of the pattern to avoid cluttering the figure. For each distinct combination of annotated label paths in a pattern and its LCA label path, the algorithm creates a pattern class. In our example, the first class is created for the combination of the three annotated label paths examined so far and the LCA label path `university.courses.course.title (u.cs.c.t in short)`. In a similar way, the algorithm proceeds to produce the IT 1.2.5.10 for the instances of the keywords at node 10. This IT also conforms to the pattern P_2 . No new patterns or pattern classes are created. Next, the keyword instances at nodes 12 and 25 are examined, and new patterns and pattern classes are produced.

After finishing with the instances at node 25, ClusterStack pops nodes from the stack backtracking to the root in order to reach the last node with keyword instances, which is node 23. While the processing is at node 2, several combinations of the instances located in the subtree of node 2 are performed to produce different ITs with LCA 1.2. Some of the corresponding patterns are depicted in entry 2 of state 12 of the stack (Figure 5.9).

They are shown in detail in Figure 7. All these patterns share the same LCA label path (`university.courses`). A different selection of annotated label paths designates a different class. Table 5.2 shows the details:

Table 5.2 Patterns Under the Node `courses` of the Data Tree T

Class	Patterns	Annotated label paths	Representative
C_1	P_4, P_5, P_6, P_7	u.cs.c.t [Advanced] u.cs.c.t [Database] u.cs.c.t [Systems]	P_4
C_2	P_8, P_9, P_{10}	u.cs.c.t [Advanced] u.cs.c.p.c.t [Database] u.cs.c.t [Systems]	P_9
C_3	P_{11}, P_{12}, P_{13}	u.cs.c.t [Advanced] u.cs.c.t [Database] u.cs.c.p.c.t [Systems]	P_{11}
C_4	P_{14}	u.cs.c.t [Advanced] u.cs.c.p.c.t [Database] u.cs.c.p.c.t [Systems]	P_{14}

When the root of the tree is popped from the stack, all patterns under the root `university` of the data tree T have been constructed and organized in classes according to the annotated label paths of their keywords. Procedures *computeCollections* and *generateOPIgraph* are then executed. Suppose that the classes shown in Figures 5.5 and 5.6 have been constructed. Table 5.3 shows for each of these classes, the representative pattern, the label path of the LCA of the class, the annotated label paths for the query keywords and the maximum pattern size (mps) of the class.

Procedure *computeCollections* identifies first descendant path homomorphisms between annotated label paths of the same keyword. These homomorphisms will be used later for determining descendant path homomorphism relations between classes. The correspondences listed in Table 5.4 are stored in the array `descPathHomorphisms`.

Procedure *computeCollections* continues in steps as shown in Table 5.5. For each LCA path and for each maximum pattern size in ascending order, the algorithm examines

Table 5.3 Pattern Classes of Q on T

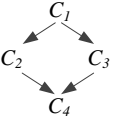
Class	Repr.	Ann. label paths	LCA path	mps
C_1	P_4	u.cs.c.t [Advanced] u.cs.c.t [Database] u.cs.c.t [Systems]	u.cs	7
C_2	P_9	u.cs.c.t [Advanced] u.cs.c.p.c.t [Database] u.cs.c.t [Systems]	u.cs	9
C_3	P_{11}	u.cs.c.t [Advanced] u.cs.c.t [Database] u.cs.c.p.c.t [Systems]	u.cs	9
C_4	P_{14}	u.cs.c.t [Advanced] u.cs.c.p.c.t [Database] u.cs.c.p.c.t [Systems]	u.cs	11
C_5	P_1	u.e.ss.s.to [Advanced] u.e.ss.s.to [Database] u.e.ss.s.to [Systems]	u.e.ss.s.to	3
C_6	P_{15}	u.cs.c.t [Advanced] u.e.s.to [Database] u.cs.c.t [Systems]	u	9
C_7	P_{17}	u.cs.c.t [Advanced] u.e.ss.s.to [Database] u.cs.c.p.c.t [Systems]	u	11

Table 5.4 Descendant Path Homomorphisms

Keyword	Descendant path homomorphism	
Advanced	2	u.e.ss.s.to [Advanced] \xrightarrow{dph} u.e.ss.s.to [Advanced]
Database	4	u.cs.c.t [Database] \xrightarrow{dph} u.cs.c.p.c.t [Database]
	6	u.e.ss.s.to [Database] \xrightarrow{dph} u.e.ss.s.to [Database]
Systems	8	u.cs.c.t [Systems] \xrightarrow{dph} u.cs.c.p.c.t [Systems]
	10	u.e.ss.s.to [Systems] \xrightarrow{dph} u.e.ss.s.to [Systems]

if there are any descendant path homomorphisms between the representative patterns of the class under examination and any class of smaller mps (maximum pattern size). If such a relationship exists between any two classes, an edge is added to connect them in the graph

Table 5.5 ComputeCollections Running Example

LCA path	mps	New collection	Assigned classes	Collection graph
u.e.ss.s.to	3	L_1	C_5	C_1
u.cs	7 9 9 11	L_4	C_1 C_2 C_3 C_4	
u	9 11	L_8	C_6 C_7	$C_1 \ C_2$

of the collection. If no such relationship is discovered, no edge is added to the graph. This is the case of collection L_8 in Table 5.5.

Finally, procedure *generateOPIgraph* computes the graph of collections in a similar manner based on relation \prec_{opi} . The graphs of the clusters are used to produce their rankings.

5.4 Experimental Evaluation

We performed experiments to assess the retrieval effectiveness of our clustering methodology and also the efficiency of our algorithm. We also compare the quality of our clustering to that of a recent state-of-the-art approach, XMean [53].

We use the DBLP¹, Mondial², SIGMOD² and NASA datasets². Statistics for these datasets are depicted in Table 5.6. For the effectiveness experiments, we use Mondial and SIGMOD datasets which are often used for this purpose in XML keyword search research [6, 53]. We used DBLP and NASA datasets for the scalability experiments which are larger. In order to further increase the size of NASA dataset, we replicated it 40 times under the same root. The experiments were conducted on a 2.9 GHz Intel Core i7 machine with 3 GB memory running Ubuntu.

¹<http://dblp.uni-trier.de/xml/>

²<http://www.cs.washington.edu/research/xmldatasets/>

Table 5.6 Mondial, SIGMOD, DBLP and NASA Dataset Statistics.

	Mondial	SIGMOD	DBLP	NASA
Size	1 MB	467 KB	1.15 GB	956 MB
# nodes	69,846	15,263	34,141,216	21,318,481
# distinct tags	50	12	44	70
# distinct label paths	119	12	196	111
Average depth	3.00	4.60	1.93	4.56
Maximum depth	5	6	5	7

We first introduce the metrics we use for the experimental evaluation; then, we present our results for the effectiveness of the clustering methodology and finally we present our efficiency experiments.

5.4.1 Metrics

As also pointed out in [53], standard IR metrics such as precision and recall are not suitable to assess the retrieval effectiveness of a hierarchical interface with user input. For this purpose, we adapt the *reach time* metric originally introduced in [40]. The reach time is defined to quantify the time spent by a system user locating his/her desired results. We assume that all the results (ITs) that comply with the same pattern are equally interesting to the user and if a pattern is found to be relevant all its results are retrieved. The value of reach time might vary depending on the system interface and the retrieval scenario under consideration. As discussed in Section 5.2.4, the user navigates through our cluster hierarchy starting from the collections. We assume that the hierarchy is traversed in a depth first manner. In order to assess the effect of ranking on reach time, we consider two different variations of our system interface: (a) where clusters are presented to the user in a ranked order using the ranking criteria we introduced in Section 5.2, and (b) where clusters are ranked arbitrarily.

We formally define the reach time as shown below:

$$t_{reach} = \left(\sum_{i=1}^h n_i \right) \quad (5.1)$$

where n_i stands for the number of clusters that are examined by the user at level i , and h is the number of levels in the hierarchy. In the case of our system, h is equal to 3. Examining a cluster consists of checking its representative to decide if the cluster contains relevant results or not.

The value of n_i depends on the retrieval scenario and also on whether the clusters are ranked or not. We compute the value of n_i as follows for different scenarios:

Retrieving all relevant patterns and results. The users has to examine all the top-level clusters and then, recursively examine all the child clusters of clusters that are identified as relevant. In this case, the ranking of the clusters is insignificant since it does not affect t_{reach} .

Retrieving at most k relevant patterns. Since the users follow a DFT, they can stop examining any further clusters after they identify k relevant patterns. Given that t_{reach} depends on the order of presentation of the clusters to the user, we compute the *minimum*, *maximum* and *expected* values for n_i by considering all the possible orderings of clusters at level i under the same parent cluster. If the clusters are ranked, we produce only orderings which comply with the ranking (by changing only the position of clusters with the same rank).

In order to determine the ground truth for the effectiveness experiments, we employed five expert users, not involved in this project, who characterized the query patterns as relevant or irrelevant to the query. The relevancy of each pattern (relevant or irrelevant) was determined by the majority of the characterizations of the expert users.

5.4.2 Effectiveness Experiments

For the effectiveness experiments, we compare our system (referred to as *Result Tree Cluster*—*RTCluster*) with *XMean* [53] which is a recent hierarchical clustering methodology of XML results. *XMean* also extracts the patterns of the results and constructs an initial cluster hierarchy by structurally relaxing the patterns (through the removal of nodes and edges in all possible ways up to the root of the pattern). This is a graph called relaxation graph. It subsequently decreases the size of the hierarchy by reducing it into a hierarchy tree. It exploits DTD information to characterize nodes in the data tree as entity nodes to filter out results. In order to allow a fair comparison of the clustering methodologies of our system and *XMean*, in our implementation of *XMean* we disable the entity node characterization which is orthogonal to the clustering process. This way the two systems are compared over the same result set. As pointed out in [4, 79], results whose LCA coincides with the root of the data tree are meaningless as they relate the keyword instances loosely. Therefore, in the experiments, we consider only patterns whose MCT root (LCA) is not the root of the data tree.

Retrieving all relevant results. We run 20 queries (shown in Tables 4.4 and 4.5) on the Mondial and SIGMOD datasets. All of the queries are chosen from previous papers by different authors [4, 53, 54, 55, 57]. The reach time values for the *RTCluster* and *XMean* approaches for all the queries on the Mondial and SIGMOD datasets are presented in Figures 5.10 and 5.11, respectively. As one can see from the experimental results, our approach outperforms *XMean* on almost all the queries of both datasets achieving lower reach time values. This shows that the clustering hierarchy produced by *RTCluster* better helps the users to retrieve their results of interest. The average reach time values over all the queries for both approaches on the Mondial and SIGMOD datasets displayed in Table 5.7 confirm this observation.

Retrieving at most k relevant patterns. In this scenario, the ranking of the clusters affects

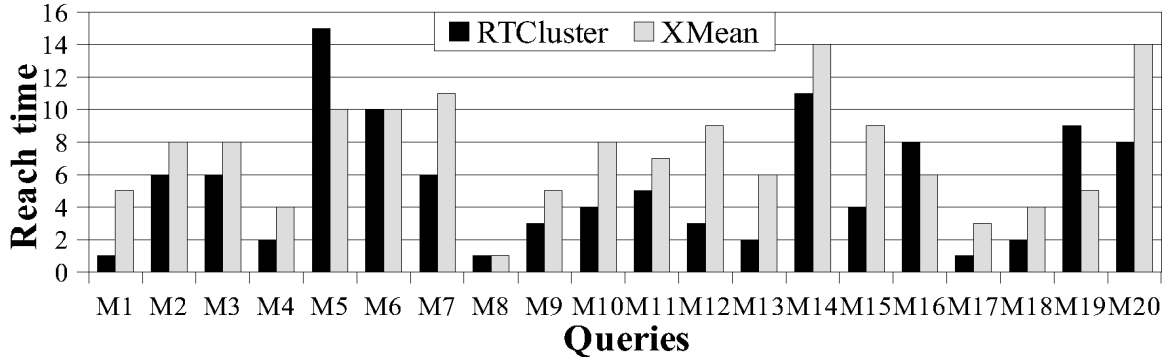


Figure 5.10 Reach time values (for retrieving all relevant patterns) for the queries of Table 4.4 on the Mondial dataset.

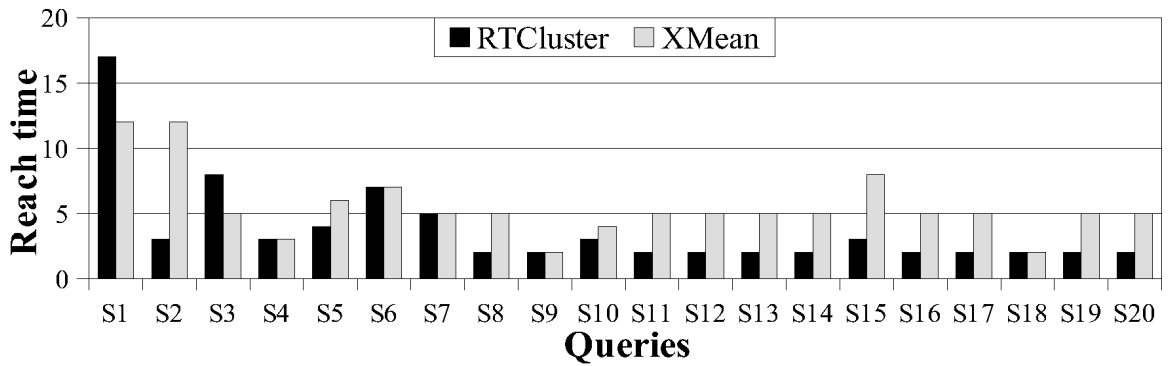


Figure 5.11 Reach time values (for retrieving all relevant patterns) for the queries of Table 4.5 on the SIGMOD dataset.

Table 5.7 Average Reach Time Values (for Retrieving All Relevant Patterns) and Hierarchy Sizes for the Queries of Tables 4.4 and 4.5

Dataset	Approach	Average Reach time	Average Hierarchy size
Mondial	<i>RTCluster</i>	5.35	14.60
	<i>XMean</i>	7.35	23.30
SIGMOD	<i>RTCluster</i>	3.75	33.35
	<i>XMean</i>	5.55	142.25

the values of reach time. Therefore, we first compare retrieving at most k relevant patterns with and without cluster ranking in order to assess the effect of our ranking technique on reach time. Figures 5.12a and 5.12b provide average reach time values for these two variations of *RTCluster* on the queries of Tables 4.4 and 4.5 on the Mondial and SIGMOD datasets, respectively, for $k = 1$ and $k = 2$. Recall that a single pattern can contain multiple results. For each variation, three reach time values are given: minimum, expected and

maximum. As one can see, the ranking of the clusters reduces the average expected reach time by almost 50% on the average on both datasets. This improvement is even more visible on the maximum reach time. Therefore, our ranking technique substantially improves the quality of our clustering system by successfully pruning the number of representatives the user has to examine.

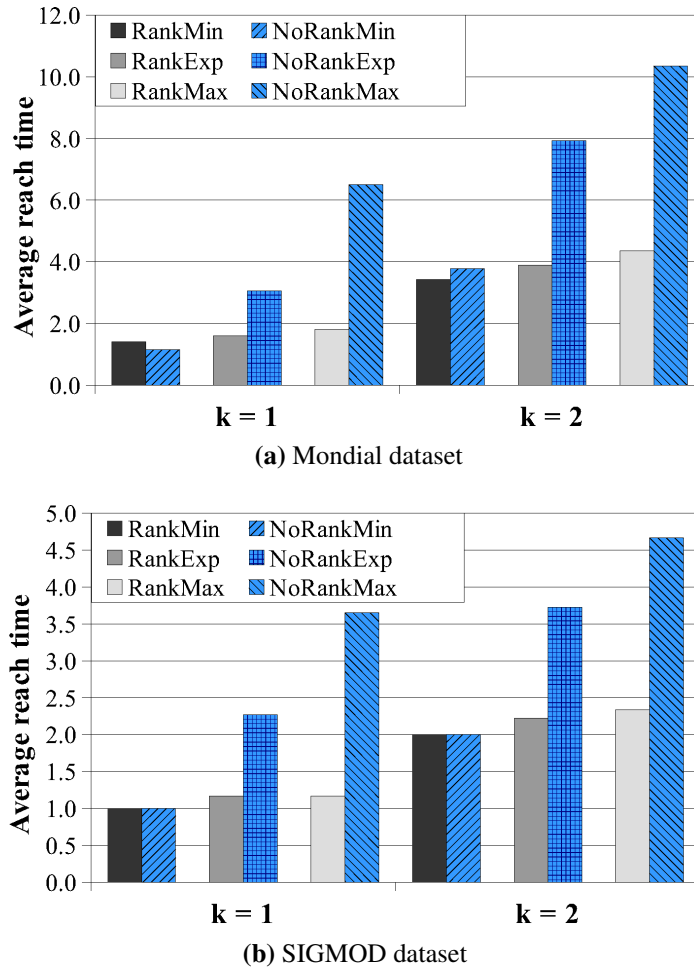


Figure 5.12 Average min, exp and max reach time values (for retrieving at most k patterns) for *RTCluster* with and without ranking of the clusters for the queries of Tables 4.4 and 4.5.

We also compare the effectiveness of *RTCluster* (with cluster ranking) and *XMean* in retrieving at most k relevant patterns. Figures 5.13a and 5.13b show the average reach time values (min, exp and max) for *RTCluster* and *XMean* for $k = 1$ and $k = 2$ on the queries of Tables 4.4 and 4.5 on the Mondial and SIGMOD datasets, respectively. As in the retrieve-all-relevant-patterns scenario, *RTCluster* outperforms *XMean* in all cases.

However, the improvement over *XMean* is more pronounced in the retrieve-at-most- k -relevant-patterns scenario since, as shown in the previous paragraph, our ranking technique (a feature not available in *XMean*) boosts the effectiveness of *RTCluster*.

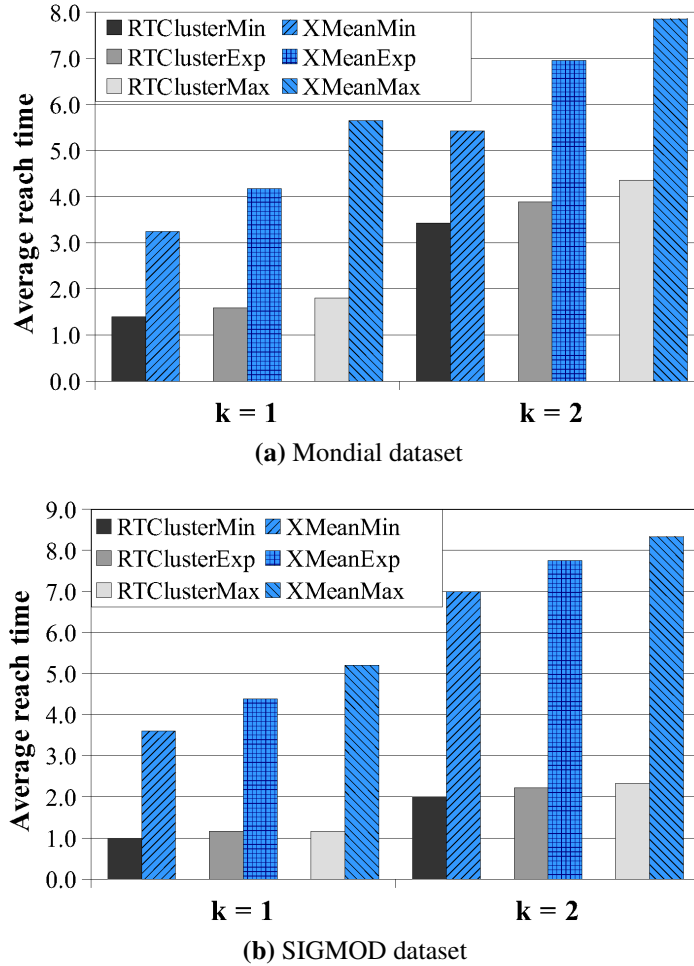


Figure 5.13 Average min, exp and max reach time values (for retrieving at most k patterns) for *RTCluster* and *XMean* for the queries of Tables 4.4 and 4.5.

Clustering hierarchy size. We also compare the effectiveness of the two approaches in terms of the size of the hierarchies constructed. The size of a hierarchy is expressed by the number of nodes (clusters) it contains. Figures 5.14 and 5.15 present the size of the hierarchies constructed by *RTCluster* and *XMean* for the queries of Tables 4.4 and 4.5. Table 5.7 provides average sizes over all the queries. Note that, the y-axis in Figure 5.15 is in logarithmic scale. The hierarchy sizes constructed by *RTCluster* are significantly smaller

than those of *XMean* in most cases. *XMean* usually generates a large hierarchy of relaxed patterns even from a small number of base patterns. Instead, *RTCluster* generates a smaller hierarchy and allows the user to navigate faster to the relevant patterns.

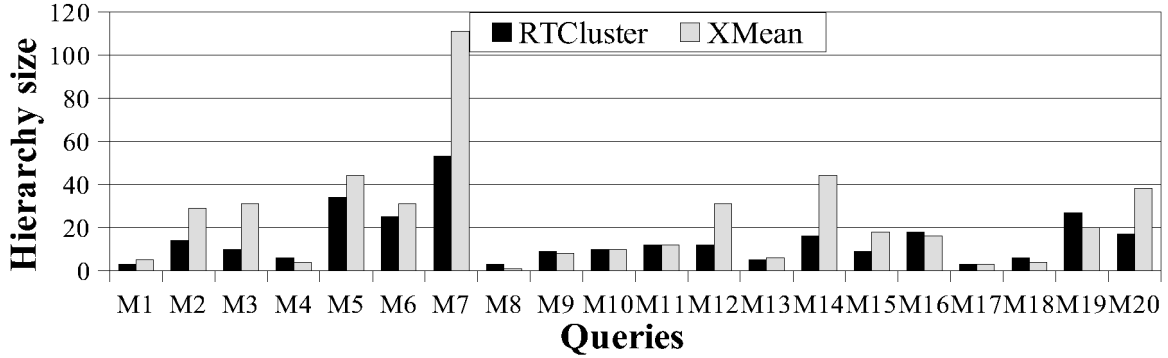


Figure 5.14 Hierarchy sizes constructed by *RTCluster* and *XMean* for the queries of Table 4.4 on the Mondial dataset.

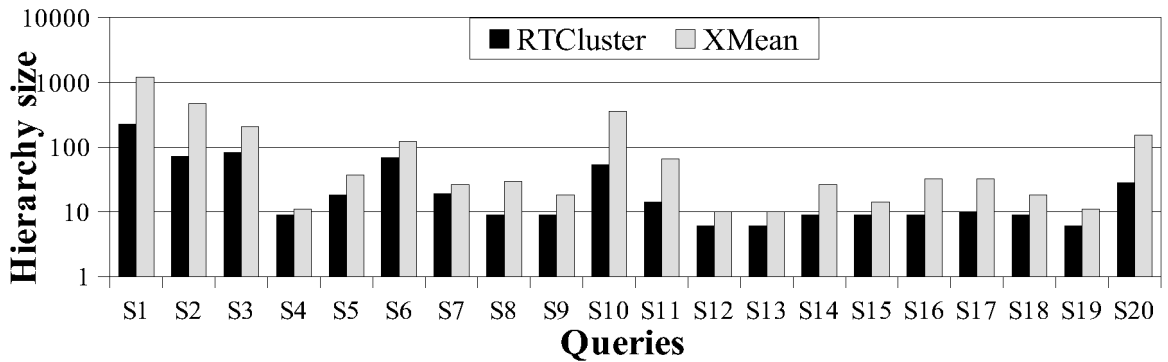


Figure 5.15 Hierarchy sizes constructed by *RTCluster* and *XMean* for the queries of Table 4.5 on the SIGMOD dataset.

5.4.3 Efficiency Experiments

In order to evaluate the efficiency of our algorithm, we measured the computation time of queries and we ran experiments to study how the computation time scales with respect to the number of keywords and the input size.

Computation time. Figures 5.16 and 5.17 present the computation times of *ClusterStack* for the queries of Table 4.4 and 4.5 on the Mondial and SIGMOD datasets, respectively.

The computation time is the time needed by our algorithm for generating the results, clustering them and building the cluster hierarchy after the query is issued. As one can see, all the measured computation times are smaller than half of a second on both datasets. This performance is comparable to that of real life systems even though no optimizations of a commercial system have been applied.

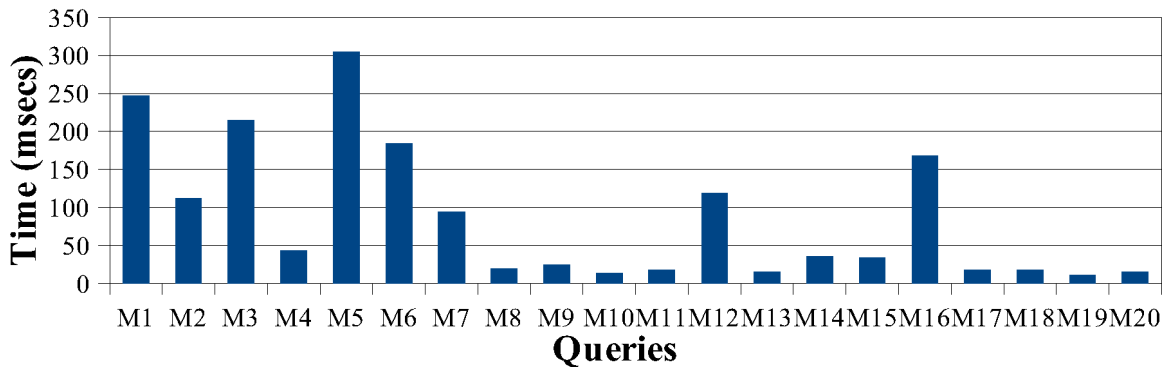


Figure 5.16 Computation time (in msec) for the queries of Table 4.4 on the Mondial dataset.

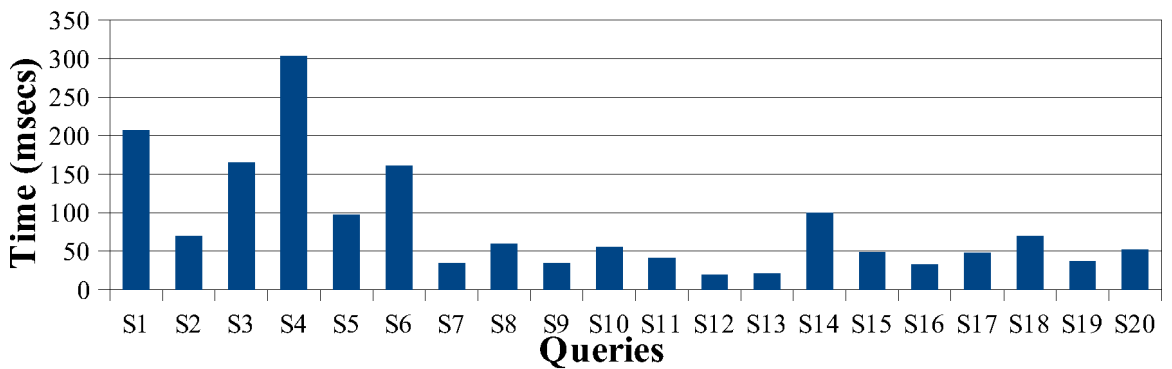


Figure 5.17 Computation time (in msec) for the queries of Table 4.5 on the SIGMOD dataset.

Scalability. For the scalability experiments, we used the DBLP and NASA datasets which are larger and they have long keyword inverted lists that can be truncated.

In order to study the scalability with respect to the number of keywords experiments, we randomly generated 10 queries with 7 keywords each on both datasets. For each of these queries, we constructed six subqueries containing from two to seven keywords by gradually removing keywords. In total, for each query cardinality from two to seven, we

obtained a set of 10 queries. Figure 5.18 presents the average computation times for each query cardinality over the 10 queries in the corresponding set. As one can see in the figure, the computation time scales smoothly increasing from 300 to less than 800 milliseconds for DBLP and from 700 to less than 1200 milliseconds for NASA when the number of keywords varies from 2 to 7 keywords.

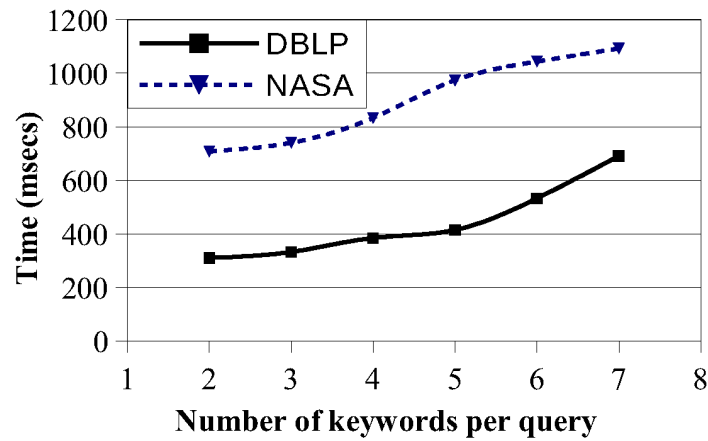
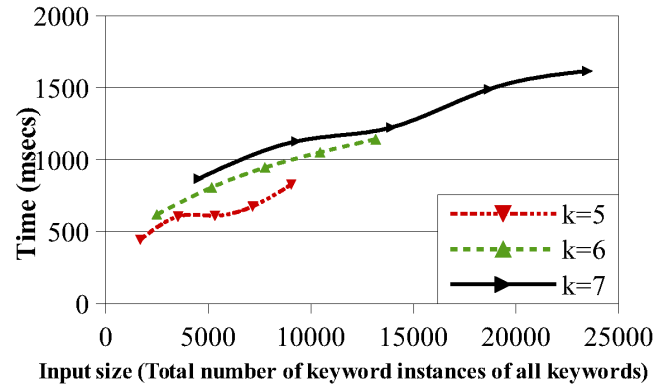
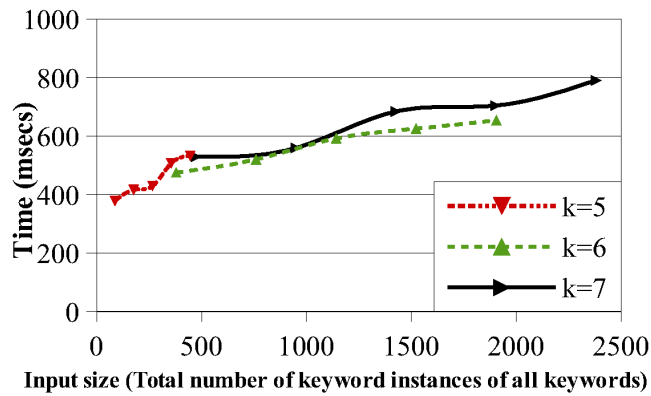


Figure 5.18 Average computation time vs. number of keywords of *ClusterStack* using queries with 2 to 7 keywords on the DBLP and NASA datasets.

In order to study scalability with respect to the input size, we measured the computation time in relation to the total number of keyword instances in the data tree for all the query keywords. We randomly chose the two sets of 7 keywords $\{anwendung, karin, wi32, ft, wirtz, mullerdew90, namsgruber\}$ and $\{nebula, galaxies_distances_and_redshifts, uk, 287, ukst, h_0, flair\}$ from the DBLP and NASA datasets, respectively. For each of these two keyword sets, we constructed three queries containing five, six and seven keywords. Further, for each query, we truncated the inverted lists of each keywords at 20%, 40%, 60% and 80% of their length. The measured computation times are presented in Figures 5.19a and 5.19b. As shown in the figures, the curves are almost linear and the computation time scales smoothly increasing in the worst case by 40% when the input size quintuples.



(a) NASA dataset



(b) DBLP dataset

Figure 5.19 Computation time vs. input size for *ClusterStack* using queries with 5,6 and 7 keywords.

5.5 Conclusion

We have addressed the problem of clustering the results of keyword search on tree data in order to support the user in her quest for meaningful answers. We introduced a multi-level clustering methodology which groups together results with similar structural and semantic features. In order to define our cluster hierarchy, we introduced different relations on patterns of results based on homomorphisms between pattern paths. In our system, the users navigate through the hierarchy by drilling down from clusters to subclusters while novel techniques for ranking the clusters at different levels facilitate and shorten their search. We designed an efficient algorithm for generating and clustering result patterns, and for building the cluster hierarchy. Our experimental studies showed that the proposed algorithm is

fast and scalable. They also showed that the proposed clustering methodology allows the users to effectively retrieve their intended results and outperforms a recent state-of-the-art competitor approach.

CHAPTER 6

DIVERSIFICATION

In this section, we introduce our approach for diversification of result patterns. We initially provide a formal definition of the diversification problem on the results of keyword search over tree data. Then, we present the different components of our diversification scheme.

6.1 Formal Problem Definition

Diversification aims to provide the users with a result set that contains relevant and diverse results. We define the problem of diversification of keyword search result patterns as an optimization problem. Previous works [1, 12, 25] have also defined diversification as an optimization problem but in a different way and on different data models (web data, flat documents, structured databases etc.). Given a data tree T and a query Q , let S be the set of result patterns of Q on T and k be a positive integer. The goal of diversification is to choose a subset R of S of size k whose elements are both relevant and diverse. Therefore, R is defined as:

$$R \in \operatorname{argmax}_{R' \subseteq S, |R'|=k} (\lambda \operatorname{relevance}(R', Q) + (1 - \lambda) \operatorname{diversity}(R'))$$

where λ is a parameter in the $[0,1]$ range which tunes the importance of relevance and diversity. This tuning factor allows us to give more importance to the relevance or diversity. If $\lambda = 1$, the result set would be formed solely based on the relevance of the patterns, whereas if $\lambda = 0$, the result set will contain the most diverse set of patterns without considering the relevance of the patterns. We call the function $\lambda \operatorname{relevance}(R', Q) + (1 - \lambda) \operatorname{diversity}(R')$, *diversification function*.

Here, we assume that the relevance of a result is independent of the relevance of the

other results in R' . Therefore, the relevance of R' with respect to Q is defined as:

$$relevance(R', Q) = \frac{1}{k} \sum_{P \in R'} rel(P, Q)$$

where $rel(P, Q)$ stands for the relevance of result pattern P with respect to Q . We define later $rel(P, Q)$ as a value in the interval $[0, 1]$. Since the sum is divided by the number k of results in R' , $relevance(R', Q)$ ranges between 0 and 1.

A set of patterns is diverse when the pairwise dissimilarity of its patterns is maximized. We define the pairwise dissimilarity of two patterns as the negative of their similarity. Therefore, the diversity of R' is given by the following formula:

$$diversity(R') = -\frac{1}{k(k-1)} \sum_{P, P' \in R', P' \neq P} sim(P, P')$$

where $sim(P, P')$ denotes the similarity between the patterns P and P' , and is defined later as a value in the range $[0, 1]$. The sum is divided by the number of pairs of patterns in R' to guarantee that $diversity(R') \in [0, 1]$.

One can see from the formula above that we need to quantify the relevance of a given pattern and the similarity of two given patterns. In the next sections, we describe how the relevance of a pattern and the similarity between two patterns can be measured. Our main goal in this work is the formulation of the diversification problem in the domain of keyword search over tree data and a measure of the diversification of a set of patterns in terms of the dissimilarity between pairs of patterns. However, we also introduce a simple way for quantifying the relevance of a pattern. Other relevance measures can also be adopted to design a diversification scheme for keyword search result patterns.

Our definition of the diversification problem does not involve ranking of the patterns. We only select a set of patterns. However, the patterns returned to the user are ranked.

6.2 Relevance of Patterns

In this section, we introduce our relevance scoring scheme. We adapt a statistical approach that incorporates both semantic and structural information of the patterns. Our approach utilizes the TF-IDF metric customized to tree data. TF-IDF [70] has been widely used in information retrieval for assigning weights to the terms. It combines the term frequency of a term in a document and its inverse document frequency. Inverse document frequency is computed as the reciprocal of the number of documents that contain a term. Intuitively, a term's occurrence frequency in a single document implies its importance for that document but if the same term occurs in many different documents, this is used to lower the term's importance. TF-IDF has also been adapted by some keyword search approaches applied on tree structured data [6, 46]. Our approach is an adaptation of the one presented in [46].

We define the weight of a keyword instance node n_k annotated by keyword k in a pattern as follows: $weight(n_k) = \log_2(ief(k))$ where ief is the *inverse element frequency* of keyword k in the data tree. The value of ief of keyword k is calculated as the ratio of the number of elements in the data tree over the number of elements in the data tree that contain k in the subtrees rooted at them. This weighting scheme gives nodes that contain the same keyword the same weight value. Consider, for instance, the data tree in Figure 6.1. The ief of keyword *physics* is $\log_2(24/18) = 0.42$ and ief of the keyword *quantum* is $\log_2(24/13) = 0.88$. These numbers also indicate the importance of the nodes that contain them in a result pattern and therefore, we use them to represent the nodes' contribution to the total relevance of the pattern. However, using merely the node weights is not enough for designing an accurate relevance score.

In order to take into account the structure of the result patterns, we reduce the weight of each keyword instance node accordingly by its distance from the LCA node, denoted as $dist(n_k, LCA(P))$. This reduced score is then aggregated to compute the relevance scores of patterns. The reason behind this reduction is that intuitively, the nodes that appear farther from the LCA node contribute to the relevance of the patterns to a lesser extent. Given a

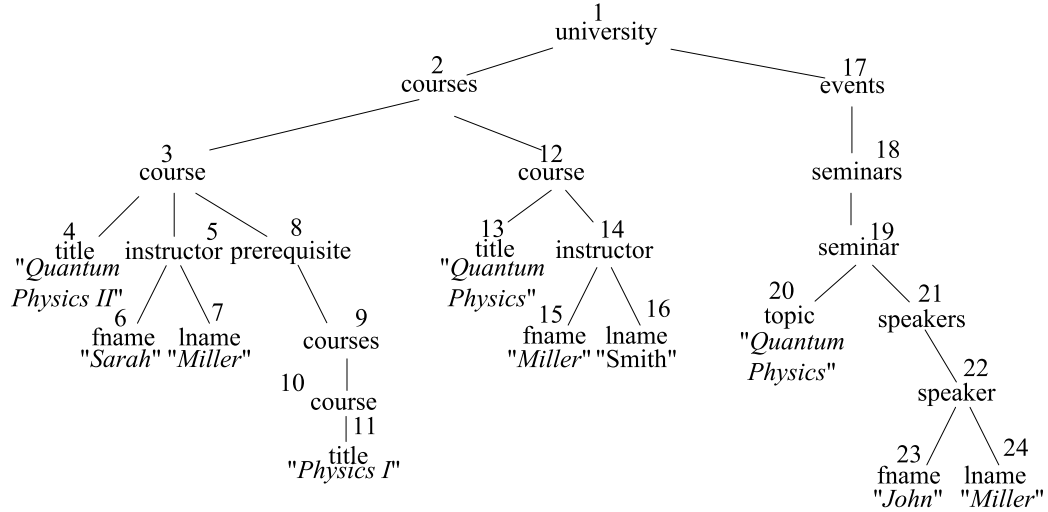


Figure 6.1 A data tree which represents a university database consisting of courses and seminars

result pattern P for a query Q that consists of n keywords,

$$rel(P, Q) = \sum_{k=1}^n \frac{weight(n_k)}{dist(n_k, LCA(P))}$$

Let d be the distance of node n_k from the LCA node of pattern P and avg_depth be the average depth of the data tree. $dist(n_k, LCA(P))$ is defined as follows:

$$dist(n_k, LCA(P)) = \begin{cases} 1 & \text{if } d = 0 \\ d & \text{if } d \leq avg_depth \\ avg_depth + (d - avg_depth)\mu & \text{otherwise} \end{cases}$$

We normalize the distance of the nodes to the LCA node by the average depth of the data tree in order to reduce the penalty of the nodes in trees which have a high average depth. We let μ be 0.5.

Consider the patterns shown in Figure 6.2 for the keyword query $Q = \{Quantum, Physics, Miller\}$ on the tree of Figure 6.1. Patterns P , P' and P'' represent courses that contain all the keywords under their subtrees. Pattern P contains the keyword *physics*

under the title of a prerequisite course, whereas P' and P'' contain all the information under a single course (a course titled “Quantum Physics” and taught by an instructor whose first name or last name is “Miller”). Pattern P''' represents a seminar that contains all the query keywords. According to the relevance formula given above, $rel(P, Q) = 0.70$, $rel(P', Q) = 0.82$, $rel(P'', Q) = 0.82$ and $rel(P''', Q) = 0.78$. One can see that patterns that have keywords closer to their LCA node usually represent a more meaningful relationship between the keywords (as a counter example, pattern P represents a loose relationship between the keywords). The relevance scores are normalized into the $[0,1]$ range before being combined with the similarity scores in the diversification function.

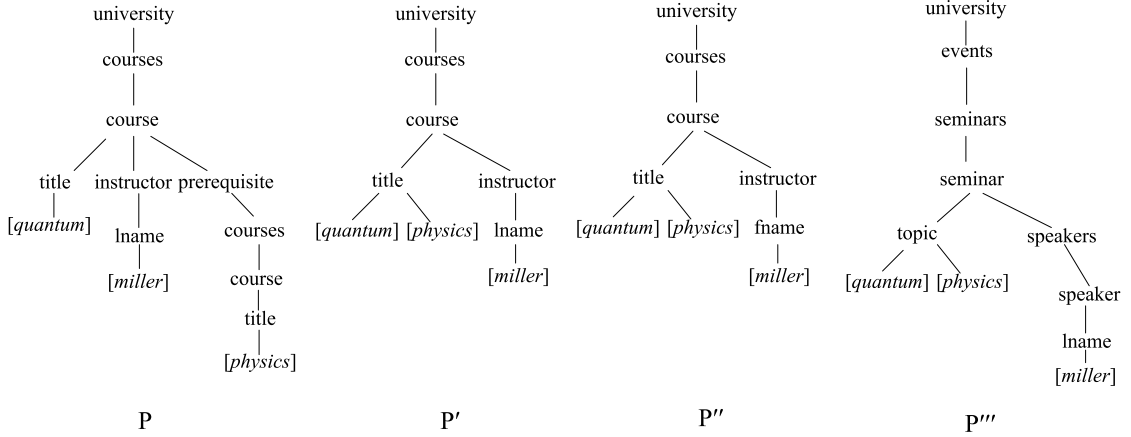


Figure 6.2 Four patterns for $Q = \{Quantum, Physics, Miller\}$ on the tree of Figure 6.1

6.3 Similarity of Patterns

We define the similarity of patterns based on the similarities of their corresponding paths. The similarity of the paths, in turn, is defined based on the similarity of the edges they contain. This similarity scoring takes advantage of both structural and semantic information of the patterns.

Given an edge $e = (n_1, n_2)$, let $c_k(e)$ be the set of annotating keywords which occur in the subtree rooted at n_2 . Let $e = (n_1, n_2)$ and $e' = (n'_1, n'_2)$ be two edges in two patterns P and P' , respectively, such that $label(n_1) = label(n'_1)$ and $label(n_2) = label(n'_2)$. The simi-

larity of e and e' , $\text{sim}(e, e') = |c_k(e) \cap c_k(e')|$. Intuitively, the number of shared annotating keywords under two edges indicates the shared context under the subtrees below these edges. For instance, consider the patterns depicted in Figure 6.3. These patterns represent keyword query results for the query *sarah, miller* on a university database. The corresponding edges in the paths of the patterns are depicted with the same subscript followed by a dot and an index. The similarity of edges e_1 and $e_{1.1}$ and the similarity of e_1 and $e_{1.2}$ are both 2, since these edges have the same set of annotating keywords. All three patterns contain the edge (university/courses) in all their root-to-annotated-node paths so this edge indicates a strong shared information about the patterns. For courses/course edges, $\text{sim}(e_2, e_{2.1}) = 2$ whereas $\text{sim}(e_2, e_{2.2}) = 1$. This shows that e_2 and $e_{2.1}$ are more similar because they contain both of the keywords under their subtrees but $e_{2.2}$ and $e_{2.3}$ contain only one of the keywords under their subtrees.

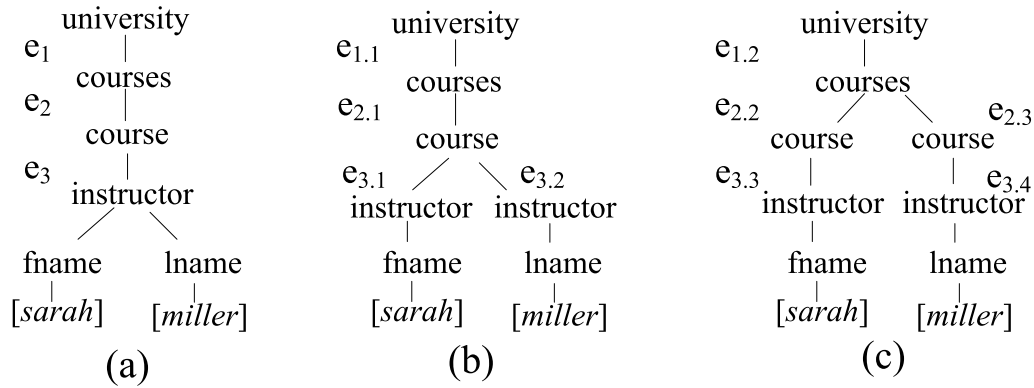


Figure 6.3 Three patterns which match the keyword query $\{sarah, miller\}$

Given two patterns P and P' for the same keyword query Q and two root-to-annotated-node paths p and p' , respectively, with the same annotated leaf node, a similarity mapping between p and p' is a one-to-one function m from edges of p to edges of p' satisfying the following properties: (a) the mapped edges have the same labels (that is, if $e = (n_1, n_2)$, $e' = (n'_1, n'_2)$ and $m(e) = e'$, then $\text{label}(n_1) = \text{label}(n_2)$ and $\text{label}(n'_1) = \text{label}(n'_2)$), and (b) the domain of m , $\text{dom}(m)$, is maximal in the sense that m cannot be extended to map any additional edges from p . Clearly, a similarity mapping might not map all the edges of p to

edges of p' . Further, there can be many similarity mappings between p and p' since there can be more than one edges in p (or in p') with the same labels. Consider, for instance the two paths in Figure 6.4. There are two possible mappings, say m and m' , from the edges of p to p' which are depicted with different dashed arrows. The difference between m and m' is the mapping of courses/course edge, e in p to either e' or e'' in p' . According to our edge scoring scheme, mapping e to e' gives a better score because these edges share more annotated keywords under their subtrees.

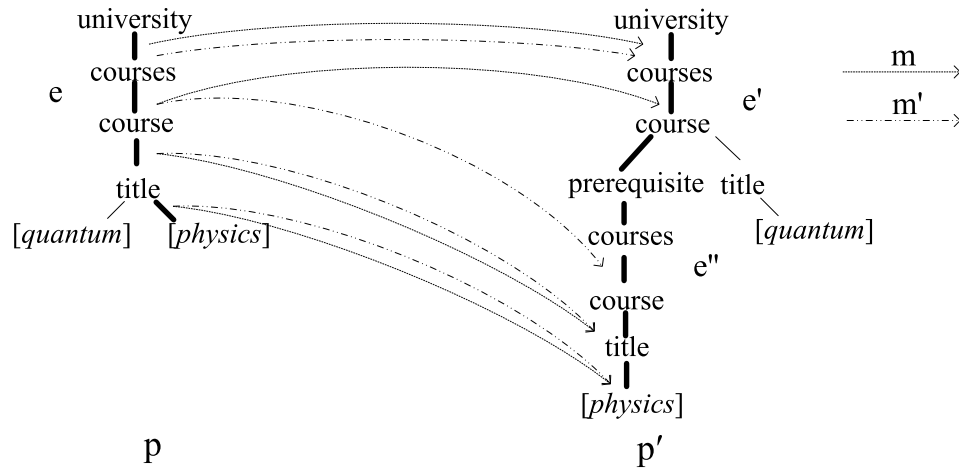


Figure 6.4 Two paths p and p' shown with bold edges and possible mappings between their edges depicted with different dashed arrows

Let now $M(p, p')$ be the set of all similarity mappings between two paths p and p' . The similarity of p and p' is

$$sim(p, p') = \max\{s \mid s = \sum_{e \in dom(m)} sim(e, e') \text{ where } e' = m(e) \text{ and } m \in M(p, p')\}$$

On the paths of Figure 6.4, $sim(p, p') = 6$ which is given by the mapping m .

We aggregate the similarity of corresponding root-to-annotated-node paths in patterns to compute the similarity of the patterns. Let P and P' be two patterns for the same keyword query Q . We first define the path mapping similarity between P and P' as $pathmap(P, P') = \sum_{p \in P, p' \in P'} sim(p, p')$. The similarity of patterns P and P' is defined as

follows:

$$\text{sim}(P, P') = \frac{\text{pathmap}(P, P')}{\frac{1}{2}(\text{pathmap}(P, P) + \text{pathmap}(P', P'))}$$

The denominator of the formula guarantees that the similarity value lies in the $[0, 1]$ range. For example, consider the four patterns in Figure 6.2. Similarity of P and P' , $\text{sim}(P, P')$, is 0.91 and $\text{sim}(P, P''')$ is 0.03. One can see that patterns that represent similar concepts (same node types) will get higher similarity values which is very useful in choosing a diverse set of results. Even though the patterns P' and P'' have the highest relevance scores, the diversification scheme will choose P''' over P'' since it is less similar to P' than P'' and therefore, P''' will contribute to a higher value for the diversification function.

6.4 Algorithm

As described in Section 6.1, selecting a diverse set of results from the set of all results is an optimization problem. Given a query Q on a data tree T and the set S of all patterns that match Q on T , the brute force solution to this optimization problem would be to generate all the k -sized subsets of S and select the ones that maximize the objective function. Clearly, this solution is not computationally feasible as the number of possible subsets is $\binom{|S|}{k}$. Different variations of the diversification problem have been proven to be NP-complete [1, 15, 28, 29]. Therefore, in this section, we introduce a heuristic algorithm, Algorithm 7, for computing the diverse set of keyword search results. This algorithm incrementally builds the result set by making greedy choices for the inclusion of every new result in the diverse result set. The resulting diverse result set is an approximation of the optimal set.

The algorithm takes as input the set of patterns of a query Q on a data tree and a parameter k which is the desired size for the diverse result set. The patterns of the query result are generated efficiently using the PatternStack algorithm [2]. We sort the input patterns with respect to their relevance to the query in descending order. The first pattern, the most relevant pattern, is included in the result set by default. Indeed, if $k = 1$ this would

Algorithm 7: *Diversify* algorithm

```

1 Diversify( $Q = \{k_1, \dots, k_n\}$ : keyword query,
2            $S$ : set of patterns,
3            $k$ : size of diverse set)
4    $R = \{\}$  /* empty list of patterns */
5    $L = \text{sortByRelevance}(S)$ 
6    $R.\text{add}(L[0])$ 
7    $i = 1$ 
8   while  $i < k$  do
9        $j = i$ 
10       $\text{maxScore} = 0$ 
11       $\text{maxIndex} = -1$ 
12      while  $j < |S|$  do
13          if  $\lambda_{\text{rel}}(L[j], Q) < \text{maxScore}$  then
14              break
15           $\text{sumSimilarity} = 0$ 
16           $l = 0$ 
17          while  $l < |R|$  do
18               $\text{sumSimilarity} = \text{sumSimilarity} + \text{sim}(L[j], R[l])$ 
19               $l = l + 1$ 
20           $\text{curScore} = \lambda_{\text{rel}}(L[j], Q) - (1 - \lambda)(\text{sumSimilarity}/|R|)$ 
21          if  $\text{curScore} > \text{maxScore}$  then
22               $\text{maxScore} = \text{curScore}$ 
23               $\text{maxIndex} = j$ 
24           $j = j + 1$ 
25       $R.\text{add}(L[\text{maxIndex}])$ 
26       $\text{tmp} = L.\text{remove}(\text{maxIndex})$ 
27       $L.\text{insert}(i, \text{tmp})$ 
28       $i = i + 1$ 
29  return  $R$ 

```

be a trivial choice (line 5). Then, we iterate over the remaining list of patterns in L to find the next pattern to be included in the result list R (lines 8-28). The iteration terminates when all patterns in L have been examined or the score of a pattern cannot be greater than the current maximum score (line 13). The algorithm stops when the size of R is equal to k . The worst case complexity of the algorithm is $O(|S|k^2)$ where k is the desired size of R .

6.5 Experimental Evaluation

In this section, we elaborate on the proposed experimental evaluation to assess the effectiveness of our diversification scheme and the efficiency of the heuristic algorithm.

6.5.1 Evaluation Metrics

Different metrics have been used in measuring the success of diversification methods. In [19], α -NDCG is introduced as an adaptation of NDCG (normalized discounted cumulative gain). The α -NDCG is adapted in a way that the gain from the retrieval of a document is reduced by the extent that the information nuggets of the document is included in the previously retrieved documents. The definition of information nuggets in keyword search over tree data results is not straightforward. In [25], authors define primary keys in relational databases to be the information nuggets. The construct that corresponds to primary keys in the context of tree data might be the LCA nodes of the results. However, since the inclusion of LCA nodes is binary, it is not possible to accurately decide for the redundancy of a result.

S-recall [17] is another diversification related evaluation metric. However, S-recall requires either a taxonomy or pre-defined topics on the dataset so that the metric can quantify the coverage of unique subtopics by the returned result set. This kind of taxonomy is not available for tree data.

Another class of metrics are the redundancy metrics introduced in [94]. Zhang et al. [94] adapt precision and recall metrics to measure the elimination success of redundant results. Redundancy-Precision is defined as the ratio of eliminated redundant results to the total eliminated results. Redundancy-Recall is defined as the ratio of the eliminated redundant results to the total number of redundant results. This metric requires binary redundancy decisions by human experts for each result. The expert decisions are done based on a ranked list and the expert decides redundancy of a result considering the previously retrieved results.

6.5.2 Experiments

As described in the previous section, defining an evaluation metric that measures the diversification quality of the result set is not a straightforward task. Therefore, one can evaluate the accuracy of our relevance and similarity metrics separately. Top-k retrieval experiments can be conducted with our relevance measure, and precision and recall metrics can be computed over the top-k results. In order to measure the effectiveness of our similarity measure (and our diversification scheme, in general), redundancy-precision and redundancy-recall measures can be utilized. Finally, experiments to assess the efficiency and scalability of our proposed algorithm can be conducted.

6.6 Conclusion

We have formulated the problem of diversification of results of keyword search on tree data. Diversification aims to balance the relevance and diversity of the results contained in the result set. A diverse result set helps the users to see different aspects of the results. To this end, we formalized the diversification problem in the domain of tree data as an optimization problem. We introduced a simple relevance measure. This measure can be computed easily through statistical information which is extracted from the data offline. As our main contribution, we devised a similarity measure between the patterns of a keyword query over tree data. This measure takes into account both the structural and semantic information of the patterns. Finally, we presented a heuristic algorithm that incrementally builds the diverse result set in a greedy fashion. We also suggested experiments to be conducted to measure the accuracy of the proposed diversification scheme and the efficiency of the heuristic algorithm.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Keyword search is a convenient way of querying data on the web. Recently, it also gained popularity in the area of tree data. However, its flexibility comes with multiple drawbacks. In this dissertation, we have proposed solutions to two different problems in the domain of keyword search over tree structured data. First, we tackled the problem of selecting relevant results of a keyword query. To this end, we presented a new keyword search approach, *XReason*, which provides filtering and ranking semantics to keyword queries on tree data. *XReason* semantics are based on reasoning with patterns of the results instead of simply assigning scores to the results or patterns. We introduced relations on patterns using different types of homomorphisms and utilized these relations to define our semantics. Instead of processing individual results on the data tree, our approach benefits from a global view of these results. We also have conducted an in-depth comparison of our approach with previous approaches. We presented a stack-based algorithm to efficiently compute the result patterns. Our exhaustive experimental results showed that our algorithm is efficient and scales smoothly. They also showed that *XReason* outperforms previous approaches both as a filtering and ranking semantics.

As a second problem, we addressed the problem of result disambiguation. Keyword queries, in all domains of data, are inherently imprecise. This imprecision become more pronounced on the domain of tree data since the users cannot also specify structural constraint. In order to tackle the result disambiguation problem, we have proposed the application of clustering of the keyword search results in order to support the user in her quest for meaningful answers. Clustering has also been proposed on the Web as a different way of organizing and browsing the results. We introduced a multi-level clustering methodology for keyword search over tree data which groups together results with similar structural and semantic features. In order to define our cluster hierarchy, we introduced

different relations on patterns of results based on homomorphisms between pattern paths. In our system, the users navigate through the hierarchy by drilling down from clusters to subclusters while novel techniques for ranking the clusters at different levels facilitate and shorten their search. We designed an efficient algorithm for generating and clustering result patterns, and for building the cluster hierarchy. Our experimental studies showed that the proposed algorithm is fast and scalable. They also showed that the proposed clustering methodology allows the users to effectively retrieve their intended results and outperforms a recent state-of-the-art competitor approach.

In order to address the result disambiguation problem from a different aspect, we also investigated the task of diversification of keyword search results. Diversification of search results aims to provide the users with a diverse subset of results without substantially compromising relevancy. It has been studied in different domains to tackle the problems of result disambiguation, over specialization and exploratory search. We proposed applying diversification to the results of keyword search on tree data. We formalized the diversification problem in the domain of tree data. In order to select a set of patterns whose elements are both relevant and diverse, we introduced a relevance measure for patterns and a similarity measure between two patterns of a keyword query. We also presented a greedy heuristic which computes an approximation of the optimal diverse result set. Finally, we elaborated on different possible experiments that can be conducted to assess the efficiency of our heuristic algorithm and the effectiveness of our diversification scheme. Future work includes the experimental assessment of the accuracy of our diversification scheme and the efficiency of our heuristic algorithm.

REFERENCES

- [1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Jeong. Diversifying search results. In *International Conference on Web Search and Data Mining*, pages 5–14, 2009.
- [2] C. Aksoy, A. Dimitriou, D. Theodoratos, and X. Wu. XReason: A semantic approach that reasons with patterns to answer XML keyword queries. In *International Conference on Database Systems for Advanced Applications*, pages 299–314, 2013.
- [3] C. Aksoy, A. Dass, D. Theodoratos, and X. Wu. Clustering query results to support keyword search on tree data. In *International Conference on Web-Age Information Management*, pages 213–224, 2014.
- [4] C. Aksoy, A. Dimitriou, and D. Theodoratos. Reasoning with patterns to effectively answer XML keyword queries. *The VLDB Journal*, 24(3):441–465, 2015.
- [5] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, Boston, MA, USA, 1999.
- [6] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective XML keyword search with relevance oriented ranking. In *International Conference on Data Engineering*, pages 517–528, 2009.
- [7] Z. Bao, J. Lu, T. W. Ling, and B. Chen. Towards an effective XML keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1077–1092, 2010.
- [8] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *International Conference on Data Engineering*, pages 431–440, 2002.
- [9] C. Botev and J. Shanmugasundaram. Context-sensitive keyword search and ranking for XML. In *International Workshop on the Web and Databases*, pages 115–120, 2005.
- [10] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.
- [11] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal XML pattern matching. In *ACM SIGMOD/PODS Conference*, pages 310–321, 2002.
- [12] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336, 1998.
- [13] C. Carpineto, S. Osiński, G. Romano, and D. Weiss. A survey of Web clustering engines. *ACM Computing Surveys*, 41(3):17:1–17:38, 2009.
- [14] Carrot2. Carrot2 Clustering Engine. <http://search.carrot2.org/stable/search>, Retrieved on December 7 2015.

- [15] B. Carterette. An analysis of np-completeness in novelty and diversity ranking. *Information Retrieval*, 14(1):89–106, 2011.
- [16] B. Chen, J. Lu, and T. W. Ling. Exploiting id references for effective keyword search in xml documents. In *International Conference on Database Systems for Advanced Applications*, volume 4947, pages 529–537. Springer Berlin Heidelberg, 2008.
- [17] H. Chen and D. R. Karger. Less is more: Probabilistic models for retrieving fewer relevant documents. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 429–436, 2006.
- [18] L. J. Chen and Y. Papakonstantinou. Supporting top-K keyword search in XML databases. In *International Conference on Data Engineering*, pages 689–700, 2010.
- [19] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 659–666, 2008.
- [20] P. Clough and M. Sanderson. Evaluating the performance of information retrieval systems using test collections. *Information Research*, 18(2), 2013.
- [21] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A semantic search engine for XML. In *International Conference on Very Large Data Bases*, pages 45–56, 2003.
- [22] W. W. W. Consortium. XML Path Language (XPath). <http://www.w3.org/TR/xpath/>, Retrieved on December 7 2015.
- [23] W. W. W. Consortium. XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery/>, Retrieved on December 7 2015.
- [24] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31(3):187 – 228, 2006.
- [25] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. Divq: Diversification for keyword search over structured databases. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 331–338, 2010.
- [26] A. Dimitriou and D. Theodoratos. Efficient keyword search on large tree structured datasets. In *International Workshop on Keyword Search and Data Exploration on Structured Data*, pages 63–74, 2012.
- [27] A. Dimitriou, D. Theodoratos, and T. Sellis. Top-k-size keyword search on tree structured data. *Information Systems*, 47(0):178 – 193, 2015.
- [28] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Records*, 39(1): 41–47, Sept. 2010.
- [29] E. Erkut, Y. Ulkusal, and O. Yenicerioglu. A comparison of p-dispersion heuristics. *Computers & Operations Research*, 21(10):1103 – 1113, 1994.

- [30] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *International World Wide Web Conference*, pages 381–390, 2009.
- [31] G. Gou and R. Chirkova. Efficient algorithms for evaluating xpath over streams. In *ACM SIGMOD/PODS Conference*, pages 269–280, 2007.
- [32] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *ACM SIGMOD/PODS Conference*, pages 16–27, 2003.
- [33] M. Hasan, A. Mueen, V. Tsotras, and E. Keogh. Diversifying query results on semi-structured data. In *International Conference on Information and Knowledge Management*, pages 2099–2103, 2012.
- [34] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *International Conference on Very Large Data Bases*, pages 670–681, 2002.
- [35] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(4):525–539, 2006.
- [36] Y. Huang, Z. Liu, and Y. Chen. Query biased snippet generation in XML search. In *ACM SIGMOD/PODS Conference*, pages 315–326, 2008.
- [37] A. Hulgeri, G. Bhalotia, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword search in databases. *IEEE Data Engineering Bulletin*, 24(3):22–31, 2001.
- [38] V. Josifovski, M. Fontoura, and A. Barta. Querying XML streams. *The VLDB Journal*, 14(2):197–210, 2005.
- [39] L. Kong, R. Gilleron, and A. L. Mostrare. Retrieving meaningful relaxed tightest fragments for XML keyword search. In *International Conference on Extending Database Technology*, pages 815–826, 2009.
- [40] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *International World Wide Web Conference*, pages 658–665, 2004.
- [41] Y. Kural, S. Robertson, and S. Jones. Deciphering cluster representations. *Information Processing & Management*, 37(4):593 – 601, 2001.
- [42] K.-H. Lee, K.-Y. Whang, W.-S. Han, and M.-S. Kim. Structural consistency: enabling XML keyword search to eliminate spurious results consistently. *The VLDB Journal*, 19(4):503–529, 2010.
- [43] G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *International Conference on Information and Knowledge Management*, pages 31–40, 2007.

- [44] G. Li, C. Li, J. Feng, and L. Zhou. SAIL: Structure-aware indexing for effective and progressive top-k keyword search over XML documents. *Information Sciences*, 179(21):3745–3762, 2009.
- [45] J. Li and J. Wang. XQSuggest: An interactive XML keyword search system. In *International Conference on Database and Expert Systems Applications*, pages 340–347. 2009.
- [46] J. Li, C. Liu, R. Zhou, and W. Wang. Suggestion of promising result types for XML keyword search. In *International Conference on Extending Database Technology*, pages 561–572, 2010.
- [47] J. Li, C. Liu, and J. Yu. Context-based diversification for keyword queries over xml data. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):660–672, March 2015.
- [48] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *International Conference on Very Large Data Bases*, pages 72–83, 2004.
- [49] W. Lian, D.-L. Cheung, N. Mamoulis, and S.-M. Yiu. An efficient and scalable algorithm for clustering XML documents by structure. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):82–96, 2004.
- [50] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *ACM SIGMOD/PODS Conference*, pages 563–574, 2006.
- [51] J. Liu, J. Wang, W. Hsu, and K. Herbert. XML clustering by principal component analysis. In *International Conference on Tools with Artificial Intelligence*, pages 658–662, 2004.
- [52] J. Liu, J. Wang, J. Hu, and B. Tian. A method for aligning RNA secondary structures and its application to rna motif detection. *BMC Bioinformatics*, 6(1):89, 2005.
- [53] X. Liu, C. Wan, and L. Chen. Returning clustered results for keyword search on XML documents. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1811–1825, 2011.
- [54] Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *ACM SIGMOD/PODS Conference*, pages 329–340, 2007.
- [55] Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. *The Proceedings of the VLDB Endowment*, 1(1):921–932, 2008.
- [56] Z. Liu and Y. Chen. Answering keyword queries on XML using materialized views. In *International Conference on Data Engineering*, pages 1501–1503, 2008.
- [57] Z. Liu and Y. Chen. Return specification inference and result clustering for keyword search on XML. *ACM Transactions on Database Systems*, 35(2):10:1–10:47, 2010.

- [58] Z. Liu and Y. Chen. Processing keyword search on XML: a survey. *World Wide Web*, 14(5-6):671–707, 2011.
- [59] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *The Proceedings of the VLDB Endowment*, 2(1):313–324, Aug. 2009.
- [60] Z. Liu, S. Natarajan, and Y. Chen. Query expansion based on clustered results. *The Proceedings of the VLDB Endowment*, 4(6):350–361, Mar. 2011.
- [61] Y. Lu, W. Wang, J. Li, and C. Liu. XClean: Providing valid spelling suggestions for XML keyword queries. In *International Conference on Data Engineering*, pages 661–672, 2011.
- [62] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *ACM SIGMOD/PODS Conference*, pages 115–126, 2007.
- [63] J. Madhavan, S. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: You can only afford to pay as you go. *Conference on Innovative Data Systems Research*, 2007.
- [64] R. Nayak. Fast and effective clustering of XML data using structural information. *Knowledge and Information Systems*, 14(2):197–215, 2008.
- [65] K. Nguyen and J. Cao. Top-k answers for XML keyword queries. *World Wide Web*, 15(5-6):485–515, 2012.
- [66] P. Ogden, D. Thomas, and P. Pietzuch. Scalable XML query processing using parallel pushdown transducers. *The Proceedings of the VLDB Endowment*, 6(14):1738–1749, Sept. 2013.
- [67] K. Q. Pu and X. Yu. Keyword query cleaning. *The Proceedings of the VLDB Endowment*, 1(1):909–920, 2008.
- [68] F. Radlinski and S. Dumais. Improving personalized web search using result diversification. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 691–692, 2006.
- [69] V. Raghavan, P. Bollmann, and G. S. Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7(3):205–229, 1989.
- [70] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.
- [71] A. Schmidt, M. Kersten, and M. Windhouwer. Querying XML documents made easy: nearest concept queries. In *International Conference on Data Engineering*, pages 321–329, 2001.

- [72] F. Shao, L. Guo, C. Botev, A. Bhaskar, M. Chettiar, F. Yang, and J. Shanmugasundaram. Efficient keyword search over virtual XML views. *The VLDB Journal*, 18(2):543–570, 2009.
- [73] B. A. Shapiro and K. Zhang. Comparing multiple rna secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6(4):309–318, 1990.
- [74] S. Soudatos, X. Wu, D. Theodoratos, T. Dalamagas, and T. Sellis. Evaluation of partial path queries on XML data. In *International Conference on Information and Knowledge Management*, pages 21–30, 2007.
- [75] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23, 2000.
- [76] J. Stefanowski and D. Weiss. Carrot and language properties in web search results clustering. In *International Atlantic Web Intelligence Conference*, pages 240–249, 2003.
- [77] C. Sun, C. Y. Chan, and A. K. Goenka. Multiway SLCA-based keyword search in XML data. In *International World Wide Web Conference*, pages 1043–1052, 2007.
- [78] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *ACM SIGMOD/PODS Conference*, pages 204–215, 2002.
- [79] A. Termehchy and M. Winslett. Using structural information in XML keyword search effectively. *ACM Transactions on Database Systems*, 36(1):4, 2011.
- [80] D. Theodoratos and X. Wu. An original semantics to keyword queries for XML using structural patterns. In *International Conference on Database Systems for Advanced Applications*, pages 727–739, 2007.
- [81] D. Theodoratos and X. Wu. Eager evaluation of partial tree-pattern queries on XML streams. In *International Conference on Database Systems for Advanced Applications*, pages 241–246, 2009.
- [82] T. Tian, J. Geller, and S. Chun. Improving web search results for homonyms by suggesting completions from an ontology. In *Current Trends in Web Engineering*, pages 175–186. 2010.
- [83] T. Tian, J. Geller, and S. Chun. Enhancing the interface for ontology-supported homonym search. In *Conference on Advanced Information Systems Engineering Workshops*, pages 544–553. 2011.
- [84] A. Turel and F. Can. A new approach to search result clustering and labeling. In *Asia Information Retrieval Symposium*, pages 283–292, 2011.

- [85] H. Wang, W. Wang, X. Lin, and J. Li. Labeling scheme and structural joins for graph-structured XML data. In *Asia Pacific Web Conference*, pages 277–289, 2005.
- [86] X. Wu. *Semantics and efficient evaluation of partial tree-pattern queries on XML*. PhD thesis, New Jersey Institute of Technology, 2010.
- [87] X. Wu, S. Soudatos, D. Theodoratos, T. Dalamagas, and T. Sellis. Efficient evaluation of generalized path pattern queries on XML data. In *International World Wide Web Conference*, pages 835–844, 2008.
- [88] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *ACM SIGMOD/PODS Conference*, pages 537–538, 2005.
- [89] Y. Xu and Y. Papakonstantinou. Efficient LCA based keyword search in XML data. In *International Conference on Extending Database Technology*, pages 535–546, 2008.
- [90] C. Yu, L. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *International Conference on Data Engineering*, pages 1299–1302, 2009.
- [91] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. in *IEEE Transaction on Knowledge and Data Engineering*, 17:1021–1035, 2005.
- [92] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 46–54, 1998.
- [93] M. Zhang and N. Hurley. Avoiding monotony: Improving the diversity of recommendation lists. In *ACM Recommender Systems Conference*, pages 123–130, 2008.
- [94] Y. Zhang, J. Callan, and T. Minka. Novelty and redundancy detection in adaptive filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 81–88, 2002.
- [95] J. Zhou, Z. Bao, W. Wang, T. W. Ling, Z. Chen, X. Lin, and J. Guo. Fast SLCA and ELCA computation for XML keyword queries based on set intersection. In *International Conference on Data Engineering*, pages 905–916, 2012.
- [96] J. Zhou, X. Zhao, W. Wang, Z. Chen, and J. X. Yu. Top-down keyword query processing on XML data. In *International Conference on Information and Knowledge Management*, pages 2225–2230, 2013.
- [97] J. Zhou, Z. Bao, W. Wang, J. Zhao, and X. Meng. Efficient query processing for XML keyword queries based on the idlist index. *The VLDB Journal*, 23(1):25–50, 2014.
- [98] R. Zhou, C. Liu, and J. Li. Fast ELCA computation for keyword queries on XML data. In *International Conference on Extending Database Technology*, pages 549–560, 2010.

- [99] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *International World Wide Web Conference*, pages 22–32, 2005.