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

PROCESSING TECHNIQUES FOR PARTIAL TREE-PATTERN QUERIES ON XML DATA

by
Pawel Placek

In recent years, eXtensible Markup Language (XML) has become a de facto standard for exporting and exchanging data on the Web. XML structures data as trees. Querying capabilities are provided through patterns matched against the XML trees. Research on the processing of XML queries has focused mainly on tree-pattern queries. Tree-pattern queries are not appropriate for querying XML data sources whose structure is not fully known to the user, or for querying multiple data sources which structure information differently. Recently, a class of queries, called Partial Tree-Pattern Queries (PTPQs) was identified. A central feature of PTPQs is that the structure can be specified fully, partially, or not at all in a query. For this reason, PTPQs can be used for flexibly querying XML data sources.

This thesis deals with processing techniques for PTPQs. In particular, it addresses the satisfiability, containment and minimization problems for PTPQs. In order to cope with structural expression derivation issues and to compare PTPQs, a set of inference rules is suggested and a canonical form for PTPQs that comprises all derived structural expressions is defined. This canonical form is used for determining necessary and sufficient conditions for PTPQ satisfiability.

The containment problem is studied both in the absence and in the presence of structural summaries of data called dimension graphs. It is shown that this problem cannot be characterized by homomorphisms between PTPQs, even when PTPQs are put in canonical form. In both cases of the problem, necessary and sufficient conditions for PTPQ containment are provided in terms of homomorphisms between PTPQs and (a possibly exponential number of) tree-pattern queries. This result is

used to identify a subclass of PTPQs that strictly contains tree-pattern queries for which the containment problem can be fully characterized through the existence of homomorphisms. To cope with the high complexity of PTPQ containment, heuristic approaches for this problem are designed that trade accuracy for speed. The heuristic approaches equivalently add structural expressions to PTPQs in order to increase the possibility for a homomorphism between two contained PTPQs to exist. An implementation and extensive experimental evaluation of these heuristics shows that they are useful in practice, and that they can be efficiently implemented in a query optimizer.

The goal of PTPQ minimization is to produce an equivalent PTPQ which is syntactically smaller in size. This problem is studied in the absence of structural summaries. It is shown that PTPQs cannot be minimized by removing redundant parts as is the case with certain classes of tree-pattern queries. It is also shown that, in general, a PTPQ does not have a unique minimal equivalent PTPQ. Finally, sound, but not complete, heuristic approaches for PTPQ minimization are presented. These approaches gradually trade execution time for accuracy.

PROCESSING TECHNIQUES FOR PARTIAL TREE-PATTERN QUERIES ON XML DATA

by
Pawel Placek

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

August 2009

APPROVAL PAGE

PROCESSING TECHNIQUES FOR PARTIAL TREE-PATTERN QUERIES ON XML DATA

Pawel Placek

6/23/09

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

6/23/2009

Dr. Narain Gehani, Committee Member — Date
Dean, College of Computing Sciences, NJIT

6/23/2009

Dr. Jim Geller, Committee Member — Date
Professor, Department of Computer Science, NJIT

6/23/09

Dr. Michael Halper, Committee Member — Date
Professor, Department of Computer Science, Kean University

24 JUNE 2009

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

BIOGRAPHICAL SKETCH

Author: Pawel Placek

Degree: Doctor of Philosophy

Date: August 2009

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 2009
- Master of Science in Computer Science
  Jagiellonian University, Cracow, Poland, 1997

Major: Computer Science

Publications:

Dimitri Theodoratos, Pawel Placek, Theodore Dalamagas, Stefanos Souldatos, and Timos K. Sellis. Containment of Partially Specified Tree-Pattern Queries in the Presence of Dimension Graphs. *The VLDB Journal*, 18(1):233–254, 2009.

Pawel Placek, Dimitri Theodoratos, Stefanos Souldatos, Theodore Dalamagas, and Timos K. Sellis. A Heuristic Approach for Checking Containment of Generalized Tree-Pattern Queries. In *Proc. of the 17th ACM Intl. Conf. on Information and Knowledge Management*, pages 551–560, 2008.

Dimitri Theodoratos, Stefanos Souldatos, Theodore Dalamagas, Pawel Placek, and Timos K. Sellis. Heuristic Containment Check of Partial Tree-Pattern Queries in the Presence of Index Graphs. In *CIKM '06: Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 445–454, 2006.

Dimitri Theodoratos, Theodore Dalamagas, Pawel Placek, Stefanos Souldatos, and Timos K. Sellis. Containment of Partially Specified Tree-Pattern Queries. In *SSDBM '06: Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 3–12, 2006.

To My Parents

ACKNOWLEDGMENT

I would like to express my sincere appreciation to Dr. Dimitri Theodoratos, my research advisor, for his constant help and support, countless resources, insight and intuition, reassurance and encouragement. I especially appreciate Dr. Theodoratos' willingness to work flexibly with me during countless evening or late evening hours, or even, if necessary, during weekends, which allowed me to complete my research while still holding a full time job.

Special thanks are given to Dr. Narain Gehani, Dr. Jim Geller, Dr. Michael Halper and Dr. Vincent Oria for valuable comments and active participation in my committee.

I also wish to give my deepest thanks to my parents, Alina and Edward Placek, for all the motivation, encouragement, and support given to me throughout the years of my doctoral research and prior to it. Finally, I would like to thank my wife, Anna Placek, without whom I would not be able to successfully balance my family, professional, and academic obligations. Without her patience, help, support and encouragement I would not be able to complete this work.

TABLE OF CONTENTS

Chapter		Page
1	INTRODUCTION	1
	1.1 Motivation	2
	1.2 Contribution	4
	1.3 Outline	7
2	RELATED WORK	8
3	THE PARTIAL TREE-PATTERN QUERY LANGUAGE	12
	3.1 Data Model	12
	3.2 Query Language	16
	3.3 PTPQ Containment (Absolute) and PTPQ Containment in the Presence of Dimension Graphs (Relative)	22
	3.4 Structural Expression Inference and PTPQ Full Form	23
	3.5 PTPQ Satisfiability	26
4	CHECKING PTPQ CONTAINMENT IN THE PRESENCE OF DIMENSION GRAPHS	28
	4.1 Valid Partial Paths Clusters	28
	4.2 Checking PTPQ Containment With Respect to a Dimension Graph	33
	4.3 Heuristic Approaches for PTPQ Containment with Respect to a Dimension Graph	38
	4.3.1 The Basic Idea	38
	4.3.2 Precomputation Heuristic Approach	41
	4.3.3 On-the-fly Heuristic Approach	53
	4.4 Experimental Evaluation	54

TABLE OF CONTENTS
(Continued)

Chapter **Page**

4.4.1 Experimental Setup 54

4.4.2 Experimental Results 56

4.5 Conclusion 61

5 HEURISTIC APPROACHES FOR CHECKING CONTAINMENT OF PARTIAL TREE-PATTERN QUERIES 62

5.1 Query Language 62

5.2 Component TPQs 63

5.3 Necessary and Sufficient Conditions for PTPQ Containment 65

5.4 PTPQs for Which Homomorphisms Are Necessary for Containment . 68

5.4.1 A Subclass of PTPQs 71

5.5 A Heuristic Approach for Checking PTPQ Containment 73

5.5.1 Adjustment of a PTPQ with a Virtual PP 74

5.5.2 A Heuristic Using the Adjustment of a PTPQ 79

5.5.3 A Heuristic Using the Complete Adjustment of a PTPQ . . . 80

5.6 Experimental Evaluation 82

5.7 Conclusion 87

6 MINIMIZATION OF PARTIAL TREE-PATTERN QUERIES 89

6.1 The PTPQ Minimization Problem 89

6.1.1 Minimal PTPQs 89

6.1.2 Problem Addressed 92

6.1.3 Homomorphisms between PTPQs 92

6.1.4 Challenges in PTPQ Minimization 92

TABLE OF CONTENTS
(Continued)

Chapter **Page**

6.2 Minimizing a Subclass of PTPQs 95

6.2.1 Component PTPQs 95

6.2.2 Using cPTPQs to Characterize PTPQ Containment 97

6.2.3 cPTPQ Minimization 97

6.3 Heuristic Algorithms for PTPQ Minimization 98

6.3.1 Why PTPQ Minimization Cannot Be Achieved Using Homomorphisms to Identify Redundant Parts 98

6.3.2 Virtual Paths and Adjustments of PTPQs 101

6.3.3 Heuristic Algorithms for Minimizing PTPQs 103

6.4 Conclusion 108

7 CONCLUSION 109

LIST OF FIGURES

Figure		Page
3.1	Database T.	13
3.2	Database T_1.	14
3.3	Database T_2.	14
3.4	Dimension graph $\mathcal{G}$.	14
3.5	Dimension graph of the database T.	15
3.6	Dimension graph of the database T.	19
3.7	PTPQ Q_1.	19
3.8	The answer of Q_1 on T_2.	21
3.9	PTPQ Q_2.	23
3.10	PTPQ Q_3.	23
3.11	Full form of PTPQ Q_1.	25
3.12	Full form of PTPQ Q_2.	25
3.13	A set of inference rules.	25
4.1	Cluster C_3 (valid).	29
4.2	Cluster C_4 (valid).	29
4.3	The dimension trees of Q_1 on $\mathcal{G}$: (a) U_1^1, (b) U_1^2.	36
4.4	The dimension trees of Q_2 on $\mathcal{G}$: (a) U_2^1, (b) U_2^2.	37
4.5	The dimension trees of Q_3 on $\mathcal{G}$: (a) U_3^1, (b) U_3^2.	38
4.6	PTPQ Q_3'.	39
4.7	PTPQ Q_3', the augmented PTPQ Q_3 with respect to $\mathcal{G}$ and $\mathcal{R}$.	44
4.8	PTPQ Q_2'.	45
4.9	PTPQ Q_3'', the augmented PTPQ Q_3 with respect to $\mathcal{G}$ and $\{\mathcal{R}, \mathcal{R}'\}$.	46

LIST OF FIGURES
(Continued)

Figure **Page**

4.10 (a) PTPQ Q_4, (b) Augmented form of Q_4 w.r.t. $\mathcal{G}$. 54

4.11 Execution time for checking PTPQ containment varying the number of root-to-leaf paths for 10, 20, 30 and 40 nodes in the dimension graph. 57

4.12 Execution time for checking PTPQ containment varying the number of nodes per PP for 1, 2, 3 and 4 PPs in the PTPQ. 58

4.13 Percentage of correct answers in checking relative PTPQ containment varying the number of root-to-leaf paths for 10, 20, 30 and 40 nodes in the dimension graph. 59

4.14 Percentage of correct answers in checking relative PTPQ containment varying the number of nodes per PP for 1, 2, 3 and 4 PPs in the PTPQ. 60

5.1 PTPQ Q_1. 63

5.2 PTPQ Q_2. 63

5.3 PTPQ Q_3. 63

5.4 PTPQ Q_1', the full form of Q_1. 64

5.5 PTPQ Q_2', the full form of Q_2. 64

5.6 The four cTPQs of PTPQ Q_2. 65

5.7 (a) PTPQ Q_4, a PTPQ with two 3-path swings, (b) PTPQ Q_5. . . . 70

5.8 (a) PTPQ Q_6, a PTPQ with a 2-path swing, (b) PTPQ Q_7. 71

5.9 (a) The virtual PP v of p_5 w.r.t. p_4 and p_6 in Q_2, (b) The adjustment Q_2'' of Q_2' with v, (c) the full form Q_2''' of Q_2'', (d) PTPQ Q_3, and an "outline" of a homomorphism from Q_3 to Q_2'''. 75

5.10 (a) The virtual PP v of p_2 w.r.t. p_1 and p_3 in Q_4, (b) The adjustment Q_4' of Q_4 with v. 76

LIST OF FIGURES
(Continued)

Figure **Page**

5.11 (a) The virtual PP v_1 of p_1 and p_2 in Q_6, (b) The adjustment Q'_6 of Q_6 with v. 78

5.12 Computation of the adjustment Q_a of a PTPQ Q. 79

5.13 (a) PTPQ Q_8, (b) The adjustment Q_{8a} of Q_8, (c) PTPQ Q_9. 80

5.14 Computation of the complete adjustment Q_{ca} of a PTPQ Q. 81

5.15 (a) the complete adjustment Q_{8ca} of Q_8, (b) PTPQ Q_{10}. 82

5.16 Execution time for checking PTPQ containment varying the size of the queries. 84

5.17 Percentage of correct answers in checking PTPQ containment varying the size of the queries. 85

5.18 Execution time for checking PTPQ containment varying the density of swings in the queries. 86

5.19 Percentage of correct answers in checking PTPQ containment varying the density of swings in the queries. 87

6.1 Two equivalent PTPQs. 90

6.2 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q). 91

6.3 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q). 91

6.4 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q). 91

6.5 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q). 93

6.6 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q). 94

6.7 Two equivalent minimal queries. 94

6.8 Three minimal and equivalent queries. 94

6.9 A PTPQ Q. 96

6.10 The two cPTPQs for the PTPQ Q of Figure 6.9. 96

LIST OF FIGURES
(Continued)

Figure **Page**

6.11 A PTPQ Q with a redundant PP. 99

6.12 A PTPQ Q with a redundant PP. 100

6.13 The adjustment of the PTPQ Q of Figure 6.11 with a virtual PP v. . 101

6.14 The adjustment of the PTPQ Q of Figure 6.12 with a virtual PP v. . 102

6.15 Heuristic algorithm 1. 104

6.16 A PTPQ Q. 105

6.17 The adjustment of the PTPQ Q of Figure 6.16. 105

6.18 Heuristic algorithm 2. 107

6.19 The complete adjustment of the PTPQ Q of Figure 6.16. 107

CHAPTER 1

INTRODUCTION

In recent years, XML (eXtensible Markup Language) [60] has been used as a de facto standard for interchanging data between various types of databases and Web sites. XML organizes data in documents which have a tree-structured form. It is capable of handling diverse data sources including sources with structured and semi-structured documents, relational databases, and object repositories. XML is software and hardware independent. It has been widely adopted by businesses, enterprises, as well as educational and governmental institutions. Some regulatory and government agencies not only use, but require that all data submitted to them be in the XML format. A prime example of such a government body is the Food and Drug Administration (FDA).

The broad range of XML applications include inter-application data exchange, streaming data, metadata management, Web publishing and searching, e-business applications, and pervasive (wireless) computing. The World Wide Web Consortium (W3C), founded in 1994, is an international consortium devoted to developing Web standards. Its mission is to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web. W3C has developed several widely used technologies for XML, which make the use of XML in current form possible and efficient. One of the attractive features of XML is its dissociation of data from schema. This is paramount for applications requiring flexible or no schemas as it is usually the case on the Web. Nevertheless, W3C developed schema definition languages, Document Type Definition (DTD) and XML Schema, which may be used to provide a structure, content and semantics for XML documents. Once an XML document conforms to a DTD or XML Schema, it is much easier to

query, process or exchange it. Since XML has become a data model, a need to query XML documents has emerged. W3C suggested XQuery [58] as a language to query XML document trees. XQuery is a powerful and complex language, which allows to query XML sources. According to W3C's Web site, XQuery is to XML what SQL is to relational databases. W3C also suggested the Extensible Stylesheet Language (XSL) and the XSL Transformations (XSLT) [59] languages for transforming XML documents into other XML documents. Working in conjunction, they allow to display XML data in different applications or on different devices. For instance, data can be displayed on a PDA, a cell phone, in a browser on a computer, or simply be printed out. For the sake of completeness one has to mention XML Linking Language (XLink) and XML Pointer Language (XPointer) [56]. They allow to create hyperlinks within XML documents as well as hyperlinks to other resources on the Web such as other XML documents, files other than XML, database searches, etc. Finally, W3C proposed another query language for XML, XPath [57]. XPath is a simple query language for navigating in XML documents, but it is an integral part and lies at the core of XQuery, XSLT, XPointer, or XLink. For this reason XPath is a target of query evaluation, processing and optimization techniques.

1.1 Motivation

The introduction of a tree-structured data model has created a need for efficient methods for querying this type of data. New languages proposed to address this need include XML-QL [14], XQL [43], Lorel [1], Quilt [11], XQuery, and XPath. All currently used languages for querying tree-structured data rely on tree patterns. Answers to the queries are computed by matching these tree patterns against the data trees. The answer is either a set of nodes or an XML tree. The effective and efficient querying of tree structured data faces several obstacles: (a) a tree-structured data

source may contain structured data (i.e., relational database) along with unstructured data (i.e., text), (b) the user may not know the (full) tree structure of a data source, (c) the structure is flexible and complex and evolves over time, (d) there is a need to query in an integrated way several data sources with different structures. Different techniques were proposed in the past to address these challenges. To cope with these problems some researchers tried to use approximation techniques. A number of approaches proposed query relaxation techniques that generate alternative forms of the queries and search for their answers in the data sources [4, 5]. Others suggested approximating the answers of XML queries [23, 41]. Another approach proposed flexible and semi-flexible semantics for tree pattern queries [26]. In any case, all these approaches have a drawback: the answer is not exact with respect to the initial query. Other approaches try to cope with the problems mentioned above using keyword search techniques [13, 32, 65]. These techniques are very popular. However, by completely ignoring the structure, they generate a different set of drawbacks: (a) structural information cannot be specified within the query to accelerate computation of an answer, (b) structural conditions cannot be imposed to filter out undesirable answers, (c) the structure of the result of the query cannot be specified. Other attempts to cope with the previous problems try to combine XML query languages with keyword-based search techniques [19, 31]. These languages, however, seem too complex for a simple user. Finally, the approach of providing a global structure [2, 12], the most widely used technique to cope with the issue of querying multiple data sources, requires extensive manual effort. This is due to the fact that the global schema is usually difficult to construct and all mappings between global schema and local schemas must be hard-coded in the integration application.

An approach proposed by Theodoratos et al. [47] addresses the challenges mentioned above in a different way. The authors define a flexible language allowing partial specification of a tree pattern query. The novelty of that language lies in

its ability of querying tree-structured data whose structure is complex or not fully known to the user. It also allows querying tree-structured data from different sources with structural differences. The language distinguishes itself from approximation techniques, because it returns an exact and not approximate answer. It also does have an edge over keyword search techniques since it allows partial or complete specification of the tree structure in the queries. This language expresses a large subset of XPath and it flexibly allows in a query pattern the specification of a full tree structure, a partial tree structure, or no structure at all. Therefore, queries in that language range from structureless keyword-based queries to completely specified tree patterns.

For a query language to be useful, it needs to be complemented with query processing and optimization techniques. This explains why studying these techniques is one of the major areas of research in tree-structured databases. Query processing involves addressing query satisfiability issues [7, 8, 25, 30], query minimization issues [3, 18, 29, 42, 54], rewriting queries using views issues [10, 24, 64], and query containment issues [46, 15, 33, 35, 36, 44, 55]. Solving these problems efficiently is central to the optimal evaluation of queries over tree-structured data.

Previous research related to query processing and optimization focuses on tree-pattern queries. In a tree-pattern query the structure of a tree pattern has to be fully specified. None of the previous methods can be applied to partially specified tree-pattern queries. Therefore, novel efficient query processing techniques have to be developed in order to fully benefit from the potential of the partial tree-pattern query language.

1.2 Contribution

Our primary goal in this thesis is to address the satisfiability, containment and minimization problems for different fragments of partial tree-pattern queries (PTPQs).

Our approach is not to provide an in-depth study of the complexity for different problems and different fragments of the language. Instead we target effective query processing techniques (possibly using heuristics) that can be used in practice, for instance, by query optimizers.

Our language represents tree-structured data by considering trees of values whose nodes are partitioned to form what we call *dimensions*. We use dimensions to define *dimension graphs*, a construct that summarizes the structural information of the database. A dimension graph can be automatically extracted from a database and supports the processing and evaluation of the PTPQs. We defined the concept of containment of two PTPQs and the concept of containment of two PTPQs with respect to a dimension graph. We called the first type of containment *absolute containment* and the second one *relative containment*. In order to handle structural expression inference and to allow query comparison, we defined a "normal form" for PTPQs, called *full form*. We further introduced the concept of *homomorphism* between two PTPQs to characterize their containment.

We initially addressed the absolute and relative containment problems for a structural subclass of PTPQs that involves value predicates. We devised techniques for checking relative query containment. Our results were experimentally evaluated and published in [48]. Next, we considered a more restricted class of PTPQs, which does not comprise value predicates. We addressed the satisfiability, and absolute and relative containment problems for this class of PTPQs. In order to deal with the inference of structural relationships we developed a set of inference rules. We provided necessary and sufficient conditions for absolute and relative query containment. We devised sound but not complete heuristic approaches for an efficient solution of the containment problem with respect to a dimension graph. These heuristics exploit structural information extracted from the dimension graph. An extensive experimental evaluation showed that our approaches greatly improve the containment checking

execution time while maintaining high accuracy. Our approaches also gradually trade execution time for accuracy. These results suggest that our techniques can be used by query processors and optimizers. This work was published in [50].

We also generalized our previous results to study the relative containment problem for an unrestricted class of PTPQs. We showed that PTPQ homomorphisms do not fully characterize the PTPQ relative containment and we provided heuristic approaches for the relative containment problem. This part of our work was published in the VLDB Journal [49].

Subsequently, we studied the containment in the absence of dimension graphs (absolute containment problem) for a class of PTPQs without value predicates. We provided necessary and sufficient conditions for PTPQ containment. We showed that a PTPQ can be equivalently represented by a set of tree-pattern queries. We also defined a broad subclass of PTPQs (strictly containing tree-pattern queries) for which a homomorphism is a sufficient and necessary condition for PTPQ containment. For PTPQs for which homomorphisms do not fully characterize containment we suggested two new heuristic techniques for absolute containment that equivalently add virtual partial paths to PTPQs. An empirical evaluation of these techniques demonstrated that they maintain high accuracy while being orders of magnitude faster than PTPQ containment check without using heuristics. Some of these results were presented in [39]. The complete results are submitted to a journal.

Finally, we addressed the problem of minimizing PTPQs. Minimizing queries is important, because it may significantly affect the evaluation time of the query. The minimization problem for PTPQs is challenging for two main reasons. First, as we showed, PTPQs cannot be minimized by removing redundant parts even though this is the case with tree-pattern queries involving branching and descendant relationships. Second, a PTPQ does not have, in general, a unique minimal equivalent PTPQ. We proved that a PTPQ can be equivalently represented by a special type of PTPQs,

called *component PTPQs*. Component PTPQs can be used to check PTPQ equivalence through the existence of homomorphism from a PTPQ to component PTPQs. We devised sound but not complete heuristic approach for minimizing PTPQs, which proceeds by first equivalently adding partial paths to a PTPQ and then removing redundant parts identified through homomorphisms.

1.3 Outline

The next chapter reviews related work. Chapter 3 contains the data model and query language, and definitions of the containment and satisfiability problems. It also includes inference rules and a definition of the full form of a PTPQ along with necessary and sufficient conditions for PTPQ satisfiability. Chapter 4 presents all results relevant to the relative containment of PTPQs. It also provides heuristics for checking relative containment and their experimental evaluation. Chapter 5 contains results related to absolute PTPQ containment, heuristics for checking absolute PTPQ containment, and empirical results. Results for minimizing PTPQs along with a heuristic approach are included in Chapter 6. Our conclusions are presented in Chapter 7.

CHAPTER 2

RELATED WORK

The wide use of XML raised the need for robust and efficient tools for querying tree-structured data. To cope with this issue many query languages were proposed. Key examples include: XML-QL [14], XQL [43], Lorel [1], and XPath [57]. Chamberlin et al. [11] also proposed a query language for XML called Quilt. According to the authors, Quilt has pulled features from other languages that have strengths in specific areas. From XPath and XQL they pulled syntax for navigating hierarchical documents. From XQL-QL they took the notion of variable binding and used it to create new structures. They also adopted some ideas used in relational languages (SQL) and object languages (OQL). With the proliferation of query languages aimed at query XML data, W3C formed the W3C XML Query Working Group to coordinate activities on developing standard query languages for XML. W3C XML Query Working Group adopted Quilt as the basis for the development of XQuery [58]. XPath and XQuery, recommended by W3C, became de facto standard languages for querying XML data. Besides XQuery and XPath, which are general query languages for XML, other specialized languages were presented for specific problems. For instance, XML-DMQL [16] is a data mining query language for XML. The answer of XPath query is a set of nodes. Some problems require as an answer a set of XML subtrees. To this end an XPath-based subtree query language was presented in [9]. As discussed in the previous chapter, there is a need for a query language that allows a flexible specification of queries. The partial tree-pattern query (PTPQ) language proposed in [47] and presented later in this work fulfills this need. Optimization techniques for evaluating PTPQs were presented in [45, 53, 61, 62, 63]. Approaches for assigning meaningful semantics to PTPQs were discussed in [51, 52].

One of the central issues in query processing is the query containment problem. Miklau and Suciu [33] study query containment for different classes of XPath queries. The fragment of XPath they analyze consists of node tests, the descendant axis (//), wildcards (*), and predicates ([]). They prove that the containment problem for this class of queries, denoted $XP^{([],*,//)}$, is co-NP complete. They also provide two algorithms for checking containment in this subclass of XPath. The first algorithm is an efficient, sound, but not complete algorithm. The second one is sound and complete algorithm, which does not run in polynomial time. Subsequently, they prove that the complexity of the problem is co-NP. They also define and discuss subclasses of queries, for which the second algorithm runs in polynomial time.

Wood [55] studies containment for different XPath fragments under Document Type Definitions (DTDs). DTDs constrain the structure of XML documents. As expected, problems in the presence of these restrictions become more complex. Wood shows that the containment problem of $XP^{([],*,//)}$ queries in the presence of DTDs is decidable. He also shows that the complexity of this problem is EXPTIME-complete. Wood also studies subclasses of DTDs, for which containment for $XP^{([])}$ can be tested in PTIME. He also shows that containment for $XP^{([])}$ under DTDs is coNP-complete [35].

Neven and Schwentick [35] present an in-depth analysis of the complexity of containment for many different classes of XPath. They prove that the containment in $XP^{([],*,/,//,|)}$, $XP^{(//,|)}$, and $XP^{(/,|)}$ is coNP-complete. The symbol "|", in this context, represents disjunction *(logical or)*. In addition, they prove that if the alphabet of node labels is finite then $XP^{([],*,/,//,|)}$ and $XP^{(/,//,|)}$ are PSPACE-complete. The authors study also the containment for a fragment of XPath under the presence of DTDs. They obtain that containment for $XP^{(/,//)}$ under DTDs can be tested in PTIME. However, by adding branching to this fragment of XPath the complexity of containment checking increases. The authors show that under DTDs, $XP^{(/,[])}$ is

coNP-complete and $XP^{(//,[])}$ is coNP-hard. Subsequently they prove that for broader XPath classes, the problem of containment checking becomes even harder. They also prove that if DTDs are considered, the containment problem in $XP^{([],*,/,//,|)}$ and $XP^{(/,//,|)}$ is EXPTIME-complete.

Part of this thesis focuses on checking PTPQ containment in the presence of a new construct called *dimension graph*. Dimension graphs are summaries of tree-structured data and they are used to support the evaluation of the PTPQs and the satisfiability and containment checking. Similar concepts have been referred to with different names in the literature, including "index graphs", "path summaries", "path indexes" and "structural summaries". They differ in the equivalence relations they employ to partition the nodes of the data tree [27, 34]. In dimension graphs, however, the nodes in the data tree are partitioned based on semantic relations provided by the user. This consideration is general and encompasses syntactic partitioning. For instance, in an XML tree, the equivalence classes can be formed by all the nodes labeled by the same element. Summaries of data have been extensively studied in recent years in both the "exact" [6, 20, 34, 40] and the "approximate" flavor [27, 28]. A common characteristic of those approaches is that the data summary is used as a back end for evaluating a class of path expressions without accessing the data tree. To this end, the equivalence classes of nodes are attached to the corresponding nodes of the data summary. In contrast to the previous approaches, the equivalence classes of the data tree nodes are not kept with the dimension graph. Therefore, partially specified tree-pattern queries are ultimately evaluated on the data tree.

The query minimization problem is another important topic in query processing. Amer-Yahia et al. [3] analyze the problem of query minimization for queries in $XP^{([],/,//)}$ both without constraints and with specific constraints (required-child and required-descendant). For the first case they provide an $O(n^4)$ algorithm. For the latter case they provide an $O(n^6)$ algorithm. This problem was studied further

by Ramanan in [42] where improvements to minimization algorithms were proposed. Ramanan designed an $O(n^2)$ algorithms for query minimization in both these cases. Flesca and Furfaro [18] study minimization in the fragment of $XP^{([],/,//,*)}$. An additional constraint they impose on this XPath fragment requires that branching nodes are not labeled by "*". Among other results they show that the decision problem "*given a tree pattern p, is p of minimum size?*" is NP-complete in this fragment of XPath.

Query processing problems in tree-pattern languages are complex problems and draw a lot of interest in the research community. Previous research, however, is restricted to tree-pattern queries. Further, none of the previous approaches provided heuristic techniques for these problems. In the next sections we will present results on checking partial tree-pattern queries satisfiability, containment, and minimization. We will also provide heuristic approaches for checking PTPQ containment in presence and absence of dimension graphs and two heuristics for PTPQ minimization.

CHAPTER 3

THE PARTIAL TREE-PATTERN QUERY LANGUAGE

We present in this chapter our data model and query language. This language is based on the query language presented by Theodoratos et al. in [47]. The data model represents tree-structured data using the concepts of database, dimension and dimension graph. The query language allows for partially specified tree patterns. We define the containment and satisfiability problems of partial tree-pattern queries (PTPQs) in the presence and absence of dimension graphs. In order to allow PTPQ comparison we also define a "normal form" for PTPQs, called a *full form*. We provide inference rules to characterize inference of structural expressions.

3.1 Data Model

We assume an infinite set of values V that includes a special value r.

Dimensions and databases. A *dimension set* over V is a partition $\mathcal{D}$ of V that includes the singleton $\{r\}$. Each element of $\mathcal{D}$ is called *dimension* of $\mathcal{D}$. The dimensions in $\mathcal{D}$ are assigned distinct names. In particular, the dimension $\{r\}$ is named R. Intuitively, a dimension is a set of semantically related values. For instance, $AUTHOR$ can be a dimension that includes authors of bibliographic entries. Since the names of the dimensions are distinct we use them to identify the dimensions of $\mathcal{D}$. Different applications may require and apply different partitions of the values in V. For the needs of this chapter, we assume that a dimension set is fixed and we denote it by $\mathcal{D}$.

Definition 3.1.1. *A* database *over $\mathcal{D}$ is a finite and rooted node-labeled tree T, such that: (a) each node label in T belongs to V, (b) value r labels only the root of T, and (c) there are no two nodes on a path in T labeled by values that belong to the same dimension in $\mathcal{D}$. In this sense, databases are not recursive.* □

We assume that nodes in a database have a unique node identifier and these node identifiers are preserved in the answers of a query on this database.

Example 3.1.1. *Let $\mathcal{D} = \{R, AUTHOR, BIB, YEAR, TITLE, SUBJECT\}$, where dimensions AUTHOR, BIB, YEAR, TITLE and SUBJECT include respectively the author name, type, publishing year, title, and subject of bibliographic entries. Figure 3.1 shows the database T. In the database, dimension of a value is shown in capital letters by the value.*

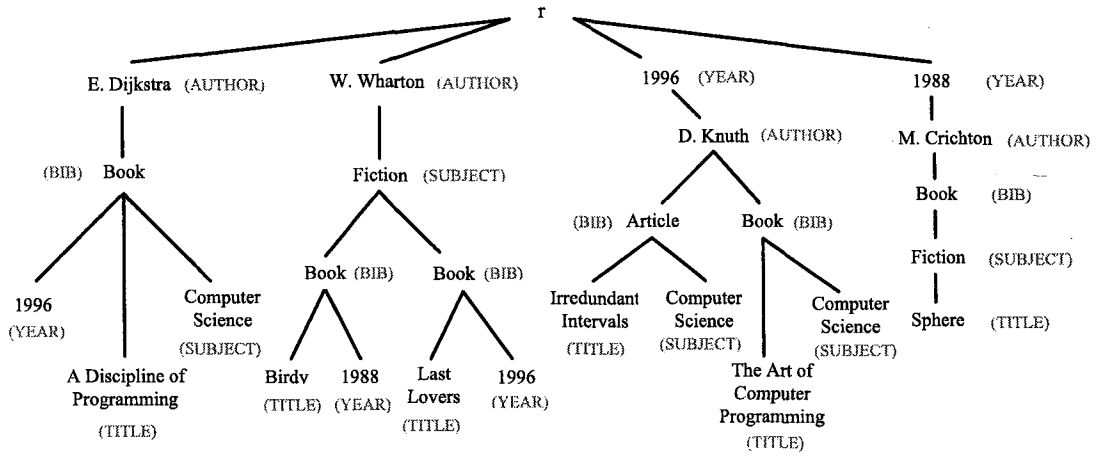

Figure 3.1 Database T.

Values from two different dimensions can appear in different order in different branches of a database. For instance, value "1996" of dimension "YEAR" is an ancestor of value "D. Knuth" of dimension "AUTHOR" in the third branch of tree T, while value "1988" of dimension "YEAR" is a descendant of value "W. Wharton" of dimension "AUTHOR" in another branch of T. □

The next example uses abstract notation, which will be used throughout the rest of this work.

Example 3.1.2. *Let $\mathcal{D} = \{R, A, B, C, D, E\}$. Unless otherwise specified, this will also be the dimension set for the rest of the examples. Figures 3.2 and 3.3 show two databases T_1 and T_2 respectively.*

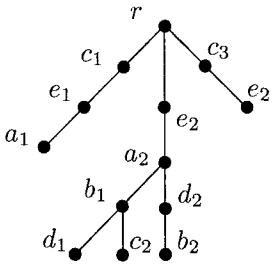

Figure 3.2 Database T_1.

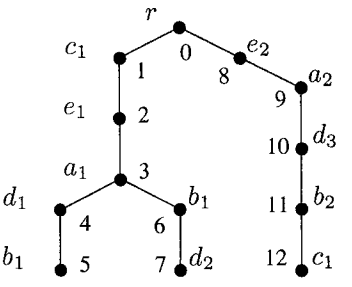

Figure 3.3 Database T_2.

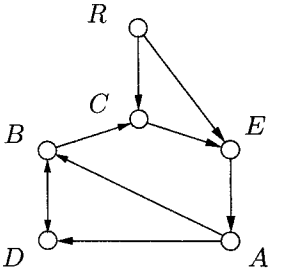

Figure 3.4 Dimension graph $\mathcal{G}$.

All the x_is in a database denote (not necessarily distinct) values of the same dimension $X \in \mathcal{D}$. For instance, values a_1 and a_2 are values of dimension A. The numbers by the nodes of the database T_2 of Figure 3.3 denote their identifiers.

As in previous example, values from two different dimensions can appear in different order in different branches of a database. For instance, value c_1 of dimension C is an ancestor of value a_1 of dimension A in the leftmost branch of tree T_1, while value c_2 of dimension C is a descendant of value a_2 of dimension A in another branch of T_1. □

The semantic interpretation of the values of a database into dimensions is provided by a user, possibly assisted by an ontology. Note, however, that dimensions can also be chosen to represent purely syntactic objects. For instance, they can be viewed as sets that group together nodes in the database that are labeled by the same element. In this case, the concept of a dimension graph is similar to that of an index graph [26].

Dimension graphs. The values of some dimension may not be children or descendants of any value of some other dimension in a database. For instance, no value of

dimension A in the databases T_1 and T_2 of Figures 3.2 and 3.3 is a child of a value of a dimension other than E. We use the concept of dimension graph to capture this type of relationship between dimensions in a database.

Definition 3.1.2. *Let T be a database over $\mathcal{D}$. A* dimension graph *$\mathcal{G}$ of T is a graph (N, E), where N is a set of nodes and E is a set of edges, defined as follows: (a) there is a node D in N if and only if there is a value in T that belongs to dimension D, and (b) there is a directed edge (D_i, D_j) in E if and only if there are nodes n_i and n_j in T labeled by values $v_i \in D_i$ and $v_j \in D_j$ respectively, such that n_j is a child of n_i in T. If $\mathcal{G}$ is a dimension graph of a database T, we say that T* underlies *$\mathcal{G}$.* □

The dimension graph of a database may have cycles. In particular, it can have a trivial cycle if, in the underlying database, a value of a dimension labels a parent node of a node labeled by a value of another dimension and conversely.

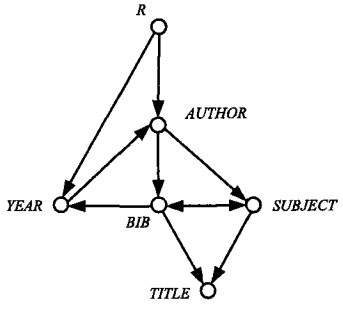

Figure 3.5 Dimension graph of the database T.

Example 3.1.3. *Figure 3.4 shows the dimension graph of the databases T_1 and T_2 of Figures 3.2 and 3.3. Trivial cycles are shown in the figures with a double headed edge (e.g. the edge between dimensions D and B in Figure 3.4). Figure 3.1 shows the dimension graph of database T of Figure 3.1.* □

The next proposition provides properties that fully characterize dimension graphs on databases. In the proposition and the following text, paths are meant to be *simple* that is, they do not meet the same node twice.

Proposition 3.1.1. *A directed graph $\mathcal{G}$ whose nodes are dimensions is a dimension graph of some database if and only if the following properties hold:*

(a) Graph $\mathcal{G}$ does not have disconnected components.

(b) There is exactly one node in $\mathcal{G}$ having only outgoing edges. We call this node root *of $\mathcal{G}$.*

(c) For every directed edge in $\mathcal{G}$ there is a path from the root of $\mathcal{G}$ that comprises this edge. □

Proof The *only if* part is a direct consequence of the definition of a dimension graph. For the *if part* let's assume that a directed graph $\mathcal{G}$ satisfies the conditions (a), (b), and (c) of the proposition. We construct a database by considering all the simple paths from the root of $\mathcal{G}$ and by merging their root node R. Clearly, this tree is a database that underlies $\mathcal{G}$. Therefore, $\mathcal{G}$ is a dimension graph. □

Dimension graphs can be automatically extracted from databases and abstract their structural information. As we show in subsequent sections, they help the evaluation of queries on databases, the detection of unsatisfiable queries and the checking of PTPQ containment.

3.2 Query Language

Partial tree-pattern queries (PTPQs) are issued on dimension sets and are evaluated on databases. Dimension graphs can support the formulation of PTPQs. To allow PTPQ composition, we require that the evaluation of a PTPQ on a database yields a database.

Syntax. A PTPQ on a dimension set provides a (possibly partial) specification of a tree of dimensions annotated with sets of values. The tree is rooted at dimension R. A PTPQ specifies such a tree through a set of (possibly partially specified) paths from the root of the tree. For succinctness, in the following, *PP* stands for *partial path.* In

some figures PP can be also denoted as *PSP*. Each PP is defined in a PTPQ by a set of annotated dimensions, and a set of precedence relationships (child and descendant relationships) among these annotated dimensions. A PTPQ further indicates nodes (annotated dimensions) that are shared among different PPs in the PTPQ tree. It also identifies a distinguished PP called output PP. The formal definition follows.

Definition 3.2.1. *A* PTPQ *on a dimension set $\mathcal{D}$ is a triple $(\mathcal{P}, \mathcal{S}, o)$, where:*

(a) $\mathcal{P}$ is a nonempty set of triples $(p, \mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ and $\mathcal{R}$ define a PP as explained below, and p is a distinct name for this PP. Since PP names are distinct, we identify PPs with their names.

(a1) $\mathcal{A}$ is a set of expressions of the form $D[p] = V$, where $D[p]$ denotes the dimension D of $\mathcal{D}$ in PP p, and V is a set of values of dimension D or a question mark ("?"). These expressions are called annotating expressions *of p. If the expression $D[p] = V$ belongs to $\mathcal{A}$ we say that D is* annotated *in p and V is its* annotation. *A dimension can be annotated only once in a PP p. Without mentioning it explicitly, we assume that dimension R is annotated with a "?" in every PP. Set $\mathcal{A}$ can be empty.*

(a2) $\mathcal{R}$ is a set of expressions of the form $D_i[p] \rightarrow D_j[p]$ or $D_i[p] \Rightarrow D_j[p]$, where D_i is an annotated dimension in $\mathcal{A}$ or R, and D_j is an annotated dimension in $\mathcal{A}$. These expressions are called precedence relationships *of p. Set $\mathcal{R}$ can be empty.*

(b) $\mathcal{S}$ is a set of expressions of the form $D[p_i] \equiv D[p_j]$, where p_i and p_j are PPs in $\mathcal{P}$, and D is a dimension annotated in p_i and p_j. These expressions are called node sharing expressions. *Roughly speaking, they state that PPs p_i and p_j have a dimension in common. Set $\mathcal{S}$ can be empty.*

(c) o is the name of one of the PPs in $\mathcal{P}$. This PP is called output PP *of the PTPQ.*

□

The term *structural expression* refers indiscreetly to a precedence relationship or to a node sharing expression.

Example 3.2.1. *Consider the database T from Example 3.1.1 and a query: "Find all books (value of BIB) in the database, which lie in the same path as their SUBJECT and TITLE values, and are above their YEAR value". This query can be expressed by the following PTPQ:*

$Q = (\mathcal{P}, \mathcal{S}, p_1)$, *where*

$\mathcal{P} = \{(p_1, \mathcal{A}_1, \mathcal{R}_1), (p_2, \mathcal{A}_2, \mathcal{R}_2)\}$,

$\mathcal{A}_1 = \{BIB[p_1] = \{Book\}, SUBJECT[p_1] = ?,\ TITLE[p_1] = ?\}$,

$\mathcal{R}_1 = \emptyset$,

$\mathcal{A}_2 = \{BIB[p_2] = ?,\ YEAR[p_2] = ?\}$,

$\mathcal{R}_2 = \{BIB[p_2] \Rightarrow YEAR[p_2]\}$,

$\mathcal{S} = \{BIB[p_1] \equiv BIB[p_2]\}$. □

We graphically represent queries using graph notation. Consider a PTPQ Q. Each PP of Q is represented as a (not necessarily connected) graph of dimensions labeled by their annotating expressions in the PP. The name of each PP is shown below the corresponding PP graph. In particular, the name of the output PP of Q is preceded by a $\star$. PP names are omitted in the annotating expressions for succinctness. Child and descendant precedence relationships in a PP are depicted using single ($\rightarrow$) and double ($\Rightarrow$) arrows between the respective nodes in the PP graph. Two nodes (annotated dimensions) in different PP graphs that participate in a node sharing expression of Q are linked in its graphical representation with a straight line labeled by the symbol '$\equiv$'.

Example 3.2.2. *PTPQ Q of Example 3.2.1 is graphically represented in Figure 3.6.* □

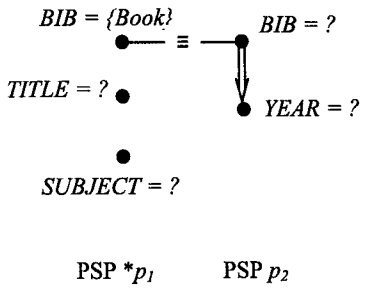

Figure 3.6 Dimension graph of the database T.

Example 3.2.3. *Consider the following PTPQ on $\mathcal{D}$: $Q_1 = (\mathcal{P}, \mathcal{S}, p_1)$, where*

$\mathcal{P} = \{(p_1, \mathcal{A}_1, \mathcal{R}_1), (p_2, \mathcal{A}_2, \mathcal{R}_2), (p_3, \mathcal{A}_3, \mathcal{R}_3)\}$,

$\mathcal{A}_1 = \{A[p_1] =?, B[p_1] = \{b_1\}, C[p_1] = \{c_1\}, D[p_1] =?\}$,

$\mathcal{R}_1 = \{A[p_1] \Rightarrow B[p_1]\}$

$\mathcal{A}_2 = \{A[p_2] = ?,\ C[p_2] = \{c_1, c_2\},\ E[p_2] = ?\}$,

$\mathcal{R}_2 = \{C[p_2] \Rightarrow A[p_2],\ E[p_2] \rightarrow A[p_2]\}$

$\mathcal{A}_3 = \{C[p_3] = ?,\ D[p_3] = ?\}$,

$\mathcal{R}_3 = \{D[p_3] \Rightarrow C[p_3]\}$, *and*

$\mathcal{S} = \{C[p_1] \equiv C[p_2]\}$. □

Example 3.2.4. *PTPQ Q_1 of Example 3.2.3 is graphically represented in Figure 3.7. The graphical representation of other queries is shown in Figures 3.9 and 3.10.* □

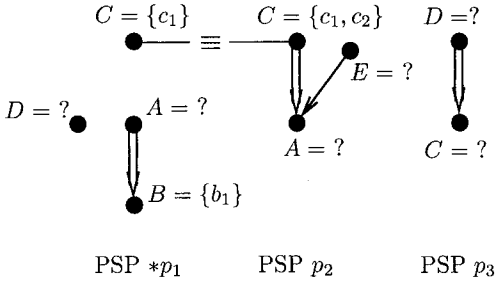

Figure 3.7 PTPQ Q_1.

Semantics. The answer of a PTPQ is based on the concept of PTPQ embedding.

Definition 3.2.2. *Let T be a database over a dimension set $\mathcal{D}$, and Q be a PTPQ on $\mathcal{D}$. An* embedding *of Q into T is a mapping M of the annotated dimensions of the PPs of Q to nodes in T such that:*

(a) The annotated dimensions of a PP in Q are mapped to nodes in T that are on the same path from the root of T.

(b) For every annotating expression $D[p] = V$ in Q, the label of $M(D[p])$ is a value in V, if V is a set, and it is a value of D, if V is a "?".

(c) For every precedence relationship $D_j[p] \rightarrow D_k[p]$ (resp. $D_j[p] \Rightarrow D_k[p]$) in Q, $M(D_k[p])$ is a child (resp. descendant) of $M(D_j[p])$ in T.

(d) For every node sharing expression $D[p_i] \equiv D[p_j]$ in Q, $M(D[p_i])$ and $M(D[p_j])$ coincide. □

A PTPQ can have more than one embedding into a database. Given an embedding M of a PTPQ Q into a database T, and a PP p in Q, the path from the root of T that comprises all the images of the annotated dimensions of p under M and ends in one of them is called *image* of p under M and is denoted $M(p)$. Notice that more than one PP of Q may have their image in the same root-to-leaf path of T (M does not have to be a bijection).

The *image of PTPQ Q* under M is the subtree of T rooted at r that comprises exactly the images of all PPs of Q under M.

The *answer of a PTPQ Q* on a database T is a database. Every path from the root to a leaf of the answer of Q on T is the image of the output path of Q under an embedding of Q into T that preserves the precedence relationships and node sharing expressions of Q. Note that the answer of a PTPQ is defined here differently than in XPath where the answer of an expression is a set of nodes.

Definition 3.2.3. *Let T be a database over a dimension set $\mathcal{D}$, and $Q = (\mathcal{P}, \mathcal{S}, o)$ be a PTPQ on $\mathcal{D}$. The* answer *of Q on T is a tree T' such that:*

(a) T' results by removing (possibly 0) paths from T.

(b) For every embedding of Q into T, the image of the output PP of Q is in T'.

(c) Every root-to-leaf path of T' is the image of the output PP of Q under an embedding of Q into T.

If there is no such a tree T', the answer of Q on T is an empty tree, and we say that the answer of Q on T is empty. □

Note that annotating a dimension with a "?" in a PP of a PTPQ is different than omitting this dimension from the PP. A dimension of a PP that is annotated with a "?" requires one of its values to be in the image of the PP under every PTPQ embedding into the database.

Example 3.2.5. *Consider the PTPQ Q_1 of Example 3.2.3, graphically shown in Figure 3.7. Consider also the database T_2 of Figure 3.9. The answer of Q_1 on T_2 is shown in Figure 3.8.*

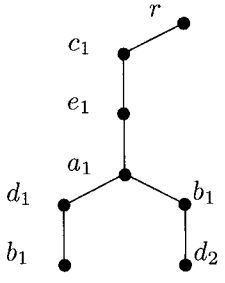

Figure 3.8 The answer of Q_1 on T_2.

There are two embeddings of Q_1 into T_2 which result in two distinct root-to-leaf paths in the answer of Q_1 on T_2. The embeddings are as follows:

Embedding 1: $C[p_1] \to 1$, $A[p_1] \to 3$, $B[p_1] \to 5$, $D[p_1] \to 4$, $C[p_2] \to 1$, $E[p_2] \to 2$, $A[p_2] \to 3$, $D[p_3] \to 10$, $C[p_3] \to 12$.

Embedding 2: $C[p_1] \rightarrow 1$, $A[p_1] \rightarrow 3$, $B[p_1] \rightarrow 6$, $D[p_1] \rightarrow 7$, $C[p_2] \rightarrow 1$, $E[p_2] \rightarrow 2$, $A[p_2] \rightarrow 3$, $D[p_3] \rightarrow 10$, $C[p_3] \rightarrow 12$.

Note that both embeddings map the dimensions of PPs p_1 *and* p_2 *of* Q_1 *to nodes on the same path from the root of* T_2. □

Clearly, the class of PTPQs encompasses TPQs: every TPQ can be represented by a PTPQ that returns the same answer on every database. A PTPQ Q' corresponding to a TPQ Q can be constructed by adding appropriately to the TPQ node sharing expressions for every node that belongs to two paths of Q. Such a construction is straightforward, and we omit the details for brevity.

Notice that the PTPQ language allows the formulation of queries with no structure at all by specifying a single node per PP and no node sharing expressions. This resembles a flat keyword-based query. On the other side, the PTPQ language also allows the formulation of queries that are completely structured trees by specifying only child relationships and node sharing expressions. Between the two extremes, there are PTPQs that provide some description of the structure without fully specifying a tree.

3.3 PTPQ Containment (Absolute) and PTPQ Containment in the Presence of Dimension Graphs (Relative)

We define in this section PTPQ containment and relative PTPQ containment.

Definition 3.3.1. *Let* Q_1 *and* Q_2 *be two PTPQs.* Q_1 contains Q_2 *(denoted* $Q_2 \subseteq Q_1$*) if and only if for every database* D*, the answer of* Q_2 *on* D *is a subset of the answer of* Q_1 *on* D*. PTPQs* Q_1 *and* Q_2 *on* $\mathcal{D}$ *are* equivalent *(denoted* $Q_2 \equiv Q_1$*) if and only if* $Q_1 \subseteq Q_2$ *and* $Q_2 \subseteq Q_1$. □

If PTPQs are evaluated on databases that underlie a specific dimension graph we can define containment with respect to dimension graph (*relative containment*).

Even though dimension graphs are not schemas in the sense of e.g. a DTD of an XML document, we use them as schemas in processing and evaluating queries. Therefore, we define PTPQ containment and equivalence with respect to a dimension graph.

Definition 3.3.2. *Let Q_1 and Q_2 be two PTPQs on $\mathcal{D}$, and $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. PTPQ Q_1* contains *PTPQ Q_2* with respect to dimension graph $\mathcal{G}$ *(denoted $Q_2 \subseteq_{\mathcal{G}} Q_1$) if and only if for every database T over $\mathcal{D}$ underlying $\mathcal{G}$, every root-to-leaf path in the answer of Q_2 on T is also a root-to-leaf path in the answer of Q_1 on T. PTPQs Q_1 and Q_2 on $\mathcal{D}$ are* equivalent with respect to dimension graph $\mathcal{G}$ *(denoted $Q_2 \equiv_{\mathcal{G}} Q_1$) if and only if $Q_1 \subseteq_{\mathcal{G}} Q_2$ and $Q_2 \subseteq_{\mathcal{G}} Q_1$.* □

Example 3.3.1. *Consider the queries Q_1, Q_2 and Q_3 shown in Figures 3.7, 3.9 and 3.10 respectively.*

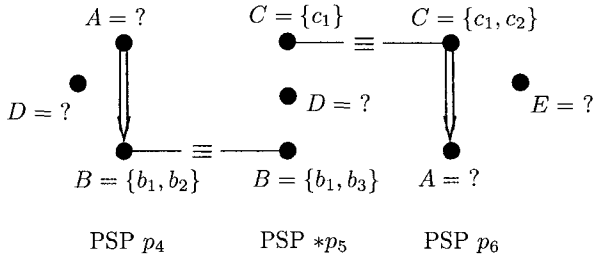

Figure 3.9 PTPQ Q_2.

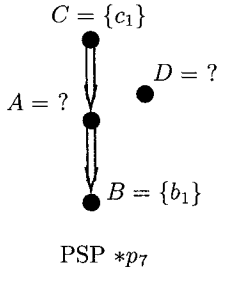

Figure 3.10 PTPQ Q_3.

Consider also the dimension graph $\mathcal{G}$ of Figure 3.4. One can see that $Q_1 \subseteq_{\mathcal{G}} Q_2$, and $Q_2 \subseteq_{\mathcal{G}} Q_3$. Further, $Q_3 \subseteq_{\mathcal{G}} Q_2$, and therefore $Q_2 \equiv_{\mathcal{G}} Q_3$. In contrast, $Q_2 \not\subseteq_{\mathcal{G}} Q_1$. One can also see that $Q_2 \subseteq Q_3$. We will prove these claims in Section 4.2. □

3.4 Structural Expression Inference and PTPQ Full Form

Because tree patterns are partially specified in the PTPQs, new, non-trivial expressions (structural expressions but also annotating expressions) can be inferred from

those explicitly specified in them. These expressions are preserved by all the embeddings of the PTPQ to a database; in other words, adding these expressions to the PTPQ does not remove paths from its answer on any database. We formalize below the notion of structural expression implication.

Definition 3.4.1. *Let $\mathcal{E}$ be the set of expressions of a PTPQ Q on a dimension set $\mathcal{D}$, and e be an expression. We say that e is* implied *from $\mathcal{E}$ (denoted $\mathcal{E} \models e$) if and only if for every database T over $\mathcal{D}$ and every embedding M of Q into T, M preserves e.* □

Example 3.4.1. *Consider the set of structural expressions $\mathcal{E} = \{A[p_1] \Rightarrow B[p_1],\ A[p_1] \equiv A[p_2],\ R[p_2] \Rightarrow B[p_2]\}$. One can see that $\mathcal{E}$ implies the precedence relationship $A[p_2] \Rightarrow B[p_2]$.* □

The *closure* of a set $\mathcal{E}$ of expressions is the set that includes the expressions in $\mathcal{E}$ and those structural expressions that can be implied from $\mathcal{E}$. In order to check queries for containment, we introduce a "normal form" for queries called full form. A PTPQ is in *full form* if its set of expressions $\mathcal{E}$ is closed under implication (that is, $\mathcal{E}$ equals the closure of $\mathcal{E}$). Note that we ignore in $\mathcal{E}$ an annotating expression $D[p] = V$, if a more restrictive annotating expression $D[p] = V'$, with $V' \subseteq V$, also belongs to $\mathcal{E}$. Clearly, a PTPQ can be equivalently put in full form by replacing its set $\mathcal{E}$ of expressions by the closure of $\mathcal{E}$.

To graphically represent queries in full form, we follow the following convention: (a) double arrows (ancestor precedence relationships) from R are not depicted, (b) a double arrow between two dimensions in a PP is not depicted if it can be transitively derived from other double arrows in the same PP, and (c) a double arrow from dimension D_1 to dimension D_2 in a PP is not depicted if there is a single arrow from D_1 to D_2 in the same PP. All the omitted double arrows and node sharing expressions can be trivially derived from the expressions explicitly represented in the PTPQ graph.

Example 3.4.2. *Consider the PTPQs Q_1 and Q_2 of Figures 3.7 and 3.9. Figures 3.11 and 3.12 show the full form of Q_1 and Q_2 respectively. PTPQ Q_3 of Figure 3.10 is in full form.* □

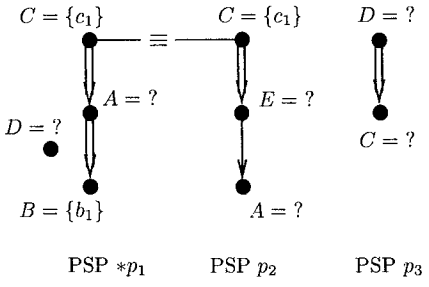

Figure 3.11 Full form of PTPQ Q_1.

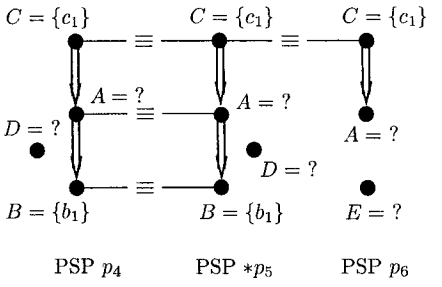

Figure 3.12 Full form of PTPQ Q_2.

A set of inference rules for structural expression implication has been provided in [48] and the complete list is presented below. The inference rules were derived in collaboration with S. Souldatos from the National Technical University of Athens.

(IR1) $\vdash r[p_1] \equiv r[p_2]$
(IR2) $a[p_1] \equiv a[p_2],\ a[p_2] \equiv a[p_3] \vdash a[p_1] \equiv a[p_3]$
(IR3) *a structural expression that involves* $a[p] \vdash r[p] \Rightarrow a[p]$
(IR4) $a[p] \rightarrow b[p] \vdash a[p] \Rightarrow b[p]$
(IR5) $a[p] \Rightarrow b[p],\ b[p] \Rightarrow c[p] \vdash a[p] \Rightarrow c[p]$
(IR6) $a[p] \rightarrow b[p],\ a[p] \Rightarrow c[p] \vdash b[p] \Rightarrow c[p]$
(IR7) $a[p] \rightarrow b[p],\ c[p] \Rightarrow b[p] \vdash c[p] \Rightarrow a[p]$
(IR8) $a[p_1] \rightarrow b[p_1],\ b[p_1] \equiv b[p_2] \vdash a[p_2] \rightarrow b[p_2]$
(IR9) $a[p_1] \Rightarrow b[p_1],\ b[p_1] \equiv b[p_2] \vdash a[p_2] \Rightarrow b[p_2]$
(IR10) $a[p_1] \Rightarrow b[p_1],\ a[p_1] \equiv a[p_2],\ r[p_2] \Rightarrow b[p_2] \vdash a[p_2] \Rightarrow b[p_2]$
(IR11) $a[p_1] \Rightarrow b[p_1],\ b[p_1] \equiv b[p_2] \vdash a[p_1] \equiv a[p_2]$
(IR12) $a[p_1] \rightarrow b[p_1],\ c[p_2] \rightarrow b[p_2],\ d[p_1] \equiv d[p_2] \vdash d[p_1] \Rightarrow a[p_1]$
(IR13) $a[p_1] \rightarrow b[p_1],\ a[p_2] \rightarrow c[p_2],\ d[p_1] \equiv d[p_2] \vdash d[p_1] \Rightarrow a[p_1]$
(IR14) $a[p_1] \Rightarrow b[p_1],\ b[p_2] \Rightarrow a[p_2],\ c[p_1] \equiv c[p_2] \vdash c[p_1] \Rightarrow a[p_1]$
(IR15) $c[p] \Rightarrow b[p],\ b[p_2] \Rightarrow c[p_2],\ a[p] \equiv a[p_1],\ b[p_1] \equiv b[p_2] \vdash a[p_1] \Rightarrow b[p_1]$
(IR16) $a[p] \Rightarrow c[p],\ b[p_2] \rightarrow c[p_2],\ a[p] \equiv a[p_1],\ b[p_1] \equiv b[p_2] \vdash a[p_1] \Rightarrow b[p_1]$
(IR17) $a[p] \equiv a[p_1],\ b[p_1] \equiv b[p_2],\ c[p] \Rightarrow d[p],\ d[p_2] \Rightarrow c[p_2] \vdash a[p] \Rightarrow c[p]$

Figure 3.13 A set of inference rules.

Let a, b, c and d be distinct dimensions and p, p_1, and p_2 be distinct PPs. We use the symbol $\vdash$ to denote that the expressions that precede it produce the expression

that follows it. The absence of expressions that precede $\vdash$ denotes an axiom. Figure 3.13 shows a set of inference rules.

Each inference rule derives a new structural expression from a set of structural expressions. Clearly, the number of structural expressions that can be derived using the inference rules in a PTPQ is bound by $O(n^2)$, where n is the product of the distinct dimensions in the PTPQ and its number of PPs. Therefore, the full form of a PTPQ can be computed in polynomial time.

3.5 PTPQ Satisfiability

We also need the concept of satisfiable PTPQ.

Definition 3.5.1. *A PTPQ is* satisfiable *if and only if it has a non-empty answer on some database.* □

One can see that adding the precedence relationship $b[p_1] \Rightarrow c[p_1]$ to the PP p_1 of PTPQ Q_1 of Figure 3.7, results in an unsatisfiable PTPQ. Indeed, $b[p_1] \Rightarrow c[p_1]$ contradicts the descendant precedence relationship $c[p_1] \Rightarrow b[p_1]$ that can be derived from the set of structural expressions of Q_1 (see the full form of Q_1 in Figure 3.11). This condition is necessary as the next proposition shows [50].

Proposition 3.5.1. *A PTPQ is unsatisfiable iff two contradicting descendant precedence relationships $a[p] \Rightarrow b[p]$ and $b[p] \Rightarrow a[p]$ (in the same PP p) appear in its full form.* □

Detecting an unsatisfiable PTPQ avoids evaluating the PTPQ to compute an empty answer. The overhead for this check amounts to computing the full form. An unsatisfiable PTPQ is contained in any PTPQ. In the following, unless otherwise specified, we assume that PTPQs are satisfiable.

Similarly to PTPQ containment, we define PTPQ unsatisfiability in the presence of a dimension graph.

Definition 3.5.2. *Let $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. A PTPQ on $\mathcal{D}$ is* unsatisfiable with respect to $\mathcal{G}$ *if its answer is empty on every database underlying $\mathcal{G}$. Otherwise, it is called* satisfiable with respect to $\mathcal{G}$. □

An unsatisfiable PTPQ with respect to dimension graph $\mathcal{G}$ is contained in any PTPQ with respect to $\mathcal{G}$. In the following, when dimension graph $\mathcal{G}$ is present and it is not specified otherwise, we assume that PTPQs are satisfiable with respect to dimension graph $\mathcal{G}$.

CHAPTER 4

CHECKING PTPQ CONTAINMENT IN THE PRESENCE OF DIMENSION GRAPHS

In this chapter we study PTPQ containment in the presence of dimension graphs. Then we provide heuristic approaches for checking PTPQ containment. Finally, we present and analyze experimental results for these techniques.

4.1 Valid Partial Paths Clusters

A set of PPs in a PTPQ that are all linked together through node sharing expressions is called cluster:

Definition 4.1.1. *A* cluster *is a set C of PPs and node sharing expressions such that for every partition of C in two non-empty sets there is a node sharing expression in Q on an element different than R involving PPs from both sets (that is, the cluster does not comprise disconnected sets of PPs).* □

We represent clusters as PTPQs without an output PP (see, for instance, Figures 4.1 and 4.2). Given a dimension graph $\mathcal{G}$, it is possible that there is a cluster that can be added to any PTPQ without affecting its answer on any database that underlies $\mathcal{G}$. To deal with this issue, we introduce the concept of valid cluster.

Definition 4.1.2. *Let $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. A cluster Q is* valid *with respect to $\mathcal{G}$ if and only if, for every database T over $\mathcal{D}$ underlying $\mathcal{G}$, there is a mapping M of the annotated dimensions of Q to nodes of T that satisfies the conditions (a), (b), (c) and (d) of definition 3.2.2 (i.e., M is an embedding of Q into T).* □

Example 4.1.1. *Consider the dimension graph $\mathcal{G}$ of Figure 3.4. Let C_1 be the cluster that consists of a single PP comprising a single dimension A annotated with a '?'. Since A appears in $\mathcal{G}$, C_1 is valid. Let also C_2 be the cluster that consists of a single PP p_2 comprising two dimensions C and E annotated with a "?" and a child precedence relationship $C \rightarrow E$. Since there is an edge (C, E) in $\mathcal{G}$, it is not difficult to see that C_2 is valid with respect to $\mathcal{G}$.*

Valid clusters can involve several PPs. Consider the clusters C_3 and C_4 shown in Figures 4.1 and 4.2. As it will become clear below, these clusters are valid with respect to $\mathcal{G}$. □

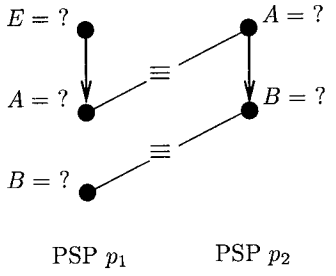

Figure 4.1 Cluster C_3 (valid).

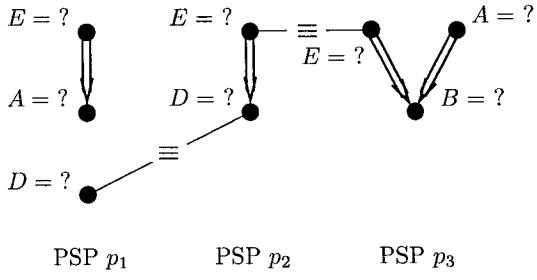

Figure 4.2 Cluster C_4 (valid).

Clearly, adding to a PTPQ Q a valid cluster or removing from a PTPQ a valid cluster that does not include the output PP of Q and does not share nodes with paths outside the cluster results in a PTPQ equivalent to Q. One can see that the only way for a cluster to be valid is that the embedding of Definition 4.1.2 maps every PP in the cluster to the *same* path in the database T. The following proposition exploits this observation to provide necessary and sufficient conditions for a cluster to be valid with respect to a dimension graph.

Proposition 4.1.1. *Let $\mathcal{G}$ be a dimension graph on $\mathcal{D}$ and C be a cluster on $\mathcal{D}$. Cluster C is* valid *with respect to $\mathcal{G}$ if and only if the following conditions hold:*

(a) Every dimension in C is annotated with a "?" (or with the set of all the values of the dimension), and

(b) There is an edge in $\mathcal{G}$ such that for every path p from the root of $\mathcal{G}$ that comprises this edge there is a mapping from the annotated dimensions in C to the dimensions of p that preserves the dimensions and all the precedence relationships in C. □

Proof: (*If part*) Let's assume that cluster C is valid and condition (a) does not hold. Then, there is a dimension D in C, which is not annotated with a "?" (or with the set of all the values of the dimension). This means there is $d \in D$ that is not in the annotation of D in at least one PP p of C. We construct a database T underlying $\mathcal{G}$ such that d is the only value of dimension D in T. Clearly, PP p does not have an embedding to T. This contradicts our assumption that cluster C is valid.

Let's now assume that cluster C is valid and condition (b) does not hold. Then, there is no edge such that for every path p from the root of $\mathcal{G}$ that comprises this edge there is a mapping from C to p preserving dimensions and precedence relationships. We construct a tree T as follows: Start with an empty database T. For each edge e in dimension graph $\mathcal{G}$ find a path p from the root of $\mathcal{G}$ containing e such that there is no embedding from cluster C into p. For each path p add a root-to-leaf path to T constructed from p by replacing labeling dimension by one of their values. These paths in T have a single common node labeled by r. Clearly, T is a database underlying graph $\mathcal{G}$. Further, by construction, there is no embedding of C into T that maps all PPs of C into the same path of T. Since the paths of T do not share nodes (other than the root node) and all the PPs of C are involved in node sharing expressions, there is no embedding of C into T. This contradicts our assumption that cluster C is valid.

(*Only if part*) Let's assume that conditions (a) and (b) hold, and let T be a database underlying $\mathcal{G}$. Consider an edge $e = (D_i, D_j)$ in $\mathcal{G}$ satisfying condition (b). By Definition 3.1.2, in every database T underlying $\mathcal{G}$, there are nodes n_i and n_j in T labeled by values $v_i \in D_i$ and $v_j \in D_j$, respectively, such that n_j is a child of n_i. Let $p' = r, n_1, \ldots, n_k, n_i, n_j$ be a path in T where $n_l \in D_l$, l in $[1, k] \cup \{i, j\}$, and p be the path $R, D_1, \ldots, D_k, D_i, D_j$ in $\mathcal{G}$. Since p contains e, there is a mapping from C to p that preserves dimensions and precedence relationships in C. Therefore, there is a mapping m from C to p' that preserves dimensions and precedence relationships. Since condition (a) holds, all the dimensions in C are annotated by "?" and therefore m is an embedding of C into p'. Thus, C can be embedded to any database underlying $\mathcal{G}$, that is, it is valid wrt $\mathcal{G}$. □

Example 4.1.2. *Consider the cluster C_3 of Example 4.1.1 shown in Figure 4.1. There are two paths from the root of $\mathcal{G}$ that comprise edge (A, B). Each path involves all the annotated dimensions in C_3 and satisfies the precedence relationships of both PPs of C_3. Therefore, cluster C_3 is valid. Consider also the cluster C_4 of Example 4.1.1 shown in Figure 4.2. As before, we can show that there are exactly two paths from the root of $\mathcal{G}$ that comprise edge (D, B) and each of them involves all the annotated dimensions in C_4 and satisfies the precedence relationships of all three PPs of C_4. Therefore, C_4 is also valid.* □

Checking for valid clusters can be performed efficiently as the next proposition shows.

Proposition 4.1.2. *Let $\mathcal{G}$ be a dimension graph on $\mathcal{D}$, and C be a cluster on $\mathcal{D}$. Let also n be the product of the number of dimensions in $\mathcal{G}$ and the number of PPs in C. Checking if C is valid with respect to $\mathcal{G}$ can be done in polynomial time on n.* □

Proof: (Sketch) Based on Proposition 4.1.1, checking if C is valid can be performed as follows: (a) for every edge (X, Y) in $\mathcal{G}$, compute the set of precedence relationships

that hold on every path from the root of $\mathcal{G}$ that comprises (X, Y), and (b) check if some of these sets contains all the precedence relationships of C. If this is the case, C is valid with respect to $\mathcal{G}$.

The set S of descendant precedence relationships that hold on every path from the root of $\mathcal{G}$ that comprises (X, Y) can be computed as follows:

(a) Initially let $S = \{X \Rightarrow Y\}$.

(b) Remove every outgoing edge from Y in $\mathcal{G}$ to create a new graph $\mathcal{G}'$.

(c) For every node $Z, Z \neq X, Z \neq Y$ of $\mathcal{G}'$, remove all the outgoing edges from Z, and check if there is no path from the root of $\mathcal{G}'$ to X. If this is the case, add $Z \Rightarrow X$ to S.

(d) For every precedence relationship $Z \Rightarrow X$ added to S in step (c), apply recursively step (c) to node Z.

The set S' of child precedence relationships that hold on every path from the root of $\mathcal{G}$ that comprises (X, Y) can be computed from S as follows: initially let $S' = \{X \rightarrow Y\}$. For every $V \Rightarrow W \in S$, if $V \rightarrow W$ appears in $\mathcal{G}'$, remove it and check if there is no path from the root of $\mathcal{G}'$ to X. If this is the case, add $V \rightarrow W$ to S'.

Since there are at most m^2 edges in $\mathcal{G}$, where m is the number of nodes of $\mathcal{G}$, and checking the existence of a path between the root of $\mathcal{G}$ and a node can be done in $O(m^2)$, the previous process can be done in polynomial time on m. Since the number of precedence relationships in C is $O(n)$, where n is the product of the number of nodes in $\mathcal{G}$ and the number of PPs in C, checking the validity of C can be done in polynomial time on n. □

In the following, we assume that a PTPQ does not comprise a valid disconnected cluster that does not contain the output PP of the PTPQ.

4.2 Checking PTPQ Containment With Respect to a Dimension Graph

In order to address PTPQ containment with respect to a dimension graph, we need the concept of homomorphism between partially specified tree-pattern PTPQs.

Definition 4.2.1. *Let Q_1 and Q_2 be two PTPQs on $\mathcal{D}$. A* homomorphism *from Q_2 to Q_1 is a mapping h from the annotated dimensions of Q_2 to the annotated dimensions of Q_1 such that:*

(a) If the annotated dimension n is labeled by a dimension D in Q_2, then $h(n)$ is also labeled by D in Q_1.

(b) All the annotated dimensions of a PP in Q_2 are mapped under h to annotated dimensions in the same PP of Q_1.

(c) If an annotated dimension n in Q_2 is annotated by $V_2 \neq ?$, then $h(n)$ in Q_1 is annotated by V_1 such that $V_1 \subseteq V_2$.

(d) The annotated dimensions in the output PP o_2 of Q_2 are mapped under h to annotated dimensions in the output PP o_1 of Q_1, and every annotated dimension in o_1 is the image under h of an annotated dimension in o_2.

(e) If $D[p] \rightarrow D'[p]$ (resp. $D[p] \Rightarrow D'[p]$) is in Q_2, then $h(D[p]) \rightarrow h(D'[p])$ (resp. $h(D[p]) \Rightarrow h(D'[p])$) is in Q_1.

(f) If $D[p] \equiv D[p']$ is in Q_2, then $h(D[p])$ and $h(D[p'])$ coincide or $h(D[p]) \equiv h(D[p'])$ is in Q_1. □

The existence of a homomorphism between PTPQs is a sufficient condition for PTPQ containment with respect to a dimension graph as the next proposition shows.

Proposition 4.2.1. *Let Q_1 and Q_2 be two PTPQs on $\mathcal{D}$, where Q_1 is in full form, and $\mathcal{G}$ be dimension graph. If there is a homomorphism from Q_2 to Q_1, then $Q_1 \subseteq_{\mathcal{G}} Q_2$.* □

Proof: Let M be an embedding of Q_1 into a database T underlying $\mathcal{G}$, and h be a

homomorphism from Q_2 to Q_1. Clearly, the composition of M on h, $M \circ h$, is an embedding of Q_2 into T. Therefore, $Q_1 \subseteq_{\mathcal{G}} Q_2$. □

Example 4.2.1. *Let Q_1' be the PTPQ shown in Figure 3.11. This is the full form of PTPQ Q_1 of Figure 3.7. Consider also PTPQ Q_2 shown in Figure 3.9. One can see that there is a homomorphism from Q_2 to Q_1'. Therefore, $Q_1 \subseteq_{\mathcal{G}} Q_2$.* □

Unfortunately, the existence of a homomorphism is not a necessary condition for PTPQ containment with respect to a dimension graph.

Example 4.2.2. *Consider, PTPQs Q_2 and Q_3 of Figures 3.9 and 3.10, respectively. PTPQ Q_3 is in full form. As mentioned in Example 3.3.1, $Q_3 \subseteq_{\mathcal{G}} Q_2$. However, there is no homomorphism from Q_2 to Q_3 since dimension E of Q_2 does not appear in Q_3.* □

In order to fully characterize PTPQ containment with respect to a dimension graph, we use the concept of dimension tree of a PTPQ on a dimension graph.

Definition 4.2.2. *Let Q be a PTPQ on a dimension set $\mathcal{D}$, and $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. Let also Q' be the full form of Q. A* dimension tree *of Q on $\mathcal{G}$ is a tree U such that:*

(a) The nodes of U are labeled by dimensions in $\mathcal{D}$ and their annotations (annotating expressions). No two nodes on a path of U are labeled by the same dimension.

(b) One of the nodes of U is marked. This node is called output node *of U, and the path from the root to the output node of U is called* output path *of U.*

(c) There is a mapping m from the set of the nodes of U to the set of nodes of $\mathcal{G}$ such that:

(c1) If n is a node of U, n and $m(n)$ are labeled by the same dimension.

(c2) If (n_1, n_2) is an edge in U, $(m(n_1), m(n_2))$ is an edge of $\mathcal{G}$.

(d) There is a mapping m' from the set of the annotated dimensions of Q' to the set of nodes of U such that:

(d1) The annotated dimensions of a PP in Q' are mapped under m' to nodes on the same path of U.

(d2) The marked node of U is the image under m' of an annotated dimension of the output PP o of Q' that is the descendant in U of the images under m' of all the other annotated dimensions of o.

(d3) For every precedence relationship $A[p] \rightarrow B[p]$ (resp. $A[p] \Rightarrow B[p]$) in Q', $m'(B[p])$ is a child (resp. descendant) of $m'(A[p])$ in U.

(d4) If a node sharing expression $D[p_1] \equiv D[p_2]$ is in Q', $m'(D[p_1]) = m'(D[p_2])$.

(d5) A dimension D of Q' annotated by V is mapped by m' to a node n labeled by D and annotated by V.

(d6) Every leaf node in U is the image under m' of an annotated dimension of Q'.

(d7) For a dimension D in PPs p_1 and p_2 in Q, $m'(D[p_1]) \neq m'(D[p_2])$ unless $D[p_1] \equiv D[p_2]$ is in Q'. □

Intuitively, a dimension tree for Q on $\mathcal{G}$ represents a mapping of Q into $\mathcal{G}$ that respects PPs, labeling dimensions, precedence relationships, and node sharing expressions. This mapping is the composition $m \circ m'$ of m' and m. The dimensions trees of Q on $\mathcal{G}$ represent all such possible mappings of Q into $\mathcal{G}$.

Example 4.2.3. *Consider PTPQ Q_1 of Figure 3.7 on the dimension graph $\mathcal{G}$ of Figure 3.4. Figure 4.3 shows the dimension trees of Q_1 on $\mathcal{G}$. For simplicity of presentation, dimension annotations that are '?' are not shown in the graphical representation of dimension trees.* □

A dimension tree of a PTPQ on a dimension graph can be seen as a PTPQ where the tree structure is completely specified: root-to-leaf paths determine PPs;

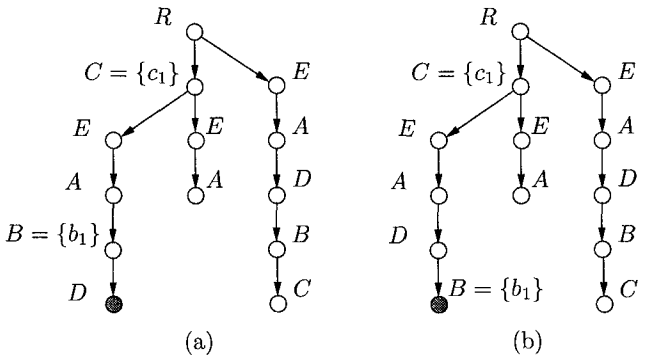

Figure 4.3 The dimension trees of Q_1 on $\mathcal{G}$: (a) U_1^1, (b) U_1^2.

the output path determines the output PP; edges determine child precedence relationships; common nodes of two paths determine node sharing expressions. Such PTPQs form a tree pattern without missing edges involving only parent-child (and not ancestor-descendant) relationships. Since dimension trees are special cases of PTPQs, we can apply to them the concepts defined on PTPQs: answer of a PTPQ, and homomorphism between PTPQs.

Given a dimension graph $\mathcal{G}$, a PTPQ Q is associated to the set $\mathcal{U}$ of its dimension trees on $\mathcal{G}$. Every path in the answer of Q on a database T underlying $\mathcal{G}$ is also a path in the answer of some $U \in \mathcal{U}$ on T, and conversely. Therefore, the answer of Q on T can be constructed by merging into a single database the answers of the dimension trees of $\mathcal{U}$ on T. The identities of the nodes are used to perform this merging.

We now state a theorem that provides necessary and sufficient conditions for relative PTPQ containment, in terms of homomorphisms between dimension trees.

Theorem 4.2.1. *Let Q_1 and Q_2 be two PTPQs on $\mathcal{D}$ and $\mathcal{G}$ be a dimension graph on $\mathcal{D}$. Let also $\mathcal{U}_1$ and $\mathcal{U}_2$ be the sets of dimension trees of Q_1 and Q_2, respectively, on $\mathcal{G}$. $Q_1 \subseteq_{\mathcal{G}} Q_2$ if and only if there is a mapping f from $\mathcal{U}_1$ to $\mathcal{U}_2$ such that, for every dimension tree U in $\mathcal{U}_1$, there is a homomorphism from $f(U)$ to U.* □

Proof: We start by the *if part.* Let M be an embedding of a dimension tree $U \in \mathcal{U}_1$

into a database T that underlies $\mathcal{G}$. Let h be an homomorphism from the dimension tree $f(U) \in \mathcal{U}_2$. The composition of M on h, $M \circ h$, is an embedding of $f(U)$ on T. Therefore, $Q_1 \subseteq_{\mathcal{G}} Q_2$.

For the *only if part*, let's assume that $Q_1 \subseteq_{\mathcal{G}} Q_2$ and there is no such mapping f from $\mathcal{U}_1$ to $\mathcal{U}_2$. Then, there exists a dimension tree U_1 in $\mathcal{U}_1$ such that there is no homomorphism from any dimension tree in $\mathcal{U}_2$ to U_1. We construct a database T by replacing in U_1 each labeling dimension by one of its annotating values. Clearly, T underlies $\mathcal{G}$. Since there is no homomorphism from a dimension tree U_2 in $\mathcal{U}_2$ to U_1, no U_2 can be embedded into T. Therefore, Q_1 has an answer on T while the answer of Q_2 on T is empty. This contradicts our assumption that $Q_1 \subseteq_{\mathcal{G}} Q_2$. □

Example 4.2.4. *Consider the PTPQs Q_1, Q_2 and Q_3 of Example 4.2.3, shown in Figures 3.7, 3.9, and 3.10, and the dimension graph $\mathcal{G}$ of Figure 3.4. Figures 4.3, 4.4, and 4.5 show the dimension trees of Q_1, Q_2, and Q_3 on $\mathcal{G}$, respectively.*

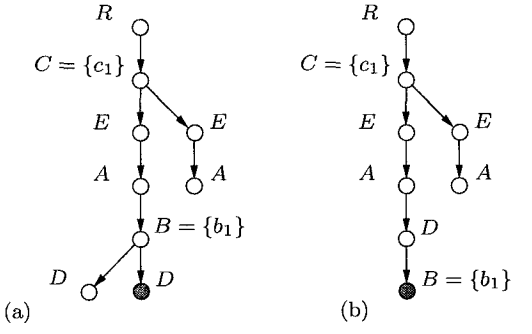

Figure 4.4 The dimension trees of Q_2 on $\mathcal{G}$: (a) U_2^1, (b) U_2^2.

Theorem 4.2.1 proves the claim of Example 4.2.3 that $Q_1 \subseteq_{\mathcal{G}} Q_2$: let f be the mapping $f(U_1^1) = U_2^1$ and $f(U_1^2) = U_2^2$. Clearly, there is a homomorphism h from the nodes of U_2^1 to U_1^1 and from U_2^2 to U_1^2. Based on Theorem 4.2.1, we can also show that $Q_2 \subseteq_{\mathcal{G}} Q_3$ and $Q_3 \subseteq_{\mathcal{G}} Q_2$. Theorem 4.2.1 also proves that, in contrast, $Q_2 \not\subseteq_{\mathcal{G}} Q_1$. □

One can see that there can be a number of dimension trees for a given PTPQ Q and dimension graph $\mathcal{G}$ that is exponential on the number of nodes of $\mathcal{G}$. However,

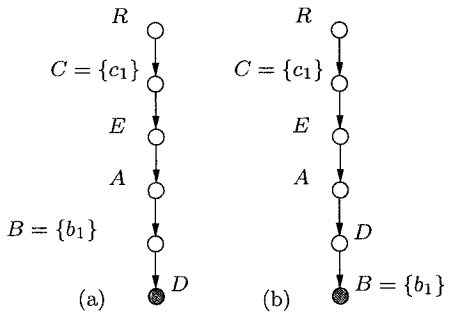

Figure 4.5 The dimension trees of Q_3 on $\mathcal{G}$: (a) U_3^1, (b) U_3^2.

if the dimension graph is a tree, PTPQ Q has a single dimension tree with respect to $\mathcal{G}$.

4.3 Heuristic Approaches for PTPQ Containment with Respect to a Dimension Graph

Checking PTPQ containment with respect to a dimension graph can be time consuming since, as we saw in the previous section, it involves checking the existence of homomorphisms between pairs of dimension trees (Theorem 4.2.1). As mentioned in the previous section, the number of these pairs can be very large. Therefore, we cannot rely on Theorem 4.2.1 for checking efficiently containment of partial tree-pattern PTPQs.

In this section, we suggest a heuristic approach for checking two PTPQs for containment with respect to a dimension graph $\mathcal{G}$ that reduces to checking the existence of a homomorphism only between two PTPQs.

4.3.1 The Basic Idea

Suppose that we want to check if PTPQ Q is contained in PTPQ Q' with respect to $\mathcal{G}$. If there is a homomorphism from Q' to the full form of Q, by Proposition 4.2.1, we deduce that $Q \subseteq_{\mathcal{G}} Q'$. However, if such a homomorphism does not exist, Q might

or might not be contained in Q'. As an example, consider PTPQs Q_2 and Q_3 shown in Figures 3.9 and 3.10. PTPQ Q_3 is in full form. Consider also the dimension graph $\mathcal{G}$ of Figure 3.4. As mentioned in Example 4.2.2, there is no homomorphism from Q_2 to Q_3. Therefore, we cannot, based on this fact, decide on the containment of Q_3 in Q_2 with respect to $\mathcal{G}$. Observe now that the following statement holds on $\mathcal{G}$: if a path from the root of $\mathcal{G}$ contains dimension A (that is, if it satisfies the precedence relationship $R \Rightarrow A$), it also satisfies the precedence relationship $E \Rightarrow A$. We call such a statement "rule instance" and the precedence relationship $E \Rightarrow A$ is qualified as *extracted*. The PP p_7 of PTPQ Q_3 (which is in full form) contains the dimension A. Therefore, if we add the extracted precedence relationship $E \Rightarrow A$ to p_7 we obtain a PTPQ which is equivalent to Q_3 with respect to $\mathcal{G}$. Similarly, one can see that the following rule instance holds on $\mathcal{G}$: if a path from the root of $\mathcal{G}$ satisfies the precedence relationship $R \Rightarrow D$, it also satisfies the precedence relationships $E \Rightarrow D$ and $A \Rightarrow D$. Therefore, we can again add to p_7 the extracted precedence relationships $E \Rightarrow D$ and $A \Rightarrow D$ to obtain a PTPQ equivalent to Q_3 with respect to $\mathcal{G}$. Taking the full form of the resulting PTPQ Q'_3, we obtain the PTPQ shown in Figure 4.6. PTPQ Q'_3 has

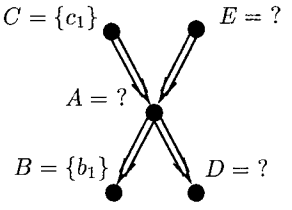

Figure 4.6 PTPQ Q'_3.

more precedence relationships than Q_3. It is not difficult to see now that there is a homomorphism from Q_2 to Q'_3. Since Q'_3 is equivalent to Q_2 with respect to $\mathcal{G}$, we can deduce that $Q_3 \subseteq_{\mathcal{G}} Q_2$.

Therefore, the basic idea is to extract precedence relationships from the di-

mension graph $\mathcal{G}$ that can be iteratively added to PTPQ Q to produce appropriately an equivalent PTPQ with respect to $\mathcal{G}$ (called augmented form of Q). The possibility for the existence of a homomorphism from PTPQ Q' to the augmented form of PTPQ Q is increased. If such a homomorphism exists we can deduce that $Q \subseteq_{\mathcal{G}} Q'$.

There are two ways to implement the heuristic approach. The first one (called *precomputation* heuristic approach) considers a rule instance pattern (called *rule*). It computes in advance and stores all the rule instances of this rule that hold on $\mathcal{G}$. When a PTPQ Q emerges, the extracted precedence relationships of these rule instances are used to compute the augmented form of Q. Using multiple rules instead of one provides a more refined characterization of $\mathcal{G}$, and increases the accuracy (completeness) of the approach. The possibility of missing the detection of a PTPQ containment case with respect to $\mathcal{G}$ is reduced. However, this gain in accuracy is obtained at the expense of the space required to store the rule instances that hold on $\mathcal{G}$, and the overall time required to compute the augmented form of Q. Therefore, a trade-off should be determined between the desired accuracy and the space and time cost incurred by the multiple rules considered by the precomputation heuristic approach.

The second way to implement the heuristic approach (called *on-the-fly* heuristic approach) considers all the precedence relationships in a PP of PTPQ Q in order to extract from $\mathcal{G}$ the precedence relationships that are used for computing the augmented form of Q. Therefore, the precedence relationships are extracted from the dimension graph at PTPQ time. We formally define below both heuristic approaches.

4.3.2 Precomputation Heuristic Approach

We first introduce the concept of precedence relationship extraction rule.

Precedence relationship extraction rules

Definition 4.3.1. *A* (precedence relationship extraction) rule *is an expression of the form* $\mathcal{P} \Longrightarrow \mathcal{C}$, *where* $\mathcal{P}$ *and* $\mathcal{C}$ *are non-empty sets of precedence relationship types. A* precedence relationship type *is a precedence relationship that involves dimension variables (instead of dimensions), and (possibly) dimension* R. □

For example, $\{R \Rightarrow X\} \Longrightarrow \{Y \Rightarrow X\}$ is a rule, where X and Y are dimension variables.

Definition 4.3.2. *An* instance *of a rule* $\mathcal{P} \Longrightarrow \mathcal{C}$ *is an expression of the form* $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ *obtained as follows: let* α_I *be an assignment of dimensions to the dimension variables occurring in* $\mathcal{P}$.

(a) $\mathcal{P}_I$, *the* premise, *is the set of precedence relationships obtained by assigning distinct dimensions to all the dimension variables in* $\mathcal{P}$ *according to* α_I, *and*

(b) $\mathcal{C}_I$, *the* conclusion, *is a set of precedence relationships where each of them is obtained from a precedence relationship type in* $\mathcal{C}$ *by replacing in it: (i) every variable* X *that occurs also in* $\mathcal{P}$ *by* $\alpha_I(X)$, *and (ii) every variable* Y *that does not occur in* $\mathcal{P}$ *by some dimension.* □

Note that as a consequence of the previous definition, a precedence relationship type in the conclusion of a rule might contribute multiple precedence relationships to the conclusion of an instance of this rule obtained by replacing a dimension variable that does not appear in the premise of the rule by multiple dimensions. It is also possible that a precedence relationship type in the conclusion of a rule does not contribute any precedence relationships to the conclusion of an instance of this rule.

Consider, for instance, the dimension graph $\mathcal{G}$ of Figure 3.4. An instance of the rule $\{R \Rightarrow X\} \Longrightarrow \{Y \rightarrow X, Y \Rightarrow X\}$ is $\{R \Rightarrow D\} \Longrightarrow \{R \Rightarrow D, A \Rightarrow D, E \Rightarrow D\}$. Intuitively, the premise of this instance characterizes all the paths from the root of $\mathcal{G}$ that involve dimension D. The conclusion comprises descendant precedence relationships to D.

Definition 4.3.3. *A rule instance* $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ holds *on a dimension graph* $\mathcal{G}$, *if the precedence relationships in* $\mathcal{C}_I$ *are satisfied by every path from the root of* $\mathcal{G}$ *that satisfies the precedence relationships in* $\mathcal{P}_I$, *and there is no rule instance* $\mathcal{P}_I \Longrightarrow \mathcal{C'}_I$ *with the same property such that* $\mathcal{C}_I \subset \mathcal{C'}_I$. □

Example 4.3.1. *Consider the rule* $\{R \Rightarrow X\} \Longrightarrow \{Y \rightarrow X, Y \Rightarrow X\}$. *One can see that the following instances of this rule hold on the dimension graph* $\mathcal{G}$ *of Figure 3.4:*

$\{R \Rightarrow A\} \Longrightarrow \{R \Rightarrow A, E \Rightarrow A, E \rightarrow A\}$,

$\{R \Rightarrow B\} \Longrightarrow \{R \Rightarrow B, A \Rightarrow B, E \Rightarrow B\}$,

$\{R \Rightarrow C\} \Longrightarrow \{R \Rightarrow C\}$,

$\{R \Rightarrow D\} \Longrightarrow \{R \Rightarrow D, A \Rightarrow D, E \Rightarrow D\}$,

$\{R \Rightarrow E\} \Longrightarrow \{R \Rightarrow E\}$.

In the case of dimensions C *and* E, *no new precedence relationships can be extracted from* $\mathcal{G}$ *by the respective rule instances.* □

Adding precedence relationships to PTPQs

Given a PTPQ Q, precedence relationships extracted from a dimension graph $\mathcal{G}$ by rule instances that hold on $\mathcal{G}$ can be appropriately added to Q to create the augmented form of Q.

Definition 4.3.4. *Consider a PTPQ Q, a dimension graph $\mathcal{G}$ and a rule $\mathcal{R}$. The* augmented *form of Q with respect to $\mathcal{G}$, and $\mathcal{R}$, say Q', is constructed from Q as follows:*

Let initially $Q' = Q$.

Repeat the following steps until no more changes can be applied to Q':

- *Put Q' in full form.*
- *For every PP p in Q' and for every instance $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ of $\mathcal{R}$ that holds on $\mathcal{G}$, if the precedence relationships in $\mathcal{P}_I$ appear in p, add to p the precedence relationships in $\mathcal{C}_I$.* □

We can now state the following proposition.

Proposition 4.3.1. *Let Q be a PTPQ, $\mathcal{G}$ be a dimension graph, and $\mathcal{R}$ be a rule. The augmented form of PTPQ Q with respect to $\mathcal{G}$ and $\mathcal{R}$ is a PTPQ equivalent to Q with respect to $\mathcal{G}$.* □

Proof: Let Q' be the augmented form of PTPQ Q with respect to $\mathcal{G}$ and $\mathcal{R}$. In constructing Q', whenever a precedence relationship P is added to a PP p of Q, there is an instance of $\mathcal{R}$, $\mathcal{R}_I$, such that $\mathcal{R}_I$ holds on $\mathcal{G}$, all the precedence relationships in the premise of $\mathcal{R}_I$ appear in p, and P appears in the conclusion of $\mathcal{R}_I$. Therefore, for every embedding of Q to a database T, the image of p satisfies P. Consequently, $Q' \subseteq Q$. Clearly, $Q \subseteq Q'$. Therefore, $Q \equiv Q'$. □

Generating the augmented form of a PTPQ Q involves adding repeatedly to it precedence relationships extracted from the dimension graph and computing the full form of the resulting PTPQ until a fixed point is reached. Computing the full form of a PTPQ possibly adds structural expressions (precedence relationships and node sharing expressions) to it. This process increases the possibility for homomorphisms from other PTPQs to Q to exist.

Example 4.3.2. *Consider PTPQs Q_2 and Q_3 of Figures 3.9 and 3.10 and the dimension graph $\mathcal{G}$ of Figure 3.4. In Example 4.2.2, we showed that there is no homomorphism from Q_2 to Q_3. Consider now the rule $\mathcal{R}$: $\{X \Rightarrow Y\} \Longrightarrow \{U \Rightarrow V\}$. Figure 4.7 shows PTPQ Q'_3, the augmented form of PTPQ Q_3 with respect to $\mathcal{G}$ and $\mathcal{R}$. PTPQ Q_3 and rule $\mathcal{R}$ are simple and therefore, one iteration is enough for computing*

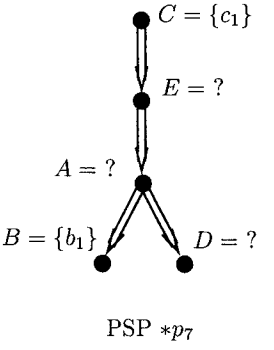

Figure 4.7 PTPQ Q'_3, the augmented PTPQ Q_3 with respect to $\mathcal{G}$ and $\mathcal{R}$.

Q'_3. Observe that Q'_3 has more precedence relationships than Q_3. Clearly, there is a homomorphism from Q_2 to Q'_3. By Proposition 4.3.1, $Q'_3 \equiv_{\mathcal{G}} Q_3$. Then, by Proposition 4.3.1, $Q_3 \subseteq_{\mathcal{G}} Q_2$. This result proves again what we showed in Example 4.2.4 using Theorem 4.2.1. □

The next proposition shows that if the dimension graph is a tree, a simple rule can guarantee total accuracy for the heuristic approach.

Proposition 4.3.2. *Let Q_1 and Q_2 be two PTPQs in full form, $\mathcal{G}$ be a dimension graph which is a tree, and $\mathcal{R}$ be the rule $\{R \Rightarrow X\} \Longrightarrow \{Y \rightarrow X\}$. Let also Q'_1 be the augmented form of PTPQ Q_1 with respect to $\mathcal{G}$ and $\mathcal{R}$. Then $Q_1 \subseteq_{\mathcal{G}} Q_2$ if and only if there is a homomorphism from Q_2 to Q'_1.* □

Proof: If $\mathcal{G}$ is a tree, Q_1 (resp. Q_2) has a single dimension tree U_1 (resp. U_2) on $\mathcal{G}$. Clearly, $Q_1 \equiv_{\mathcal{G}} U_1$.

If there is a homomorphism from Q_2 to Q'_1, then by Proposition 4.2.1, $Q'_1 \subseteq_{\mathcal{G}} Q_2$. Since $\mathcal{G}$ is a tree, Q'_1 is equivalent to U_1. Since $Q_1 \equiv_{\mathcal{G}} U_1$, $Q_1 \subseteq_{\mathcal{G}} Q_2$.

If $Q_1 \subseteq_{\mathcal{G}} Q_2$ then, from Theorem 4.2.1, there is a homomorphism from U_2 to U_1. Then, since Q'_1 is equivalent to U_1, there is a homomorphism h from U_2 to Q'_1. By definition 4.2.2, there is a mapping m' from the nodes of Q_2 to those of U_2. The composition of h on m', $h \circ m'$, is a homomorphism of Q_2 to Q'_1. □

Using multiple rules

Using additional rules in the heuristic approach improves its accuracy.

Example 4.3.3. *Consider PTPQ Q'_2 of Figure 4.8 (a slight variation of PTPQ Q_2 of Figure 3.9). Consider also PTPQ Q_3 of Figure 3.10, and the dimension graph $\mathcal{G}$*

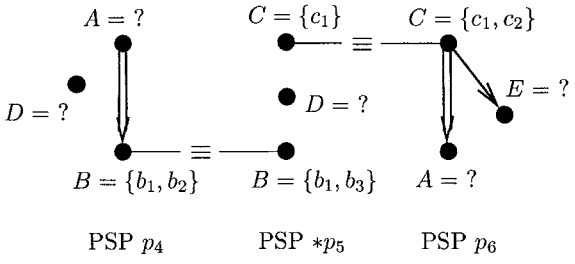

Figure 4.8 PTPQ Q'_2.

of Figure 3.4. Figure 4.7 shows PTPQ Q'_3, the augmented form of PTPQ Q_3 with respect to $\mathcal{G}$ and rule $\mathcal{R}$: $\{X \Rightarrow Y\} \Longrightarrow \{U \Rightarrow V\}$. There is no homomorphism from Q'_2 to Q'_3. Therefore, a heuristic approach that uses only $\mathcal{R}$ fails to detect that $Q_3 \subseteq_{\mathcal{G}} Q'_2$. Let's assume now that the heuristic approach employs not only rule $\mathcal{R}$ but also rule $\mathcal{R}'$: $\{X \Rightarrow Y\} \Longrightarrow \{U \rightarrow V\}$. Figure 4.9 shows PTPQ Q''_3, the augmented form of PTPQ Q_3 with respect to $\mathcal{G}$ and $\{\mathcal{R}, \mathcal{R}'\}$. Clearly, there is a homomorphism from Q'_2 to Q''_3. Therefore, the heuristic approach that uses both rules succeeds in deducing that $Q_3 \subseteq_{\mathcal{G}} Q'_2$. □

The gains in accuracy are obtained at the expense of (a) additional time for determining the rule instances that hold on $\mathcal{G}$, (b) extra space for storing those rule instances, and (c) additional time for computing the augmented form of the PTPQ (possibly more rule instances to be checked for application, more precedence

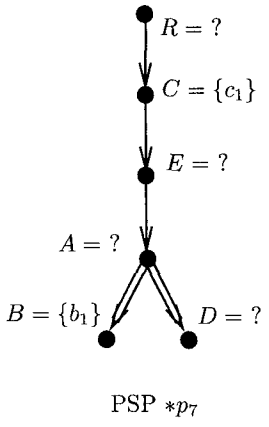

Figure 4.9 PTPQ Q_3'', the augmented PTPQ Q_3 with respect to $\mathcal{G}$ and $\{\mathcal{R}, \mathcal{R}'\}$.

relationships to be added to the PTPQ, and more iterations in the computation of the augmented form of the PTPQ). These drawbacks can be alleviated if we use a sequence of rules where each one is *more refined* than the previous one. We first explain what "more refined" means.

Definition 4.3.5. *Let $\mathcal{R}$ and $\mathcal{R}'$ be two rules. We say that $\mathcal{R}'$ is* more refined *than $\mathcal{R}$, denoted $\mathcal{R} \prec \mathcal{R}'$, if for every instance $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ of $\mathcal{R}$, and every premise $\mathcal{P}'_I$ of a rule instance of $\mathcal{R}'$ such that $\mathcal{P}'_I \models \mathcal{P}_I$,[1] there is an instance $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ of $\mathcal{R}'$, such that $\mathcal{C}_I \subseteq \mathcal{C}'_I$.* □

Example 4.3.4. *Let $\mathcal{R}$ be the rule $\{R \Rightarrow X\} \Longrightarrow \{Y \Rightarrow X\}$,*
$\mathcal{R}'$ be the rule $\{R \Rightarrow X, R \Rightarrow Y\} \Longrightarrow \{U \rightarrow V, U \Rightarrow V\}$, and $\mathcal{R}''$ be the rule $\{X \Rightarrow Y\} \Longrightarrow \{U \rightarrow V, U \Rightarrow V\}$, where X, Y, U and V are dimension variables.
It is obvious that $\mathcal{R} \prec \mathcal{R}'$, and $\mathcal{R}' \prec \mathcal{R}''$. □

Clearly, $\prec$ is a partial order on the set of rules. One can see that if we disallow the trivial rules $\{R \Rightarrow X\} \Longrightarrow \{R \Rightarrow X\}$ and $\{R \rightarrow X\} \Longrightarrow \{R \rightarrow X\}$ in the set of rules, the rules $\{R \Rightarrow X\} \Longrightarrow \{R \rightarrow X\}$, $\{R \Rightarrow X\} \Longrightarrow \{Y \Rightarrow X\}$ and $\{R \Rightarrow X\} \Longrightarrow \{R \Rightarrow Y\}$ are minimal elements of $\prec$.

[1] Implication of a set of precedence relationships is a straightforward extension of the implication of a single precedence relationship: $\mathcal{P}'_I \models \mathcal{P}_I$ iff $\forall \theta \in \mathcal{P}_I, \mathcal{P}'_I \models \theta$.

The utility of a sequence of rules where each rule is more refined than its previous one is based on the following proposition.

Proposition 4.3.3. *Let $\mathcal{R}$ and $\mathcal{R}'$ be two rules such that $\mathcal{R} \prec \mathcal{R}'$, and $\mathcal{G}$ be a dimension graph. Then, for every instance $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ of $\mathcal{R}'$, and every instance $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ of $\mathcal{R}$ that hold on $\mathcal{G}$, if $\mathcal{P}'_I \models \mathcal{P}_I$ then $\mathcal{C}_I \subseteq \mathcal{C}'_I$.* □

Proof: Let $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$, $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ be two instances of $\mathcal{R}$ and $\mathcal{R}'$ respectively, that hold on $\mathcal{G}$ such that $\mathcal{P}'_I \models \mathcal{P}_I$. Since $\mathcal{P}'_I \models \mathcal{P}_I$, the set S' of from-the-root paths in $\mathcal{G}$ that satisfy all the precedence relationships in $\mathcal{P}'_I$ is a subset of the set S of from-the-root paths in $\mathcal{G}$ that satisfy all the precedence relationships in $\mathcal{P}_I$. Therefore, the set of precedence relationships satisfied by all the paths in S is a subset of those satisfied by all the paths S'. Since $\mathcal{R} \prec \mathcal{R}'$, $\mathcal{C}_I \subseteq \mathcal{C}'_I$. □

Example 4.3.5. *Consider the instances $\mathcal{R}_I, \mathcal{R}'_I$ and $\mathcal{R}''_I$ of the rules $\mathcal{R}, \mathcal{R}'$ and $\mathcal{R}''$ of Example 4.3.4.*

$\mathcal{R}_I$: $\{R \Rightarrow A\} \Longrightarrow \{R \Rightarrow A, E \Rightarrow A\}$,

$\mathcal{R}'_I$: $\{R \Rightarrow A, R \Rightarrow C\} \Longrightarrow \{R \Rightarrow E, R \Rightarrow A,\ R \Rightarrow C, E \rightarrow A, E \Rightarrow A\}$, *and*

$\mathcal{R}''_I$: $\{A \Rightarrow C\} \Longrightarrow \{R \Rightarrow E, R \Rightarrow A, R \Rightarrow B, R \Rightarrow C,\ R \rightarrow E, E \rightarrow A, B \rightarrow C, E \Rightarrow A, E \Rightarrow B, E \Rightarrow C,\ A \Rightarrow B, A \Rightarrow C, B \Rightarrow C\}$.

Since $\{A \Rightarrow C\} \models \{R \Rightarrow A, R \Rightarrow C\} \models \{R \Rightarrow A\}$, *rule instances $\mathcal{R}_I, \mathcal{R}'_I$ and $\mathcal{R}''_I$ confirm Proposition 4.3.3.* □

As a consequence of Proposition 4.3.3, if two rules $\mathcal{R}$ and $\mathcal{R}'$ are employed in the heuristic approach and $\mathcal{R} \prec \mathcal{R}'$, we can use an incremental technique for storing their instances that hold on $\mathcal{G}$, and for computing the augmented form of a PTPQ. We explain below this incremental technique for rule instance storage and augmented PTPQ computation.

Let $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ be an instance of $\mathcal{R}'$ that holds on $\mathcal{G}$, and $\mathcal{P}^1_I \Longrightarrow \mathcal{C}^1_I, \ldots, \mathcal{P}^k_I \Longrightarrow \mathcal{C}^k_I$ be the instances of $\mathcal{R}$ that hold on $\mathcal{G}$ such that $\mathcal{P}'_I \models \mathcal{P}^i_I$, $i = 1, \ldots, k$. Then,

instead of storing $\mathcal{C}'_I$ for rule instance $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$, it suffices to store only the precedence relationships in $\mathcal{C}'_I - \cup_{i\in[1,k]}\mathcal{C}^i_I$. The rest of the extracted precedence relationships for $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ can be recovered from the precedence relationships stored for the rule instances $\mathcal{P}^i_I \Longrightarrow \mathcal{C}^i_I$, $i = 1, \ldots, k$. Further, during the computation of the augmented form of a PTPQ, if the precedence relationships in the premise $\mathcal{P}'_I$ of a rule instance $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ appear in the PP p of a PTPQ, only the precedence relationships in $\mathcal{C}'_I - \cup_{i\in[1,k]}\mathcal{C}^i_I$ need to be added to p: since $\mathcal{P}'_I \models \mathcal{P}^i_I$, $i = 1, \ldots, k$, the precedence relationships in the premise $\mathcal{P}^i_I$ of the rule instances $\mathcal{P}^i_I \Longrightarrow \mathcal{C}^i_I$, $i = 1, \ldots, k$, also appear in p and therefore, the precedence relationships in $\mathcal{C}^i_I$, $i = 1, \ldots, k$, will be added to p during some step of the computation. Notice that the incremental rule instance storage and augmented form computation technique can also be applied recursively to the rule instances $\mathcal{P}^i_I \Longrightarrow \mathcal{C}^i_I$, $i = 1, \ldots, k$.

The previous incremental technique is called *vertical* because it exploits overlapping among rule instances of different rules. Besides the vertical, we can also apply a *horizontal* incremental technique for rule instance storage and augmented form computation. This one exploits overlapping among rule instances of the same rule. Consider, for instance, the rule $\mathcal{R}: \{X \Rightarrow Y\} \Longrightarrow \{U \Rightarrow V\}$, and its two instances $\mathcal{R}^1_I: \{R \Rightarrow A\} \Longrightarrow \{R \Rightarrow A, R \Rightarrow E, E \Rightarrow A\}$ and $\mathcal{R}^2_I: \{R \Rightarrow B\} \Longrightarrow \{R \Rightarrow A, R \Rightarrow E, E \Rightarrow A, R \Rightarrow B, E \Rightarrow B, A \Rightarrow B\}$ that hold on the dimension graph $\mathcal{G}$ of Figure 3.4. Since the premise $R \Rightarrow A$ of $\mathcal{R}^1_I$ appears in the conclusion of $\mathcal{R}^2_I$, the precedence relationships $E \Rightarrow B, A \Rightarrow B$ in the conclusion of $\mathcal{R}^2_I$ (that also appear in the conclusion of $\mathcal{R}^1_I$) need not be stored with $\mathcal{R}^2_I$. During the computation of the augmented form of a PTPQ with respect to $\mathcal{G}$ and $\mathcal{R}$, $\mathcal{R}^1_I$ will be applicable to a PP any time $\mathcal{R}^2_I$ is applicable. Therefore, the missing precedence relationships $E \Rightarrow B, A \Rightarrow B$ from the conclusion of $\mathcal{R}^2_I$ will be added to this same PP by the mandatory application of $\mathcal{R}^1_I$.

Both incremental techniques, the vertical and the horizontal one, are used in

the experimental evaluation of the precomputation heuristic approach presented in the next section.

Rule selection for the precomputation heuristic approach

Usually, we are interested in rules characterizing paths in the dimension graph that gradually involve: (1) one dimension, (2) two dimensions with no specific order between them, (3) a descendant precedence relationship between two dimensions, and (4) a child precedence relationship between two dimensions. We want these rules to extract both child and descendant precedence relationships. We therefore initially consider the following sequence of rules:

$\mathcal{R}_1'$: $\{R \Rightarrow X\} \Longrightarrow \{U \Rightarrow V,\ U \rightarrow V\}$,

$\mathcal{R}_2$: $\{R \Rightarrow X,\ R \Rightarrow Y\} \Longrightarrow \{U \Rightarrow V, U \rightarrow V\}$,

$\mathcal{R}_3$: $\{X \Rightarrow Y\} \Longrightarrow \{U \Rightarrow V,\ U \rightarrow V\}$, and

$\mathcal{R}_4$: $\{X \rightarrow Y\} \Longrightarrow \{U \Rightarrow V,\ U \rightarrow V\}$,

where X, Y, U, V are dimension variables. These rules can be simplified as we show below.

Two rules are *computationally equivalent*, denoted $\equiv_c$, if they generate the same augmented form, for any input PTPQ and any dimension graph. Computational equivalence can be extended to sets of rules in a straightforward way.

Rule $\mathcal{R}_4$ is redundant in the presence of rule $\mathcal{R}_3$. This is shown by the proposition below and allows us to exclude $\mathcal{R}_4$ from further consideration.

Proposition 4.3.4. $\{\mathcal{R}_3, \mathcal{R}_4\} \equiv_c \{\mathcal{R}_3\}$. □

Proof: Let A and B be two dimensions in the dimension graph $\mathcal{G}$. Let $\mathcal{P}_I^3 \Longrightarrow \mathcal{C}_I^3$ and $\mathcal{P}_I^4 \Longrightarrow \mathcal{C}_I^4$ be the instances of $\mathcal{R}_3$ and $\mathcal{R}_4$ that hold on $\mathcal{G}$ for X and Y instantiated to A and B respectively. If the PTPQ does not contain the precedence relationship $A \rightarrow B$ then $\mathcal{P}_I^4 \Longrightarrow \mathcal{C}_I^4$ cannot be used to compute its augmented form. Thus, the proposition holds. Otherwise, if there is no edge from A to B in $\mathcal{G}$, then $\mathcal{P}_I^4 \Longrightarrow \mathcal{C}_I^4$

does not hold on $\mathcal{G}$ and therefore, it cannot be used to compute the augmented form of a PTPQ that contains $A \rightarrow B$, and the proposition holds again. Let's now assume that the PTPQ contains $A \rightarrow B$ and there is an edge from A to B in $\mathcal{G}$. Then, $\mathcal{C}_I^4$ contains exactly all the precedence relationships that are satisfied by every path of $\mathcal{G}$ from the root to A which does not go through B and the precedence relationship $A \rightarrow B$. $\mathcal{C}_I^4$ contains exactly all the precedence relationships that are satisfied by every path of $\mathcal{G}$ from the root to A which does not go through B and possibly the precedence relationship $A \rightarrow B$. Since the PTPQ contains $A \rightarrow B$, the effect of the two rule instances in the computation of the augmented form of the PTPQ is the same. □

The next proposition states that in rule $\mathcal{R}_1'$, we can restrict our attention only to extracted precedence relationships from some dimension *to* dimension X. Consider the following rule:

$\mathcal{R}_1$: $\{R \Rightarrow X\} \Longrightarrow \{U \Rightarrow X,\ U \rightarrow X\}$.

Proposition 4.3.5. $\mathcal{R}_1' \equiv_c \mathcal{R}_1$. □

Proof: Let $\mathcal{G}$ be a dimension graph and Q be a PTPQ. Clearly, if a precedence relationship is added to Q during the computation of its augmented form by rule $\mathcal{R}_1$, it is also added to it during the computation of its augmented form by rule $\mathcal{R}_1'$. Let now $A \Rightarrow B$ be a precedence relationship added to Q during the computation of its augmented form by the instance of $\mathcal{R}_1'$ whose premise is $R \Rightarrow C$. We show that $A \Rightarrow B$ will be also added to Q during the computation of its augmented form by an instance of $\mathcal{R}_1$. If B and C are the same dimension, then $A \Rightarrow B$ is also added to the augmented form of Q by the instance $\mathcal{R}_1$ whose premise is $R \Rightarrow C$. If B and C are distinct dimensions, every path from the root of $\mathcal{G}$ to C also goes through dimensions A, B and C in that order. This implies that every path from the root of $\mathcal{G}$ to B also goes through A (since otherwise there would be a path from the root of

$\mathcal{G}$ to C that goes through B without going through A, which contradicts that every path from the root of $\mathcal{G}$ to C also goes through A and B). Since every path from the root of $\mathcal{G}$ to C also goes through B, $B \Rightarrow C$ will be added to the PTPQ during the computation of its augmented form (if not already there) by the instance of $\mathcal{R}_1$ whose premise is $R \Rightarrow C$. When the instance of $\mathcal{R}_1$ whose premise is $R \Rightarrow B$ is considered in the computation of the augmented form of Q, the precedence relationship $A \Rightarrow B$ will be added to Q. Similarly we prove that if $A \rightarrow B$ is added to Q during the computation of its augmented form by an instance of $\mathcal{R}'_1$, it is also added to Q during the computation of its augmented form by an instance of $\mathcal{R}_1$ □

Rules $\mathcal{R}'_1$ and $\mathcal{R}_1$ have the same premise, while the conclusion of $\mathcal{R}_1$ is more restrictive than that of $\mathcal{R}'_1$. Therefore, if we use $\mathcal{R}_1$ instead of $\mathcal{R}'_1$ we reduce both: (a) the storage space needed for the rule instances that hold on a dimension graph, and (b) the attempts to add extracted precedence relationships to a PTPQ which have already been extracted from other rule instances.

Clearly, it is $\mathcal{R}_1 \prec \mathcal{R}_2 \prec \mathcal{R}_3$. Thus, we can apply the "vertical" (across the rules in the sequence) incremental technique for storing rule instances and for computing augmented forms of PTPQs. This technique was discussed in Section 4.3.2. The application of a "horizontal" (across the instances of the same rule) incremental technique for rule instance storage and augmented PTPQ computation is based on the following proposition.

Proposition 4.3.6. Let $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ and $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ be two instances of rule $\mathcal{R}_3$ that hold on a dimension graph. Let also θ and θ' be two precedence relationships. If $\theta \in \mathcal{P}_I$, $\theta' \in \mathcal{C}_I$, and $\theta \in \mathcal{C}'_I$ then $\theta' \in \mathcal{C}'_I$. □

Proof: Let $\mathcal{G}$ be the dimension graph. Since $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ holds on $\mathcal{G}$, $\theta \in \mathcal{P}_I$, and $\theta' \in \mathcal{C}_I$, every path from the root of $\mathcal{G}$ that satisfies θ, also satisfies θ'. Since, $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ holds on $\mathcal{G}$, and $\theta \in \mathcal{C}_I$, every path from the root of $\mathcal{G}$ that satisfies $\mathcal{P}'_I$ also satisfies θ.

Therefore, it also satisfies θ', that is, $\theta' \in \mathcal{C}'_I$. □

As a consequence, if θ is stored in $\mathcal{C}'_I$, θ' need not be stored explicitly as an extracted precedence relationship in $\mathcal{C}'_I$. Precedence relationship θ' can be extracted from $\mathcal{P}_I \Longrightarrow \mathcal{C}_I$ using (possibly recursively) other rule instances. The contracted form of $\mathcal{C}'_I$ is also used in the application of rule instance $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$. Precedence relationship θ' which is not added to a PP by $\mathcal{P}'_I \Longrightarrow \mathcal{C}'_I$ is added to it by a (possibly recursive) application of other rule instances.

In the experimental evaluation part of the paper (Section 4.4), we examine a family of three precomputation heuristic approaches H_1, H_2 and H_3, which gradually involve more rules: H_1 involves $\mathcal{R}_1$; H_2 involves $\mathcal{R}_1$ and $\mathcal{R}_2$; and H_3 involves $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$. The next proposition shows that the precedence relationships for rules $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ can be extracted efficiently.

Proposition 4.3.7. *The precedence relationships for the instances of rules $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ can be extracted from the dimension graph $\mathcal{G}$ in polynomial time on the number of dimensions in $\mathcal{G}$.* □

Proof: The proof is similar to the proof of Proposition 4.1.2 where we showed how to compute all the precedence relationships that hold on all paths containing a given precedence relationship. □

Computing the augmented form of a PTPQ involves iteratively adding extracted precedence relationships to a PTPQ and computing its full form. As mentioned in Section 3.4, the full form of a PTPQ can be computed in polynomial time on n, where n is the product of the number of PPs and the number of distinct dimensions in the PTPQ. The number of precedence relationships that can be extracted from $\mathcal{G}$ for a precedence relationships is $O(m^2)$, where m is the number of distinct dimensions in $\mathcal{G}$, and can be done in a polynomial time in m (Proposition 4.3.7). A PP of a PTPQ can contain at most $O(m^2)$ precedence relationships. Thus, the

augmented form of a PTPQ can be computed in time polynomial on n. Therefore, the heuristic approach which involves rules $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ runs in polynomial time on n.

4.3.3 On-the-fly Heuristic Approach

Using a rule instance whose premise comprises all the precedence relationships in a PP of a PTPQ Q, we can extract for this PP, in general, more precedence relationships from $\mathcal{G}$ compared to using the rule instances of a given set of rules. Based on this remark, we provide a new augmented form for PTPQs.

Definition 4.3.6. *Consider a PTPQ Q and a dimension graph $\mathcal{G}$. The* augmented *form of Q with respect to $\mathcal{G}$ is the PTPQ constructed from Q by iteratively: (a) adding to every PP of Q (child and descendant) precedence relationships extracted from $\mathcal{G}$ using all precedence relationships in the PP, and (b) computing the full form of the resulting PTPQ. The process stops when a fixed point is reached.* □

Clearly, the augmented form of PTPQ Q with respect to G is equivalent to Q, and is not less restrictive than the augmented form of PTPQ Q with respect to G and a given set of rules.

Example 4.3.6. *Consider, the PTPQ Q_4 of Figure 4.10(a), and the dimension graph $\mathcal{G}$ of Figure 3.4. Figure 4.10(b) shows the augmented form of Q_4 with respect to $\mathcal{G}$. This PTPQ is a fully specified tree-pattern PTPQ (without descendant precedence relationships). It is not difficult to see that it cannot be obtained from Q_4 using any set of rules that have one precedence relationship type in their premise.* □

Nevertheless, with the on-the-fly heuristic approach, the extraction of precedence relationships can only be performed after the PTPQ is issued. Therefore, PTPQ containment checking is subject to the additional cost of precedence relationship extraction. In extreme cases the number of from-the-root paths in the dimension

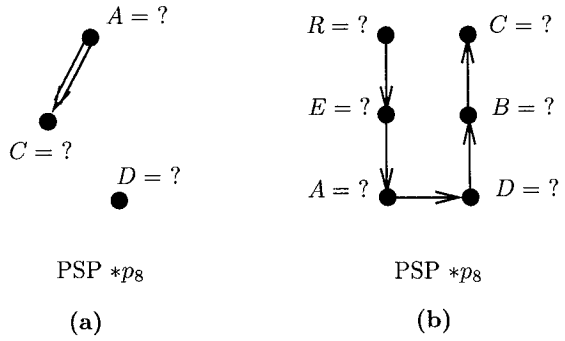

Figure 4.10 (a) PTPQ Q_4, (b) Augmented form of Q_4 w.r.t. $\mathcal{G}$.

graph $\mathcal{G}$ is exponential on the number of nodes in the dimension graph. In practice, the number of these paths in $\mathcal{G}$ is restricted and the approach performs much better than computing all the dimension trees of the PTPQs.

4.4 Experimental Evaluation

The experiments were implemented and executed by our colleagues from the National Technical University of Athens, S. Souldatos and T. Dalamagas, who collaborated on the work presented in this chapter.

We present in this section an implementation of our approaches for checking PTPQ containment and we report on their experimental evaluation.

4.4.1 Experimental Setup

To study the effectiveness of our PTPQ containment checking approaches, we ran a comprehensive set of experiments. Checking PTPQ containment in the presence of dimension graphs, is expected to be time consuming. However, our experimental evaluation shows that the heuristic approaches for checking containment can save a considerable amount of time, while maintaining high accuracy.

For the experiments, we considered tree structured data encoded as XML documents. We assumed that dimensions are syntactic "objects" that comprise exactly

the elements with the same tag in the XML document. We used dimension graphs whose number of root-to-leaf paths does not exceed five times the number of their nodes. This is in conformance with the dimension graphs of several popular XML benchmarks, like XMark[2] and XMach[3], where the number of root-to-leaf paths does not exceed twice the number of their nodes.

We implemented a graph generator to construct random dimension graphs, given a set of dimensions and a number of root-to-leaf paths in the graph. The generator guarantees that the graphs constructed are dimension graphs (that is, they satisfy the conditions of Proposition 1). We construct graphs as follows. First we generate a random set of node (i.e., dimension) paths. Then, we merge these paths to construct a dimension graph. If the graph does not have the requested number of root-to-leaf paths, we add a new random path or we replace a path in the set with another random path and we repeat the merging.

In our experiments, we compared the execution time and the accuracy for containment check with respect to a dimension graph $\mathcal{G}$. We measure accuracy by the percentage of pairs (Q_1, Q_2) of PTPQs with $Q_1 \subseteq_{\mathcal{G}} Q_2$ out of a set of randomly generated pairs (Q_i, Q_j) of PTPQs satisfying the following conditions: (a) Q_i and Q_j are satisfiable with respect to $\mathcal{G}$, and (b) there is not a homomorphism from Q_j to Q_i. Therefore, we consider pairs of PTPQs such that containment cannot be detected based on Proposition 4.2.1. The accuracy of our heuristic approaches for randomly generated pairs of PTPQs without the above restrictions is expected to be even higher.

We implemented a PTPQ generator to construct pairs of randomly generated PTPQs satisfying the conditions (a) and (b) above, given a dimension graph $\mathcal{G}$, the number p of PPs in the PTPQs, and the number n of nodes per PP. Since the

[2]http://monetdb.cwi.nl/xml/

[3]http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html

generator needs to make sure that the PTPQs are satisfiable with respect to $\mathcal{G}$, it proceeds as follows to construct a PTPQ Q_i: (a) it extracts a "dimension tree" from $\mathcal{G}$ with p root-to-leaf paths, (b) it "splits" the paths in the tree by adding node sharing expressions between the common nodes to create a partial tree-pattern PTPQ, (c) it computes the full form of the PTPQ, and (d) it randomly removes precedence relationships and node sharing expressions leaving n nodes in every PP. The PTPQ generator repeats the process to construct another PTPQ Q_j, and checks whether there is a homomorphism from Q_j to Q_i. If there is such a homomorphism, it repeats the process until it finds a Q_j such that there is no homomorphism from Q_j to Q_i.

Our measurements involve the following cases: (a) checking PTPQ containment based on Theorem 4.2.1 (case C), (b) checking PTPQ containment using the on-the-fly heuristic approach (case CFH), (c) checking PTPQ containment using the three precomputation heuristic approaches H_1, H_2 and H_3 discussed in Section 4.3.2 (cases CH_1, CH_2 and CH_3 respectively).

We ran our experiments on a dedicated Linux PC (AMD Sempron 2600+) with $2GB$ of RAM. The reported values are the average of repeated measurements. Specifically, for every measure point, 100 pairs of PTPQs were generated (10 pairs of PTPQs for each one of the 10 dimension graphs used).

4.4.2 Experimental Results

Execution time and accuracy varying the density of the dimension graph. We measured the execution time and the accuracy for checking PTPQ containment varying the number of root-to-leaf paths for different numbers of nodes in the dimension graph. In Figures 4.11 and 4.13, we present the results obtained for dimension graphs having 10, 20, 30 and 40 nodes. The number of PPs in the PTPQs and the number of nodes per PP are fixed to 2 and 4, respectively.

As expected, checking for homomorphisms between several pairs of dimension

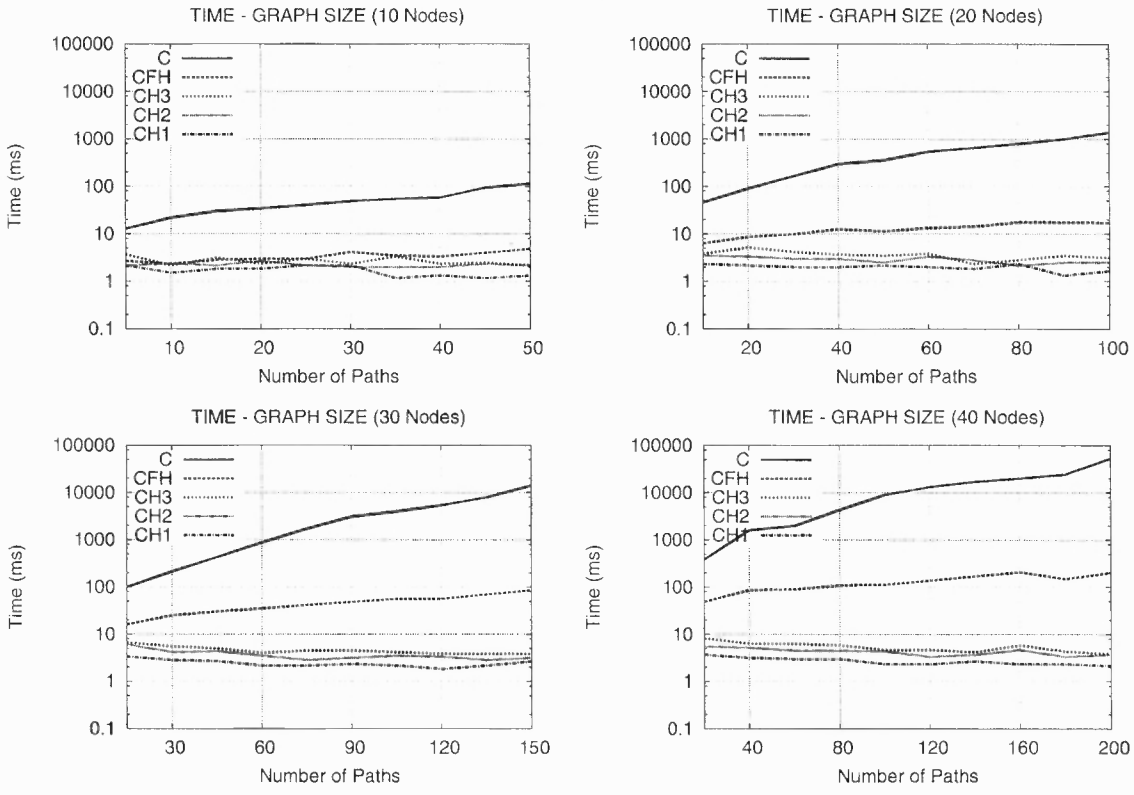

Figure 4.11 Execution time for checking PTPQ containment varying the number of root-to-leaf paths for 10, 20, 30 and 40 nodes in the dimension graph.

trees is expensive compared to checking for a homomorphism between two PTPQs. The larger the number of paths in the dimension graph, the more is the time taken by case C. This is due to the increase in the number of matchings of the PPs to the paths of the dimension graph. Such an increase causes more dimension trees to be produced.

Overall, our results show that all of our heuristic approaches clearly improve the execution time of case C. Note that CH_1 is the fastest among all the heuristic approaches we suggest, giving in some cases an improvement of more than two orders of magnitude compared to case C.

The execution time for cases CH_1, CH_2 and CH_3 slightly drops as the number of paths in the dimension graph increases. The reason is that the density of the dimension graph increases, too, which in turn decreases the number of precedence

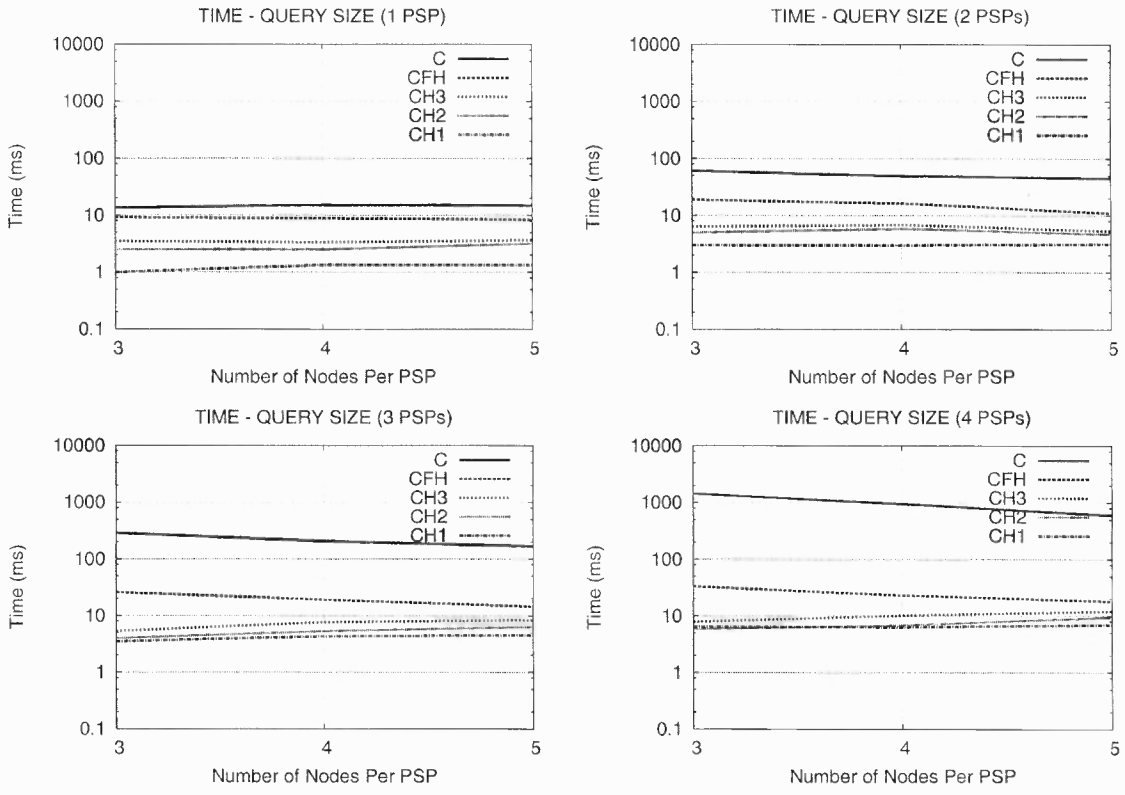

Figure 4.12 Execution time for checking PTPQ containment varying the number of nodes per PP for 1, 2, 3 and 4 PPs in the PTPQ.

relationships extracted from the graph. Note that the precedence relationships from the dimension graph are precomputed. Thus, the execution time does not include the time required to extract the precedence relationships from the dimension graph.

For a growing number of paths in the dimension graph, the execution time for the on-the-fly heuristic approach CFH increases. This is caused by the increase in the number of paths examined during the (on-the-fly) precedence relationship extraction.

Regarding the accuracy, the on-the-fly heuristic approach CFH is clearly more accurate than all the other heuristic approaches, approximating 100% of the accuracy of the non-heuristic containment check approach C. Heuristic cases CH_1, CH_2 and CH_3 have an accuracy higher than 45%, 65% and 85%, respectively, for dimension graphs whose number of root-to-leaf paths does not exceed twice the number of their

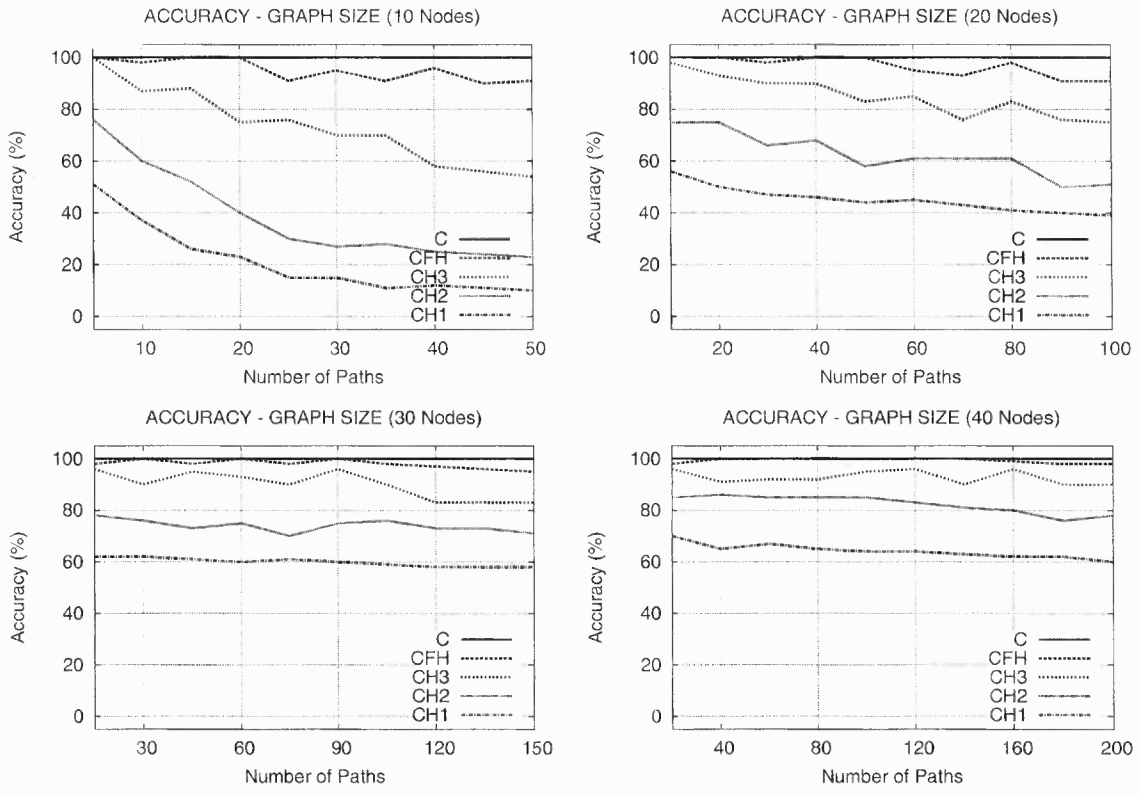

Figure 4.13 Percentage of correct answers in checking relative PTPQ containment varying the number of root-to-leaf paths for 10, 20, 30 and 40 nodes in the dimension graph.

nodes.

Execution time and accuracy varying the density of PTPQs. We measured the execution time and the accuracy for checking PTPQ containment varying the number of nodes per PP for different numbers of PPs in the PTPQs. In Figures 4.12 and 4.14, we present the results obtained for PTPQs having 1, 2, 3 and 4 PPs. The number of nodes and paths in the dimension graph are fixed to 30 and 15 respectively.

The execution time of case C goes up as the number of PPs in the PTPQs increases. The reason is that a larger number of dimension trees are generated and, thus examined in the containment check. On the other hand, the execution time of case C decreases as the number of nodes per PP goes up, since more restricted PTPQs result in a smaller number of dimension trees to be examined in the containment check.

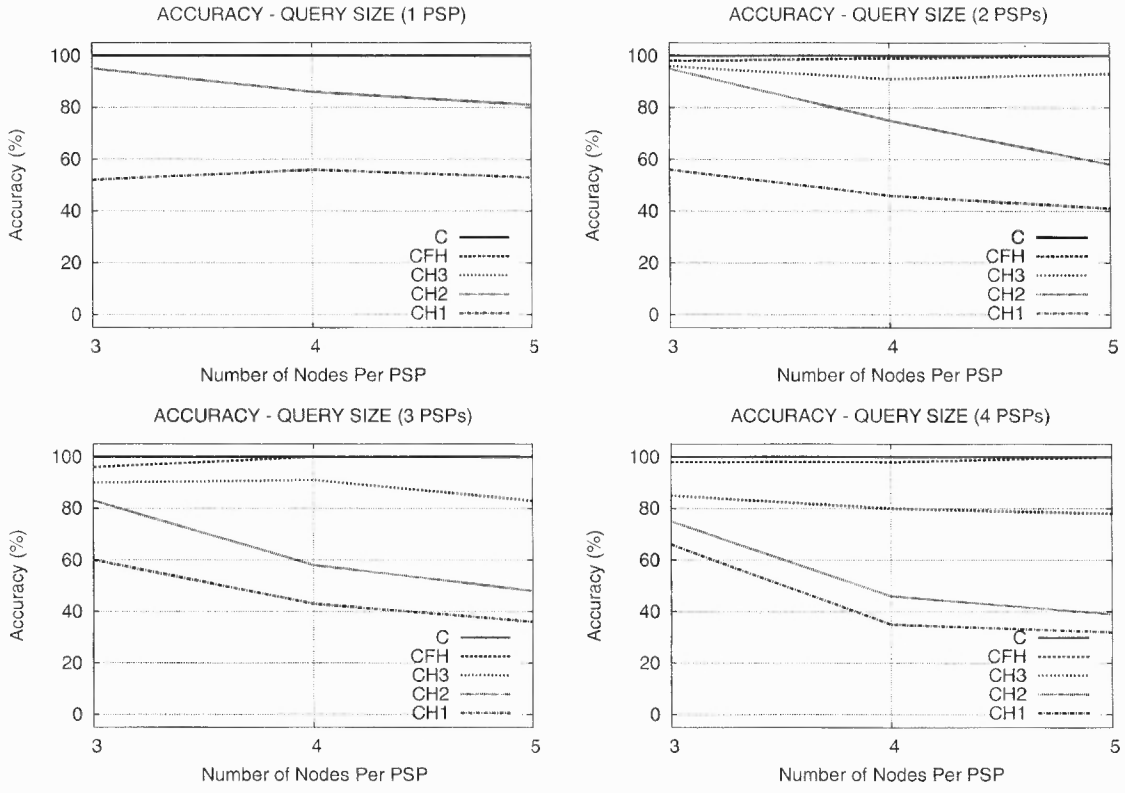

Figure 4.14 Percentage of correct answers in checking relative PTPQ containment varying the number of nodes per PP for 1, 2, 3 and 4 PPs in the PTPQ.

Again, our heuristic approaches clearly improve case C, with Case CH_1 being the fastest among all. For a growing number of nodes per PP in the PTPQ, the execution time of the precomputation heuristic containment cases CH_1, CH_2 and CH_3 is only slightly affected. On the contrary, the larger is the number of PP nodes, the less is the time spent by the approach CFH. This is due to the decrease in the number of paths examined during the (on-the-fly) precedence relationship extraction from the dimension graph.

The accuracy of approach CFH is close to 100%. The accuracy for the other heuristic approaches decreases as the number of nodes per PP in the PTPQs increases. However, the accuracy of case CH_3 is almost in all cases above 80% for an execution time which is close to that of cases CH_1 and CH_2.

Remarks. All of our heuristic approaches clearly improve the time of approach C.

Even though these approaches are sound, they are not complete. Therefore, a trade-off has to be determined between desired accuracy on the one side and time resources on the other side. Our experiments show clearly the benefit of using the on-the-fly heuristic approach CFH when accuracy is the goal. Approach CFH is more than one order of magnitude faster than approach C, while scoring an accuracy close to 100%. When efficiency is important, a full spectrum of precomputation heuristic approaches (including possibly those that involve additional rules besides $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$) gradually trade accuracy for efficiency.

4.5 Conclusion

In this chapter we studied the problem of PTPQ containment in the presence of dimension graphs, and we provided necessary and sufficient conditions for PTPQ containment. We further devised sound but not complete heuristic approaches that exploit structural information extracted from the dimension graph either in advance or at query time. A detailed experimental evaluation of our approaches shows that they greatly improve the PTPQ containment checking execution time, and that they gradually trade execution time for accuracy. These results allow their use for query processing and optimization.

CHAPTER 5

HEURISTIC APPROACHES FOR CHECKING CONTAINMENT OF PARTIAL TREE-PATTERN QUERIES

In this chapter we provide results and heuristic approaches for checking PTPQ containment in the absence of the dimension graphs. The language for PTPQ we consider in this chapter is somewhat simplified, since we do not allow value predicates for the elements. Also the answer of a PTPQ is a set of nodes instead of subtree, in compliance with XPath.

5.1 Query Language

We start by formally presenting the query language and data model. Let $\mathcal{E}$ be an infinite set of elements that includes a distinguished element r. A *database* is a finite tree of nodes labeled by elements in $\mathcal{E}$, rooted at a node labeled by r (such a root node can always be added to a data tree if it is not initially there). For simplicity, we assume that the same element does not label two nodes on the same path. The attributes of an XML document are modeled here using the element nodes of such a tree.

Definition 5.1.1. A *Partial Tree-Pattern Query* (PTPQ) is a triple $Q = (\mathcal{P}, \mathcal{N}, o)$, where:

(a) $\mathcal{P}$ is a nonempty set of pairs $(p, \mathcal{R})$ called *Partial Paths* (PPs). p is the name of the PP. $\mathcal{R}$ is a set of expressions of the form $e_i \rightarrow e_j$ (*child precedence relationship*) and/or $e_i \Rightarrow e_j$ (*descendant precedence relationship*), where e_i and e_j are distinct elements. The names of the PPs in Q are distinct. Therefore, we identify PPs in Q with their names. The expression $e[p]$ denotes the element e in PP p.

(b) $\mathcal{N}$ is a set of expressions of the form $e[p_i] \approx e[p_j]$, where p_i and p_j are PPs in $\mathcal{P}$, and e is an element. These expressions are called *node sharing expressions.* Roughly speaking, they state that PPs p_i and p_j have element e in common (they share it). Set $\mathcal{N}$ can be empty.

(c) $o[p]$ is a special node of a PP p in $\mathcal{P}$ called *output node* of Q. Intuitively, it represents the node to be returned to the user. □

The *answer* of PTPQ Q on database D is the set of the images of the output node of Q under all possible embeddings of Q to D.

Graphical representation of a PTPQ is similar to the one described in Chapter 3. The two differences are: a) the output node of Q is denoted by a filled black node, b) a node sharing expression $e[p_i] \approx e[p_j]$ is represented by an edge between element e of the PP graph p_i and element e of the PP graph p_j labeled by $\approx$. Figures 5.1, 5.2, and 5.3 show three PTPQs.

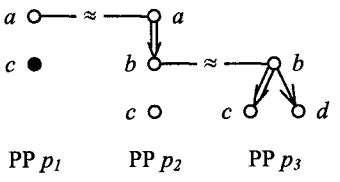

Figure 5.1 PTPQ Q_1.

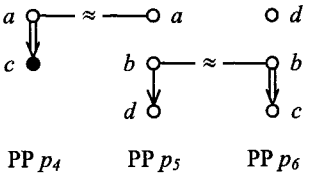

Figure 5.2 PTPQ Q_2.

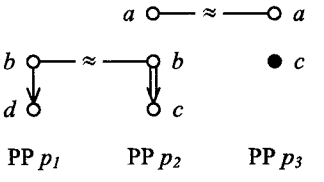

Figure 5.3 PTPQ Q_3.

Figures 5.4 and 5.5 show the PTPQs Q_1' and Q_2' which are the full forms of the PTPQs Q_1 and Q_2 of Figures 5.1 and 5.2 respectively. Query Q_3 of Figure 5.3 is in full form. Observe that the full form of Q_1 shows that this PTPQ is also a TPQ.

5.2 Component TPQs

We show now that the answer of a PTPQ on any database can be computed from a set of TPQs called component TPQs of Q. Consider a PTPQ Q. Observe that by adding descendant precedence relationships between every two nodes in the same PP

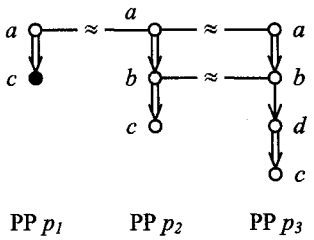

Figure 5.4 PTPQ Q'_1, the full form of Q_1.

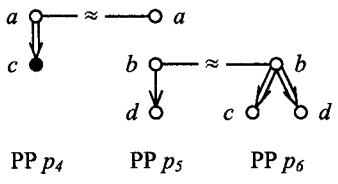

Figure 5.5 PTPQ Q'_2, the full form of Q_2.

of Q, the resulting query is an unsatisfiable PTPQ or a PTPQ equivalent to a TPQ. The following proposition determines when a PTPQ is equivalent to a TPQ.

Proposition 5.2.1. *Let Q be a PTPQ. If Q is satisfiable and there is a precedence relationship between any two nodes of the same PP in the full form of Q, Q is equivalent to a TPQ.* □

Proof: The proof is straightforward if we observe that: (a) since Q is satisfiable, the nodes in any PP form a total order that respects the precedence relationships in the full form of the query (no cycles), and (b) for any node sharing expression $a[p_1] \approx a[p_2]$ in the full form of Q, the two sequences of nodes in the two PPs p_1 and p_2 from their roots to $a[p_1]$ and $a[p_2]$ respectively, are identical and the corresponding nodes participate in node sharing expressions. Clearly, the pattern resulting by merging the nodes participating in node sharing expressions is a TPQ. □

In the following, we might use the term TPQ for a PTPQ which is equivalent to a TPQ.

Definition 5.2.1. *Let Q be a PTPQ. A component TPQ (abbreviated as cTPQ) of Q is a TPQ resulting by adding descendant precedence relationships to Q.* □

Therefore, a component TPQ of a PTPQ Q is a TPQ resulting by specifying a *total order* for the nodes in every PP of Q that respects existing precedence relationships in Q.

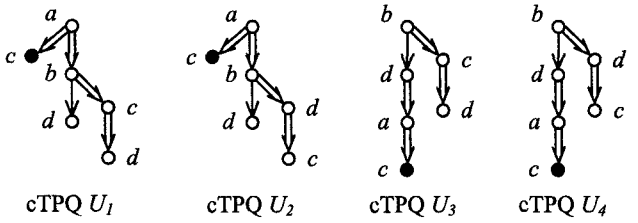

Figure 5.6 The four cTPQs of PTPQ Q_2.

Consider the PTPQ Q_2 of Figure 5.2. Figure 5.6 shows the four component TPQs U_1, U_2, U_3 and U_4 of Q_2. The PTPQ Q_1 of Figure 5.1 has only one cTPQ since its full form shown in Figure 5.4 is a TPQ.

The cTPQs of a PTPQ Q can be used to compute the answer of Q as follows:

Proposition 5.2.2. *Let $\{U_1, \ldots, U_k\}$ be the set of cTPQs of a PTPQ Q and D be a database. Let also $A, A_1, \ldots, A_k$ be the answers of $Q, U_1, \ldots, U_n$ respectively on D. Then, $A = \bigcup_{i=1,\ldots,k} A_i$.* □

Proof: Let a be a node in the answer A_i of a cTPQ U_i of Q. Clearly, the embedding defining this node as an answer of U_i is also an embedding of Q that maps the output node of Q to a. Therefore, $a \in A$. Let now, a be a node in the answer A of Q. The embedding M of Q defining this node as an answer of Q determines, for every PP p of Q, a total order for all the nodes in p that respects all the precedence relationships in p. Then, M is also an embedding of the cTPQ U_i of Q defined by these total orders, and it maps the output node of U_i to a. Therefore, $a \in A_i$. □

The previous proposition states that the answer of a PTPQ is the union of the answers of its cTPQs.

5.3 Necessary and Sufficient Conditions for PTPQ Containment

For TPQs that involve the descendent axis (that is, descendent precedence relationships), and branching ([]), but no wildcards (*), the existence of a homomorphism

is a necessary and sufficient condition for containment [3, 33]. In order to check if a similar result holds for PTPQs we extend the concept of homomorphism for TPQs to homomorphism for PTPQs:

Definition 5.3.1. *Let Q_1 and Q_2 be two queries on $\mathcal{D}$. An* homomorphism *from Q_2 to Q_1 is a mapping h from the nodes of Q_2 to the nodes of Q_1 such that: (a) nodes of Q_2 are mapped by h to nodes of Q_1 labeled by the same element, (b) nodes of Q_2 on the same PP are mapped by h to nodes of Q_1 on the same PP, (c) the output node of Q_2 is mapped under h to the output of Q_1, or to a node involved in a node sharing expression with the output node of Q_1, (d) $\forall\ e_i[p] \rightarrow e_j[p]$ (resp. $e_i[p] \Rightarrow e_j[p]$) in Q_2, $h(e_i[p]) \rightarrow h(e_j[p])$ (resp. $h(e_i[p]) \Rightarrow h(e_j[p])$) is in Q_1, and (e) $\forall\ e[p_i] \approx e[p_j]$ in Q_2, $h(e[p_i])$ and $h(e[p_j])$ coincide or $h(e[p_i]) \approx h(e[p_j])$ is in Q_1.* □

The next proposition shows that the existence of a homomorphism is a sufficient condition for PTPQ containment.

Proposition 5.3.1. *Let Q_1 and Q_2 be two PTPQs. If there is a homomorphism from Q_2 to Q_1, $Q_1 \subseteq Q_2$.* □

The proof is not difficult. Since the full form of a query Q is a query equivalent to Q we can also show the following corollary.

Corollary 5.3.1. *Let Q_1 and Q_2 be two PTPQs. If there is a homomorphism from Q_2 to the full form of Q_1, $Q_1 \subseteq Q_2$.* □

The previous corollary forms a better basis for checking containment based on the existence of a homomorphism: the full form of a PTPQ Q has at least the precedence relationships and node sharing expressions of Q. Therefore, when checking containment of Q into another PTPQ, the full form Q' of Q provides more chances than Q for a homomorphism to Q' to exist.

Example 5.3.1. *Consider the PTPQs Q_1 and Q_2 of Example 3.3.1 shown in Figures 5.1 and 5.2 respectively. One can see that there is no homomorphism from Q_2 to Q_1. However, there is a homomorphism from Q_2 to the full form of Q_1 which is shown in Figure 5.4. This proves our claim of Example 3.3.1 that $Q_1 \subseteq Q_2$.* □

Unfortunately, the existence of a homomorphism from Q_2 to Q_1 is not a necessary condition even if Q_1 is in full form. Consider, for instance, the PTPQs Q_2' and Q_3 of Figures 5.5 and 5.3 respectively. PTPQ Q_2' is the full form of PTPQ Q_2 of Figure 5.2. One can easily see that there is no homomorphism from Q_3 to Q_2'. However, as we mentioned in Example 3.3.1, $Q_2 \subseteq Q_3$.

We elaborate in Section 5.4.1 on the reasons of this behaviour of PTPQs and we identify a subclass of PTPQs for which the existence of a homomorphism between two PTPQs is a necessary condition for containment.

We now provide necessary and sufficient conditions for PTPQ containment in terms of homomorphisms from a PTPQ to TPQs:

Theorem 5.3.1. *Let Q_1 and Q_2 be two PTPQs. Let also $\mathcal{U}_1$ be the set of component TPQs for Q_1. $Q_1 \subseteq Q_2$ iff for every component TPQ $U \in \mathcal{U}_1$, there is a homomorphism from Q_2 to U.* □

Proof: (Sufficiency) If there is a homomorphism from Q_2 to every cTPQ $U_i, i = 1, \ldots, k$ of Q_1, $U_i \subseteq Q_2$. Thus, if A^1, A^2, $A_1, \ldots, A_k$ are the answers of Q_1, Q_2, $U_1, \ldots, U_k$ respectively on a database, $A_i \subseteq A^2$, $i = 1, \ldots k$. Then, $\bigcup_{k\in[1,k]} A_i \subseteq A^2$. Since $A^1 = \bigcup_{k\in[1,k]} A_i$, $A^1 \subseteq A^2$. Therefore, $Q_1 \subseteq Q_2$.

(Necessity) Let's assume that $Q_1 \subseteq Q_2$ and there is no homomorphism from Q_2 to a cTPQ U of Q_1. We create a database (tree) D based on U, by replacing descendant edges of U by simple edges. Clearly, U and therefore, Q_1 has an answer on D. However, because there is no homomorphism from Q_2 to U, there is no embedding

of Q_2 to D. Thus, Q_2 does not have an answer on D. This contradicts our assumption that $Q_1 \subseteq Q_2$. □

Example 5.3.2. *Consider again the PTPQs Q_1, Q_2 and Q_3 of Example 3.3.1 shown in Figures 5.1, 5.2, and 5.3 respectively. One can see that there is a homomorphism from Q_3 to each one of the cTPQs of Q_2 which are shown in Figure 5.6. This proves our claim of Example 3.3.1 that $Q_3 \subseteq Q_2$.*

In contrast, it is easy to see that there no homomorphism from Q_1 to at least one of the cTPQs of Q_2 (in fact, there is no homomorphism to any one of the cTPQs of Q_2). This proves our claim of Example 3.3.1 that $Q_2 \not\subseteq Q_1$. □

Unfortunately, the previous result does not lead to a practical approach for checking PTPQ containment. The reason is that the number of cTPQs of a PTPQ can be exponential on the number of nodes of the PTPQ. As an example, consider a trivial PTPQ which comprises n nodes in one PP (besides the root r) without any precedence relationship between them. This PTPQ has $n!$ cTPQs corresponding to the different orderings of these nodes. Olteanu et al. [38, 37] also showed that even though an XPath expression with reverse axes can be rewritten equivalently as a set of XPath expressions with only forward axes, this conversion might result in a number of XPath expressions with only forward axes which is exponential on the number of steps of the input XPath expression. Clearly, in our context, it is inefficient to check general PTPQ containment by checking the existence of homomorphisms between a PTPQ and an exponential number of TPQs.

5.4 PTPQs for Which Homomorphisms Are Necessary for Containment

We show in this section why the existence a homomorphism between two PTPQs does not fully characterize query containment. Then, we identify a subclass of PTPQs for which the existence of a homomorphism is a necessary condition for containment.

These results are exploited in the next section for devising heuristics approaches for checking query containment for PTPQs.

The presence of two node sharing expressions $a[p_1] \approx a[p_2]$ and $b[p_2] \approx b[p_3]$ in a PTPQ Q when no precedence relationship can be derived between $a[p_2]$ and $b[p_2]$ can create discrepancies: it may force every tree in which there is an embedding of Q to comprise a path that involves a number of nodes and satisfies a number of precedence relationships which together cannot be derived in any PP of Q. We call such a set $\{a[p_1] \approx a[p_2], b[p_2] \approx b[p_3]\}$ of two node sharing expressions a *3-path swing* because the elements $a[p_2]$ and $b[p_2]$ in PP p_2 (and their corresponding node sharing expressions) can freely "swing" above or below each other. Another query Q' that involves in the same PP all these nodes and precedence relationships might contain Q. However, there will not not exist a homomorphism from Q' to Q since these nodes and precedence relationships do not appear together in any PP of Q.

Example 5.4.1. *Consider the PTPQ Q_2 of Figure 5.2 whose full form is shown in Figure 5.5. Clearly, no precedence relationships can be derived between a and b in PP p_2 since no such relationships exists in the full form of Q_2 shown in Figure 5.5. This PTPQ comprises the 3-path swing $\{a[p_1] \approx a[p_2], b[p_2] \approx b[p_3]\}$. One can see that in every embedding of Q_2 into a database, the elements a, b, and c appear in a path of the database, even though these nodes together cannot be derived in any PP of Q_2. As a consequence a PTPQ that involves all three nodes a, b and c in one PP might contain Q_2 even though no homomorphism exists from this PTPQ to the full form of Q_2. This is the case of PTPQ Q_3 of Figure 5.3 which does not have a homomorphism to the full form of PTPQ Q_2 even though, as proved in example 5.3.2, $Q_2 \subseteq Q_3$.*

Multiple 3-path swings involving the same PPs can force more nodes from different PPs to be embedded to the same path of a database.

Example 5.4.2. *Consider the PTPQ Q_4 of Figure 5.7(a). This PTPQ is in full form. Clearly, no precedence relationship can be derived between a and b and between b and*

c in PP p_2. This PTPQ comprises two 3-path swings $\{a[p_1] \approx a[p_2],\ b[p_2] \approx b[p_3]\}$ *and* $\{c[p_1] \approx c[p_2],\ b[p_2] \approx b[p_3]\}$. *One can see that in every embedding of* Q_4 *into a database, the elements a, b, c and d and the precedence relationship* $d \Rightarrow e$ *appear in a path of the database, even though these nodes and precedence relationships together cannot be derived in any PP of* Q_4. *For this reason, PTPQ* Q_5 *shown in Figure 5.7(b) contains* Q_4. *However, there is no homomorphism from* Q_5 *to* Q_4. □

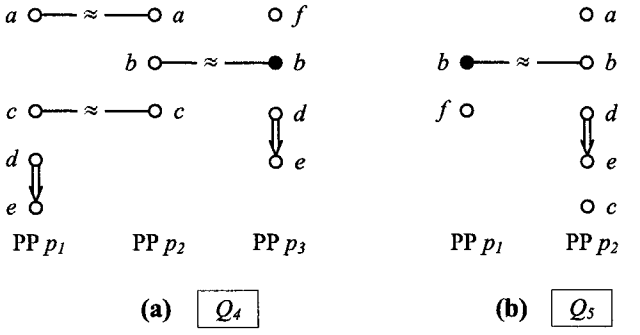

Figure 5.7 (a) PTPQ Q_4, a PTPQ with two 3-path swings, (b) PTPQ Q_5.

A similar phenomenon appears when two node sharing expressions $a[p_1] \approx a[p_2]$ and $b[p_1] \approx b[p_2]$ appear in a PTPQ Q along with the "chains" of child precedence relationships $a[p_1] \rightarrow a_1[p_1]$, $a_1[p_1] \rightarrow a_2[p_1]$, ..., $a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2]$, $b_1[p_2] \rightarrow b_2[p_2]$, ..., $b_{l-1}[p_2] \rightarrow b_l[p_2]$, when no precedence relationship can be derived between $a[p_2]$ and $b[p_2]$. An example of such a PTPQ is shown in Figure 5.8(a). Then, every tree in which there is an embedding of Q comprises a path that satisfies the child precedence relationships $a[p_1] \rightarrow a_1[p_1]$, $a_1[p_1] \rightarrow a_2[p_1]$, ..., $a_{k-1}[p_1] \rightarrow a_k[p_1]$, $b[p_2] \rightarrow b_1[p_2]$, $b_1[p_2] \rightarrow b_2[p_2]$, ..., $b_{l-1}[p_2] \rightarrow b_l[p_2]$ even if these child precedence relationships together cannot be derived in any PP of Q. We call such a set $\{a[p_1] \approx a[p_2],\ b[p_1] \approx b[p_2]\}$ of two node sharing expressions a *2-path swing.*

Example 5.4.3. *Consider the PTPQ* Q_6 *of Figure 5.8(a). This PTPQ is in full form and contains the 2-path swing* $\{a[p_1] \approx a[p_2],\ b[p_1] \approx b[p_2]\}$. *Consider also the*

PTPQ Q_7 of Figure 5.8(b). There is no homomorphism from Q_7 to Q_6. However, one can see that for every embedding of Q_6 to a database, there is an embedding of Q_7 to the same database. Therefore, $Q_6 \subseteq Q_7$. □

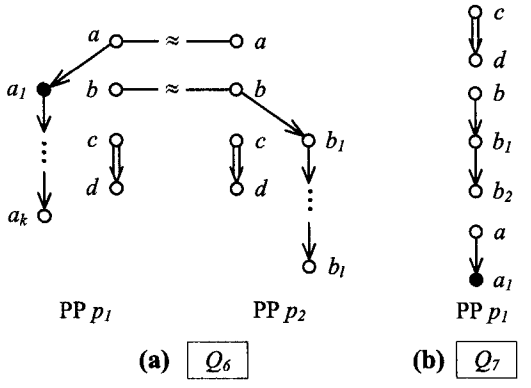

Figure 5.8 (a) PTPQ Q_6, a PTPQ with a 2-path swing, (b) PTPQ Q_7.

5.4.1 A Subclass of PTPQs

We now define a class $\mathcal{C}$ of PTPQs whose 3-path and 2-path swings appear in a symmetric way.

Definition 5.4.1. *Let $\mathcal{C}$ be the class of PTPQs Q such that: (a) if the full form of Q contains the 3-path swing $a[p_1] \approx a[p_2]$ and $b[p_2] \approx b[p_3]$, then it also contains symmetrically the 3-path swing $a[p_2] \approx a[p_3]$ and $b[p_1] \approx b[p_2]$, and (b) if the full form of Q contains the 2-path swing $a[p_1] \approx a[p_2]$, $b[p_1] \approx b[p_2]$, and the chains of child precedence relationships $a[p_1] \rightarrow a_1[p_1]$, $a_1[p_1] \rightarrow a_2[p_1]$, ..., $a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2]$, $b_1[p_2] \rightarrow b_2[p_2]$, ..., $b_{l-1}[p_2] \rightarrow b_l[p_2]$, $k, l \geq 0$, then it also symmetrically contains the chains $a[p_2] \rightarrow a_1[p_2]$, $a_1[p_2] \rightarrow a_2[p_2]$, ..., $a_{k-1}[p_2] \rightarrow a_k[p_2]$ and $b[p_1] \rightarrow b_1[p_1]$, $b_1[p_1] \rightarrow b_2[p_1]$, ..., $b_{l-1}[p_1] \rightarrow b_l[p_1]$, and the node sharing expressions $a_1[p_1] \approx a_1[p_2]$, ..., $a_k[p_1] \approx a_k[p_2]$, and $b_1[p_1] \approx b_1[p_2]$, ..., $b_l[p_1] \approx b_l[p_2]$.* □

Clearly, class $\mathcal{C}$ comprises all TPQs. However, it also comprises queries which are not TPQs as it contains PTPQs that involve two nodes in the same path with no derived precedence relationship between them.

The next theorem shows that for a PTPQ Q in $\mathcal{C}$, the existence of a homomorphism is a necessary condition for Q to be contained in another PTPQ.

Theorem 5.4.1. *Let Q be a PTPQ in full form from class $\mathcal{C}$ and Q' be a PTPQ. Then, $Q \subseteq Q'$ if and only if there is a homomorphism from Q' to Q.* □

Proof: The "if part" is straightforward. We show now the"only if part". Without loss of generality we assume that for any two PPs p_1 and p_k in Q, there is a sequence of node sharing expressions $a_1[p_1] \approx a_1[p_2], \ldots, a_{k-1}[p_{k-1}] \approx a_{k-1}[p_k]$ that starts in a node of P_1 and ends in a node of P_k (if this is not the case, we can consider "clusters" of PPs of Q that satisfy this property). Let $\mathcal{T}$ be the set of cTPQs constructed by adding descendant precedence relationships to Q so that in every PP of Q, every node that does not participate in a node sharing expression lies below a node that participates in a node sharing expression. This is feasible with any satisfiable PTPQ. However, because $Q \in \mathcal{C}$, given a cTPQ U of Q: (a) every path p of U comprises exactly the same nodes as the corresponding partial path p in Q, and (b) for every node sharing expression $a[p_1] \approx a[p_2]$ (that is, for any node a occurring in two paths p_1 and p_2 in the tree like form) in U, there is a node sharing expression $a[p_1] \approx a[p_2]$ in Q. Since $Q \subseteq Q'$ there is a homomorphism from Q' to each one of the cTPQs in $\mathcal{C}$. Because of the properties of the cTPQs in $\mathcal{C}$ there is a 1-1 mapping N from the nodes of each cTPQ U in $\mathcal{C}$ to Q that preserves the same path constraints and the node sharing expressions and does not violate the precedence relationships. Therefore, there is a 1-1 mapping from Q' to Q which preserves the same path constraints and the node sharing expressions and does not violate the precedence relationships. Let's assume that none of these mappings preserves the precedence relationships (that is, none of them is a homomorphism). Consider a PP p' of Q'. It there is no mapping that maps

all precedence relationships of p' to a PP in Q, for each PP p of Q which is the image if p' under some mapping and for each precedence relationship $A[p'] \Rightarrow B[p']$ in P' we add to P the precedence relationship $B[p] \Rightarrow A[p]$ if the precedence relationship $A[p] \Rightarrow B[p]$ does not exist in P and compute the full form of Q. Clearly, Q' will not have any homomorphism to any of the cTPQs of the resulting query (which are also cTPQs of Q) contradicting our assumption that $Q \subseteq Q'$. Otherwise, for every mapping that maps all precedence relationships of p' to Q, there is another PP p'' such that the image p_0 of p'' under this mapping does not satisfy all the precedence relationships of p''. We consider each one of these mappings in turn. For each precedence relationship $A[p''] \Rightarrow B[p'']$ in p'' we add to p_0 the precedence relationship $B[p_0] \Rightarrow A[p_0]$ if the precedence relationship $A[p_0] \Rightarrow B[p_0]$ does not exist in P_0 and compute the full form of Q. We repeat this process with p' and another mapping until all mappings are considered. Clearly, Q' will not have any homomorphism to to any of the cTPQs of the resulting query (which are also cTPQs of Q) contradicting our assumption that $Q \subseteq Q'$. Therefore, here is a homomorphism from Q' to Q. □

In [3, 33] it is shown that the containment of tree-pattern queries involving only branching and child and descendant relationships can be fully characterized by the existence of a homomorphism between the two queries. The previous theorem confirms this result since these tree-pattern queries can be represented by PTPQs from class $\mathcal{C}$.

5.5 A Heuristic Approach for Checking PTPQ Containment

As we saw in the previous section, we might fail to detect the containment of a PTPQ Q_1 in another PTPQ Q_2 based on homomorphisms. Even if Q_1 is contained in Q_2, there might be a PP in Q_2 that contains precedence relationships and is involved in

node sharing expressions which together cannot be mapped through a homomorphism to the precedence relationships and node sharing expressions of a PP of Q_1. Such a homomorphism might not exist even if all possible derived precedence relationships and node sharing expressions are equivalently added to Q_1 by computing its full form. The main idea of our heuristic approach consists in computing additional PPs, called *virtual* PPs, that contain precedence relationships from two PPs. The virtual PPs along with node sharing expressions are appropriately added to Q_1 to produce PTPQs, called *adjustments* of Q_1. It is important to note that an adjustment of Q_1 is equivalent to Q_1. Since an adjustment of Q_1 has new PPs, Q_2 has increased chances to have a homomorphism into it. This increases the possibility for detecting the containment of Q_1 into Q_2 based on the existence of a homomorphism.

5.5.1 Adjustment of a PTPQ with a Virtual PP

We define two types of virtual PPs which are based on 3-path and 2-path swings.

Definition 5.5.1. *Let Q be a PTPQ that comprises a 3-path swing $\{a[p_1] \approx a[p_2],\ b[p_2] \approx b[p_3]\}$. A* virtual PP *for p_2 w.r.t. p_1 and p_3 in Q is a PP (set of precedence relationships) that comprises: (a) a precedence relationship $r \Rightarrow a$ for every node a participating in a swing involving PPs p_1, p_2 and p_3, and (b) all precedence relationships that are common to p_1 and p_3.* □

The virtual PP v comprises precedence relationships from two different PPs.

Example 5.5.1. *Consider query Q_2' of Figure 5.5 (which is the full form of query Q_2 of Figure 5.2). Figure 5.9(a) shows the virtual PP v of PP p_5 w.r.t. PPs p_4 and p_6 in Q_2'.*

Figure 5.10(a) shows the virtual PP v_1 of PP p_2 w.r.t. p_1 and p_3 of the PTPQ Q_4 of Figure 5.7(a). Note that besides the three nodes a, b, and c, v_1 includes also the precedence relationship $d \Rightarrow e$ which is common in PPs p_1 and p_3. □

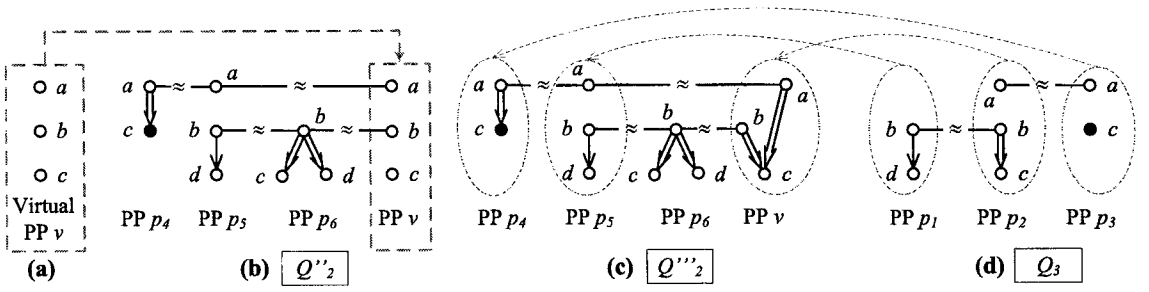

Figure 5.9 (a) The virtual PP v of p_5 w.r.t. p_4 and p_6 in Q_2, (b) The adjustment Q_2'' of Q_2' with v, (c) the full form Q_2''' of Q_2'', (d) PTPQ Q_3, and an "outline" of a homomorphism from Q_3 to Q_2'''.

We use the concept of virtual PP to define the adjustment of the query.

Definition 5.5.2. *Let v be a virtual PP for PP p_2 w.r.t. PPs p_1 and p_3 in a PTPQ Q. The* adjustment *of Q with v is a query Q' such that: (a) Q' contains all the PPs and node sharing expressions of Q, (b) Q' contains the PP v with a name (say v) that does not occur in Q, (c) for every node sharing expression $a[p_i] \approx a[p_j]$, $i, j \in [1, 3]$, Q' contains the node sharing expressions $a[v] \approx a[p_i]$ and $a[v] \approx a[p_j]$, and (d) there is no homomorphism from Q' to Q.* □

Note that condition (d) in the definition above guarantees that the adjustment Q' of Q with v exists only if Q' contains constructs (precedence relationships and node sharing expressions) that together do not appear in Q.

Example 5.5.2. *Figure 5.9(b) shows the adjustment Q_2'' of query Q_2' of Figure 5.5 with the virtual PP v shown in Figure 5.9(a). The full form Q_2''' of Q_2'' is shown in Figure 5.9(c). Figure 5.9(d) redraws query Q_3 of Figure 5.3. As proven in example 5.3.2, $Q_2 \subseteq Q_3$ (and $Q_2' \subseteq Q_3$ since $Q_2 \equiv Q'2$). Also, as stated in Section 5.3 (and one can easily check), there is no homomorphism from Q_3 to Q_2'. However, Figures 5.9(c) and (d), show a homomorphism from Q_3 to the Q_2''' (the full form of the adjustment of Q_2'. We prove later in this section that $Q_2'' \equiv Q_2'$ (and of course the same holds for the full form Q_2''' of Q_2''). Therefore, using the concept of adjustment*

of a PTPQ with a virtual PP, the containment of Q_2 into Q_3 can be detected based on the existence of a homomorphism. □

The same observation can be made in a more complex context as the next example shows.

Example 5.5.3. *Figure 5.10(b) shows the adjustment, Q'_4, of query Q_4 of Figure 5.7(a) with the virtual PP v_1 shown in Figure 5.10(a). As shown in example 5.4.2, the PTPQ Q_5 of Figure 5.7(b) contains the PTPQ Q_4 of Figure 5.7(a), and there is no homomorphism from Q_5 to Q_4. One can easily see that there is a homomorphism from Q_5 to Q'_4. As we show below $Q'_4 \equiv Q_4$. Therefore, using the adjustment of Q_4 we can detect PTPQ containment based on the existence of a homomorphism.* □

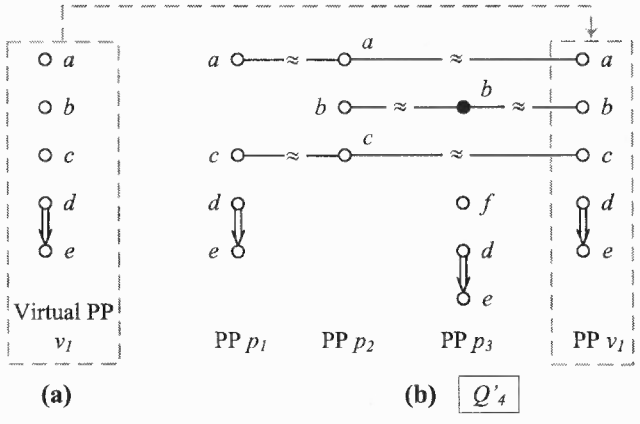

Figure 5.10 (a) The virtual PP v of p_2 w.r.t. p_1 and p_3 in Q_4, (b) The adjustment Q'_4 of Q_4 with v.

Similarly to 3-path swings, 2-path swings can be used to define virtual paths:

Definition 5.5.3. *Let Q be a PTPQ that comprises a 2-path swing $\{a[p_1] \approx a[p_2],\ b[p_1] \approx b[p_2]\}$ in Q. Let also $a[p_1] \rightarrow a_1[p_1],\ a_1[p_1] \rightarrow a_2[p_1],\ \ldots,\ a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2],\ b_1[p_2] \rightarrow b_2[p_2],\ \ldots,\ a_{l-1}[p_2] \rightarrow a_l[p_2]$ be the chains of child precedence relationships attached to $a[p_1]$ and $b[p_2]$. A* virtual PP *for p_1 and p_2 in Q is a PP (set of precedence relationships) that comprises: (a) the precedence relationships*

$a \rightarrow a_1$, $a_1 \rightarrow a_2$, ..., $a_{k-1} \rightarrow a_k$ and $b \rightarrow b_1$, $b_1 \rightarrow b_2$, ..., $a_{l-1} \rightarrow a_l$, and (b) all precedence relationships that are common to p_1 and p_2. □

A PTPQ can be adjusted with virtual paths which are based on 2-path swings.

Definition 5.5.4. *Let v be a virtual PP for PPs p_1 and p_2 in a PTPQ Q. The* adjustment *of Q with v is a query Q' such that: (a) Q' contains all the PPs and node sharing expressions of Q, (b) Q' contains the PP v with a name (say v) that does occur in Q, (c) for every node sharing expression $a[p_1] \approx a[p_2]$, Q' contains the node sharing expressions $a[v] \approx a[p_1]$ and $a[v] \approx a[p_2]$, and (d) there is no homomorphism from Q' to Q.* □

Example 5.5.4. *Figure 5.11(b) shows the adjustment, Q'_6, of query Q_6 of Figure 5.8(a) with the virtual PP v_1 shown in Figure 5.11(a). As shown in example 5.4.3, the PTPQ Q_5 of Figure 5.7(b) contains the PTPQ Q_4 of Figure 5.7(a). However, there is no homomorphism from Q_7 to Q_6. One can easily see that there is a homomorphism from Q_7 to Q'_6. As we show below $Q'_6 \equiv Q_6$. Therefore, also for 2-path swings, using the adjustment of a PTPQ we can detect containment based on the existence of a homomorphism.* □

We can now show the following theorem for the adjustment of a PTPQ with a virtual PP.

Theorem 5.5.1. *Let v be a virtual PP of a PTPQ Q (based on a 3-path or a 2-path swing). Let also Q' be the adjustment of Q with v. Then, Q' is equivalent to Q.* □

Proof: Clearly $Q' \subseteq Q$. We now show that $Q \subseteq Q'$. Let's assume that v is a virtual PP for PP p_2 w.r.t. PPs p_1 and p_3 in Q. In any cTPQ U of Q, there is a total order for the node sharing expressions of Q involving any two of the PPs p_1, p_2 and p_3 induced by the precedence relationships between nodes in the PPs p_1, p_2 and p_3 of U. This implies that at least one of p_1 or p_3 will comprise precedence relationships $r \Rightarrow a$ for

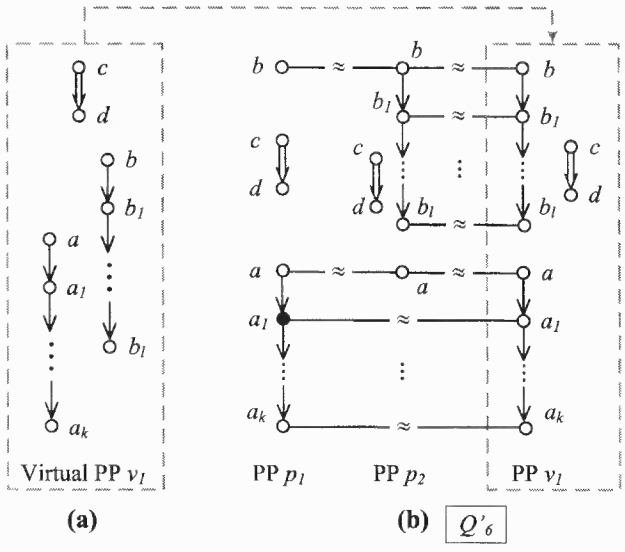

Figure 5.11 (a) The virtual PP v_1 of p_1 and p_2 in Q_6, (b) The adjustment Q'_6 of Q_6 with v.

all the nodes a participating in the node sharing expressions of Q involving any two of the PPs p_1, p_2 and p_3 along with all the precedence relationships that are common to p_1 and p_3. Therefore, all the precedence relationships of the virtual PP v in Q' can be mapped to the cTPQ U and the same holds for the node sharing expressions involving v in Q'. This means that there is a homomorphism from Q' to U. Since Q' can be mapped to every cTPQ of Q through a homomorphism, by Theorem 5.3.1, $Q \subseteq Q'$. We deal with the case of a virtual PP v which is based on a 2-path swing in a similar way. □

Since the adjustment of a PTPQ Q with a virtual PP is equivalent to Q it can be used instead of Q when checking containment of Q in another PTPQ Q'. Since the adjustment of Q with a virtual PP has an additional PP with "new" combinations of elements and precedence relationships, it increases the possibility of the existence of a homomorphism from Q' to Q when Q is contained into Q'.

5.5.2 A Heuristic Using the Adjustment of a PTPQ

A PTPQ Q might have multiple virtual PPs based on different swings in Q. Since the full form of Q can only add precedence relationships and node sharing expressions to Q, putting Q in full form can only increase the number of precedence relationships in the virtual PPs of Q. Suppose that we need to check the containment of Q into another PTPQ Q_1. Our first heuristic approach computes the full form Q' of Q, identifies all the virtual PPs v of Q', and iteratively computes the adjustment of Q' with v. Clearly, the PTPQ Q_a resulting by this process is unique up to homomorphism, independent of the order of computation of the adjustments. PTPQ Q_a is called *adjustment* of Q. The formal process is shown in Figure 5.12.

Input: a PTPQ Q
Output: the adjustment of Q.

Compute the full form Q' of Q.
$Q_t := Q'$.
for each virtual PP v of Q' /* virtual PPs can be
    based on 3-path or 2-path swings of Q'*/
    create the adjustment Q_v of Q_t with v.
    if Q_v exists then $Q_t := Q_v$
compute and return the full form of Q_t.

Figure 5.12 Computation of the adjustment Q_a of a PTPQ Q.

Based on Theorem 5.5.1, Q_a is equivalent to Q. Our first heuristic approach checks containment of Q into Q_1 by checking the existence of a homomorphism from Q_1 to Q_a. Clearly, this approach is sound but not complete: if there is a homomorphism from Q_1 to Q_a, $Q \subseteq Q_1$. However, it is possible that $Q \subseteq Q_1$ and there is no homomorphism from Q_1 to Q_a. As shown in the next example, this heuristic can detect the containment of a PTPQ Q into a PTPQ Q_1 even when none of the PPs of Q_1 can be mapped to (the full form of) Q trough a homomorphism.

Example 5.5.5. *Consider the PTPQs Q_8 and Q_9 of Figures 5.13(a) and (c) respectively. PTPQ Q_8 is in full form. Clearly, there is no homomorphism from Q_9 to Q_8. Figure 5.13(b) shows the adjustment Q_{8a} of PTPQ Q_8 which comprises two virtual PPs v_1 and v_2. Figures 5.13(b) and (c) also outline a homomorphism from Q_9 to Q_{8a}. This proves that $Q_8 \subseteq Q_9$. Note that the detection of this containment through a homomorphism became possible only because the adjustment of a PTPQ was used.* □

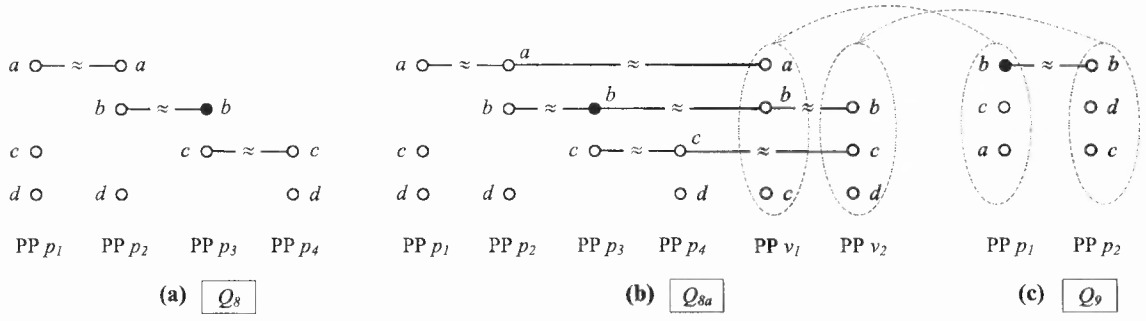

Figure 5.13 (a) PTPQ Q_8, (b) The adjustment Q_{8a} of Q_8, (c) PTPQ Q_9.

5.5.3 A Heuristic Using the Complete Adjustment of a PTPQ

Computing adjustments of a query Q with virtual PPs can define new virtual PPs in the resulting query that does not exist in Q. These "new" virtual PPs v can be used to compute adjustments of the resulting query with v until a fixed point is reached. This process is shown in Figure 5.14.

The resulting query is unique up to homomorphism It is called *complete adjustment* of Q, and is denoted Q_{ca}. Based on Theorem 5.5.1, Q_{ca} is equivalent to Q.

Our second heuristic approach checks containment of Q into a PTPQ Q_1 by checking the existence of a homomorphism from Q_1 to Q_{ca}. Since Q_{ca} can only have additional PPs and node sharing expressions compared to Q_a, there might exist a homomorphism from Q_1 to Q_{ca} even when no homomorphism exists from Q_1 to Q_a.

Input: a PTPQ Q
Output: the complete adjustment of Q.

compute the full form Q' of Q
$Q_t := Q'$
compute the set V of the virtual PPs of Q_t
repeat
    $Q_f := Q_t$
    for each $v \in V$
        create the adjustment Q_v of Q_t with v
        if Q_v exists then $Q_t := Q_v$
    compute the full form Q' of Q_t
    $Q_t := Q'$
    let N be the set of newly added PPs
    compute the set V of all virtual PPs that
    are based on swings involving at least one
    PP from N
until $Q_f = Q_t$
return Q_f

Figure 5.14 Computation of the complete adjustment Q_{ca} of a PTPQ Q.

That is, the second heuristic has increased chances to detect containment through the detection of a homomorphism compared to the first one. Like the first heuristic, it is sound but not complete.

Example 5.5.6. *Consider the PTPQs Q_{10} of Figure 5.15(b). Clearly, there is no homomorphism from Q_{10} to the PTPQ Q_8 of Figure 5.13(a) which is in full form. One can also see that here is no homomorphism from Q_{10} to the adjustment Q_{8a} of Q_8 shown in Figure 5.13(b). However, there is no homomorphism from Q_{10} to the complete adjustment Q_{8ca} of Q_8 shown in Figure 5.15(a). This proves that $Q_8 \subseteq Q_{10}$. The detection of this containment through a homomorphism became possible only because the second heuristic approach was employed.* □

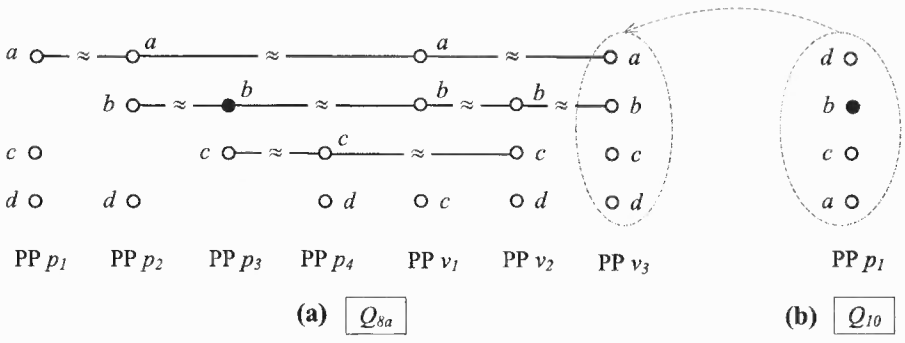

Figure 5.15 (a) the complete adjustment Q_{8ca} of Q_8, (b) PTPQ Q_{10}.

5.6 Experimental Evaluation

The implementation and execution of the experiments were conducted by our colleagues from the National Technical University of Athens, S. Souldatos and T. Dalamagas, who collaborated on the work presented in this chapter.

To study the effectiveness of our PTPQ containment checking techniques, we ran a comprehensive set of experiments. Checking PTPQ containment is expected to be time consuming for queries that involve a large number of component TPQs. However, our experimental evaluation shows that the heuristic approaches for checking PTPQ containment can save a considerable amount of time, while maintaining high accuracy.

Setup. We ran our experiments on a dedicated Linux PC (AMD Sempron 2600+) with $2GB$ of RAM. The reported values are the average of repeated measurements. Specifically, for every measure point, 100 pairs of queries were generated. For all pairs (Q_1, Q_2) of PTPQs used for containment check, $Q_1 \subseteq Q_2$, but there is no homomorphism from Q_2 to Q_1 due to the existence of 2-path or 3-path swings.

Experiments. In our experiments, we compared the execution time and the accuracy for PTPQ containment check among the following cases: (a) checking the existence of a homomorphism from Q_2 to Q_1 (Hom), (b) checking the existence of a homomorphism from Q_2 to the adjustment of Q_1 ($ComAdj$), (c) checking the ex-

istence of a homomorphism from Q_2 to the complete adjustment of Q_1 (Adj), and (d) checking the existence of a homomorphism from Q_2 to all component TPQs of Q_1 ($ComTPQs$). We tested the impact of the size and the density of the PTPQs on the execution time and on the accuracy for containment check in the above cases for different structures of the PTPQs. The accuracy of a technique is defined as the percentage of containment detections using this technique. Below, we present the detailed results.

Execution time and accuracy varying the size of queries. We measured the execution time and the accuracy for checking PTPQ containment varying the number of elements in the queries for different numbers of PPs in the queries. In Figures 5.16 and 5.17, we present the results obtained for queries with 5 to 7 elements where the number of PPs ranges between 2 and 5. All queries include the minimum number of swings that involves all PPs. Queries with 2 PPs include one 2-path swing, queries with 3 PPs include one 3-path swing, queries with 4 PPs include two 3-path swings, and queries with 5 PPs include three 3-path swings.

For a growing number of elements in the queries, the execution time of the homomorphism existence check (Hom) is almost unaffected. However, it increases as the number of PPs goes up because of the raise in the number of matchings between the PPs of the involved queries that need to be examined.

The execution time of PTPQ containment check ($ComTPQs$) goes up as the number of elements or PPs in the queries increases. The reason is that a larger number of complete TPQs are generated and, then, examined in the containment check.

Our heuristic techniques clearly improves $ComTPQs$ check. The execution time of Adj and $ComAdj$ is not generally affected by the number of elements in the queries. However, similar to Hom, the execution time of Adj and $ComAdj$ increases as the number of PPs goes up, because of the raise in the number of matchings

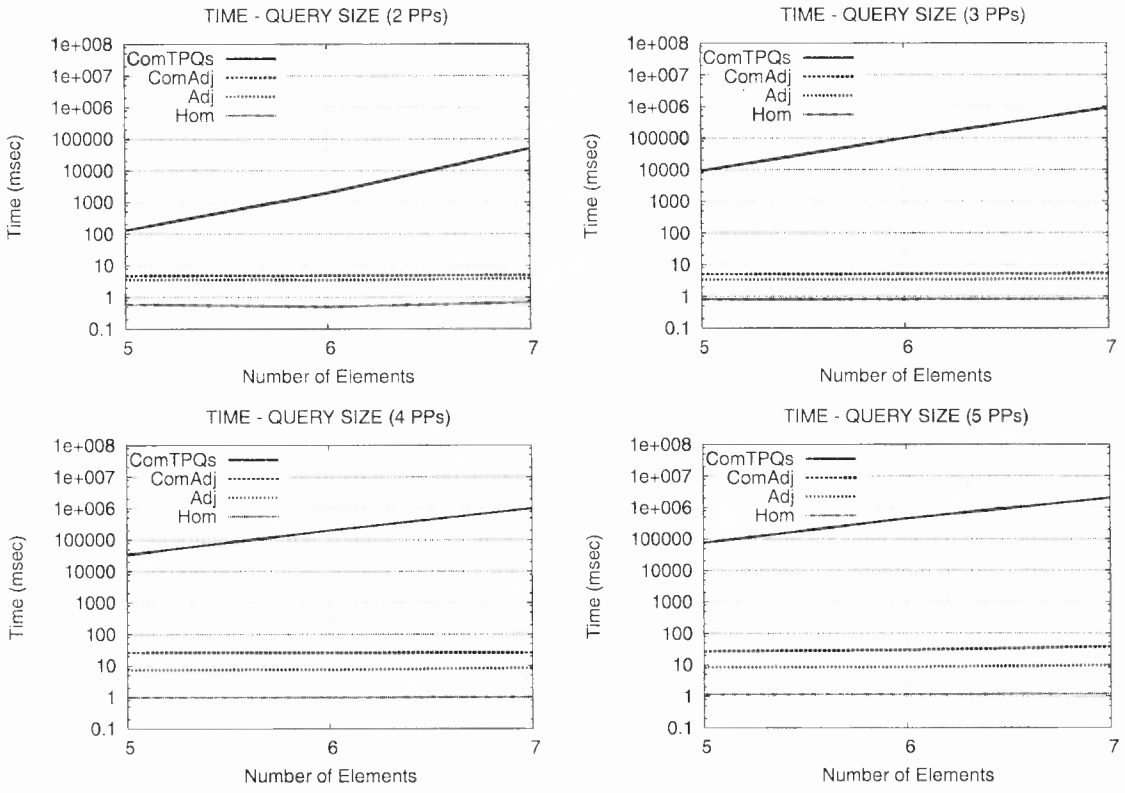

Figure 5.16 Execution time for checking PTPQ containment varying the size of the queries.

between the PPs of the involved queries that need to be examined.

The accuracy of *ComAdj* is higher than the accuracy of *Adj*, although the execution time of the former is not much Worse than the latter. For all measurements of this experiment, the accuracy of *ComAdj* is above 80%. Note that the accuracy percentages represent containment detection on pairs of contained PTPQs where containment cannot be detected through the existence of a homomorphism. Clearly, these percentages are much higher for random pairs of contained PTPQs since *Adj* and *ComAdj* can correctly detect containment when a homomorphism exists between the contained PTPQs.

Execution time and accuracy varying the density of swings in the queries. We measured the execution time and the accuracy for checking PTPQ containment varying the number of elements in the queries for different numbers of 3-path swings.

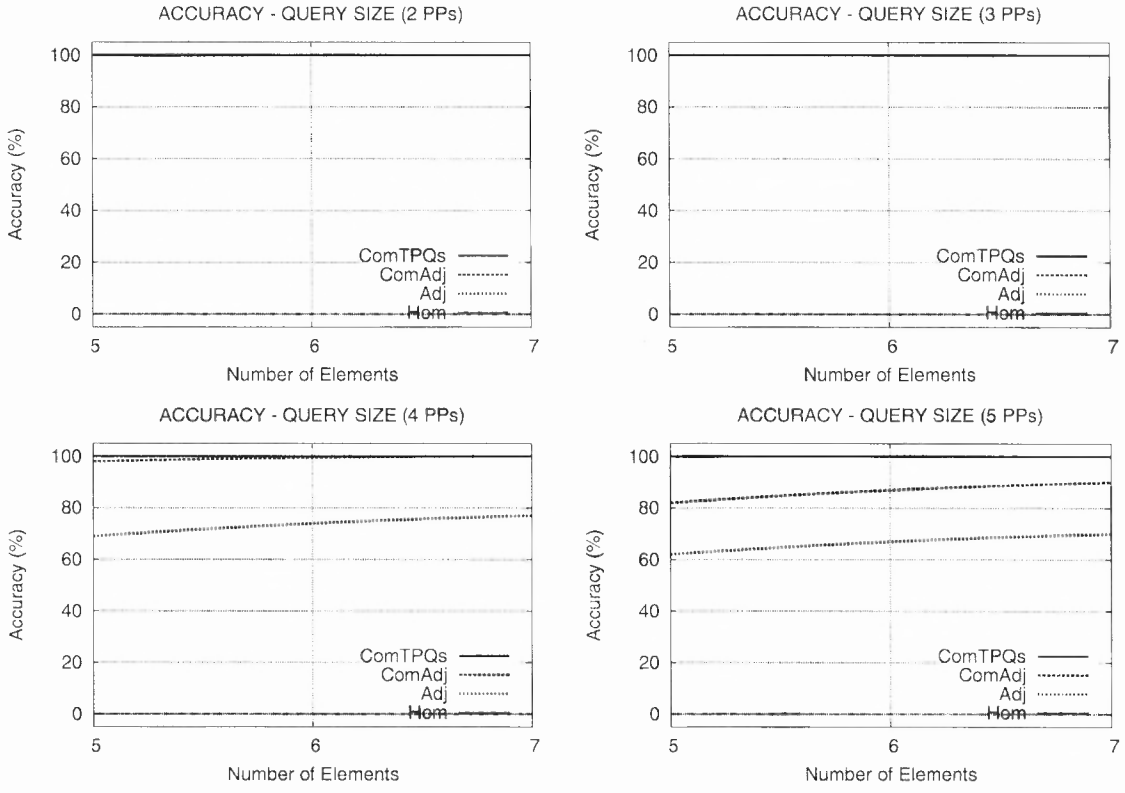

Figure 5.17 Percentage of correct answers in checking PTPQ containment varying the size of the queries.

In Figures 5.18 and 5.19, we present the results obtained for queries having 5 to 7 elements, while the number of 3-path swings ranges from 3 to 9. The number of PPs in the queries is fixed to 5.

As expected, the execution time of the homomorphism existence check (*Hom*) is not affected by the number of swings in the queries.

Similarly to the previous experiment, the execution time of containment check based on component TPQs (*ComTPQs*) goes up as the number of elements in the queries increases. It decreases slightly as the number of swings increases. The reason is that a larger number of swings involves a larger number of node sharing expressions among query nodes. Each node sharing expression restricts two query nodes to match to the same nodes on the XML tree. Thus a smaller number of component TPQs are generated and examined in the containment check.

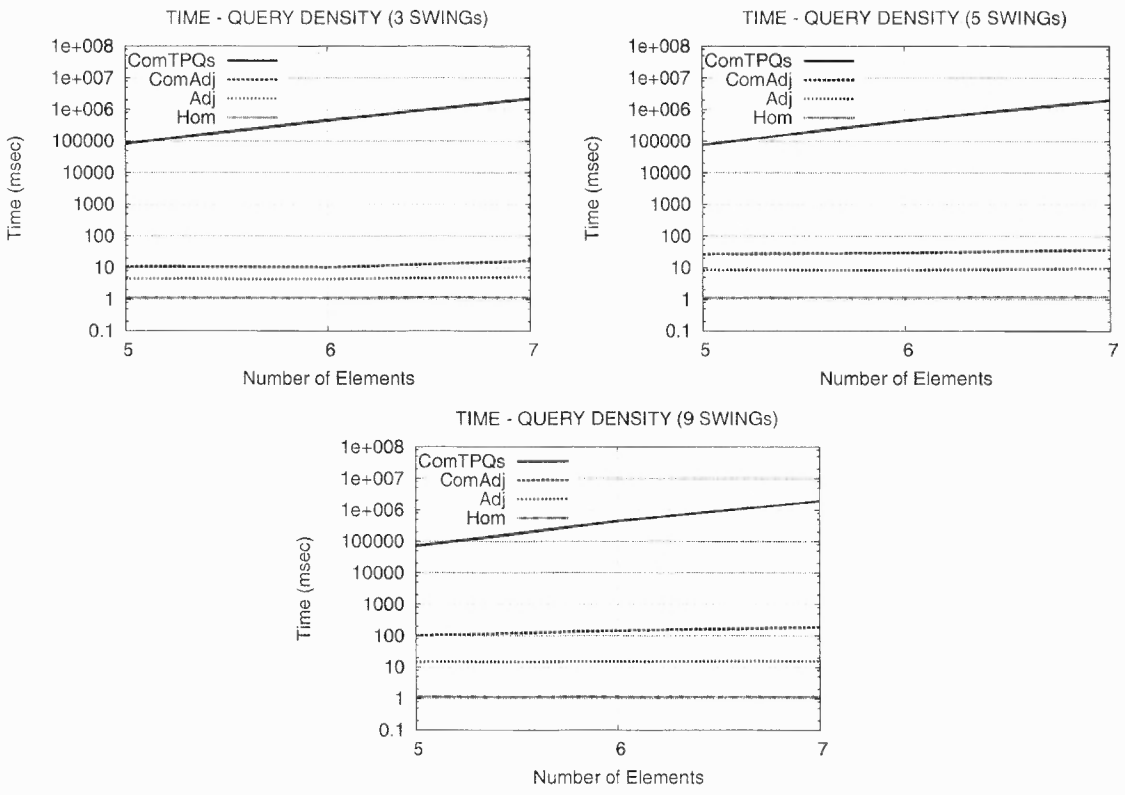

Figure 5.18 Execution time for checking PTPQ containment varying the density of swings in the queries.

Our heuristic techniques again improve the execution time of *ComTPQs*. When the number of swings in the queries goes up, the execution time of both *Adj* and *ComAdj* increases with *ComAdj* being affected more importantly. The reason is that the complete adjustment of queries with a larger number of swings generally includes a larger number of virtual paths that need to processed in the containment check.

When the number of elements in the queries increases, the execution time of *Adj* and *ComAdj* slightly increases as well. In this case, the number of swings (and consequently the number of virtual PPs) in the queries is not affected, but the increase in the number of elements slightly increases the processing time for containment check.

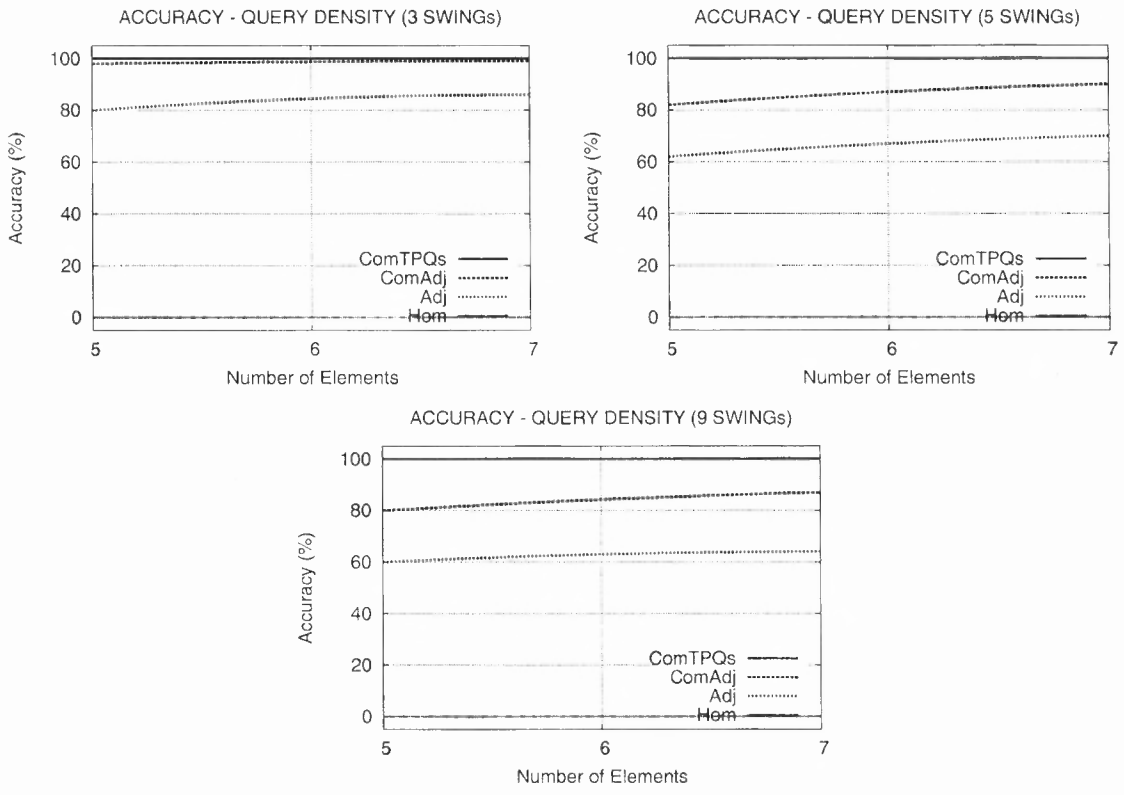

Figure 5.19 Percentage of correct answers in checking PTPQ containment varying the density of swings in the queries.

The accuracy of *ComAdj* stays above 80% in all measurements.

Remarks. Our heuristic techniques clearly improve the time of PTPQ containment check. Although these techniques are not complete, our experiments show that the accuracy of the complete adjustment heuristic is very high, while it is orders of magnitude faster than PTPQ containment check.

5.7 Conclusion

In this chapter we studied the containment problem for PTPQ in the absence of dimension graphs, and we provided necessary and sufficient conditions for PTPQ containment. We identified a subclass of PTPQs where containment can be fully

characterized by the existence of homomorphisms. We further devised a sound but not complete heuristic approaches that equivalently add additional partial paths to PTPQs. Detailed experimental evaluations of our approaches show that they greatly improve the query containment checking execution time, and that they gradually trade execution time for accuracy. These results allow their use for query processing and optimization.

CHAPTER 6

MINIMIZATION OF PARTIAL TREE-PATTERN QUERIES

6.1 The PTPQ Minimization Problem

When evaluating queries, we are interested in minimizing their evaluation cost. According to the analysis of [21, 22], the evaluation efficiency of XPath queries greatly depends on their size. Therefore, similarly to [3, 17, 54] we consider the size of XPath queries as a measure of their cost and we focus on minimizing the number of nodes in a PTPQ and the number of PPs in the PTPQ. We define in this section the concept of minimal PTPQ, we formally state the problem addressed, and we show its difficulties.

6.1.1 Minimal PTPQs

In defining minimality for PTPQs we are interested in minimizing the number of number of nodes in the PTPQ. However, in PTPQs, nodes can participate in node sharing expressions which indicate that the involved nodes are always embedded in the same XML tree node. Since these nodes essentially represent the same node, they should be counted as one node. Therefore, in defining minimal queries below we consider *equivalence classes* of nodes as these are determined by node sharing expressions: two nodes from different PPs belong to the same equivalence class iff they participate in the same node sharing expression. In the following, minimizing the number of nodes in a given query refers to minimizing the equivalence classes of nodes.

However, as the next example shows, two equivalent queries that have a minimal number of (equivalent classes of) nodes, might have a different number of PPs.

Example 6.1.1. *Consider the queries Q and Q′ of Figure 6.1. These PTPQs are equivalent and one can see that they have the same number of (equivalent classes of) nodes and there is no other equivalent PTPQ with less (equivalent classes of) nodes. However, PTPQ Q has three PPs while PTPQ Q′ has two PPs.* □

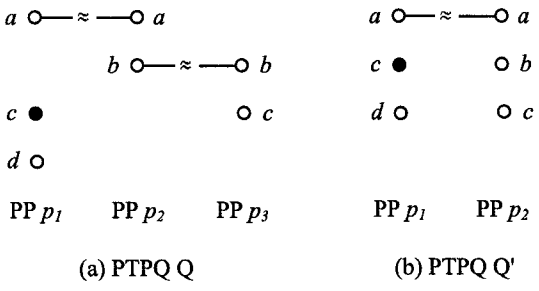

Figure 6.1 Two equivalent PTPQs.

Therefore, in defining minimal PTPQs we also take into account the number of PPs in them:

Definition 6.1.1. *A PTPQ is minimal if (a) there is no equivalent PTPQ with less (equivalent classes of) nodes, and (b) there is no equivalent PTPQ with the same number of (equivalent classes of) nodes that has less PPs.* □

Further, two nodes in different PPs may not belong to the same equivalence class unless the query is put in full form (because the relevant node sharing expression can be derived from the inference rules). Note that computing the full form of a PTPQ does not add new PPs, or new equivalence classes of nodes to the PTPQ. It can only add precedence relationships in existing PPs (which does not affect the number of equivalence classes of nodes in the PTPQ), or node sharing expressions between nodes from different PPs (which might reduce the number of equivalence classes of nodes). Therefore, we can show the following proposition.

Proposition 6.1.1. *If a PTPQ is minimal, its full form is also minimal.* □

A minimal query is of interest if it is also equivalent to a given different query.

Example 6.1.2. *Figures 6.2, 6.3 and 6.4 show three PTPQs Q and for each of them a minimal PTPQ Q' which is equivalent to Q. In Figure 6.2, Q has less nodes and less PPs than Q'.*

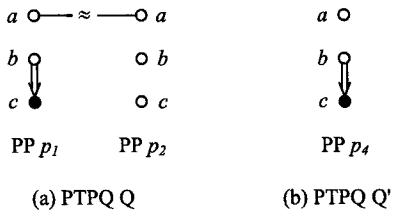

Figure 6.2 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q).

In Figures 6.3 and 6.4, Q' has less nodes than Q' and the same number of PPs than Q'.

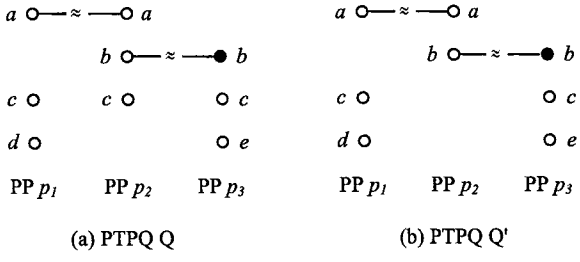

Figure 6.3 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q).

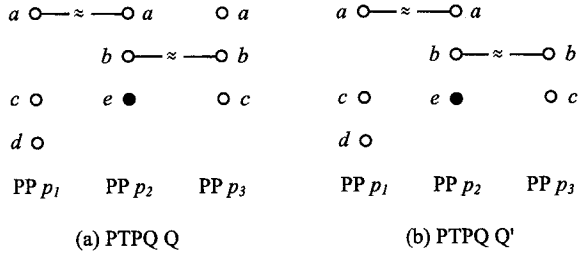

Figure 6.4 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q).

Though a tedious process, one can verify that the three queries Q' of figures 6.2, 6.3 and 6.4 are minimal and equivalent to the corresponding queries Q. □

6.1.2 Problem Addressed

We can now formally define the problem we address in this paper (called *PTPQ minimization problem*): given PTPQ Q, find a minimal PTPQ Q' which is equivalent to Q.

Before discussing the challenges involved in minimizing PTPQs we introduce below the concept of homomorphism between PTPQs.

6.1.3 Homomorphisms between PTPQs

The minimization problem for different classes of TPQs can be fully characterized through the existence of homomorphisms between TPQs [3]. In order to examine whether similar results can be obtained for PTPQs, we define use the concept of homomorphism between PTPQs defined in Definition 5.3.1.

As an example one can see that there is a homomorphism from the PTPQ Q to the PTPQ Q' of Figure 6.2 and vice versa (from Q' to Q).

6.1.4 Challenges in PTPQ Minimization

In the example of Figure 6.2 above, the equivalent minimal PTPQ Q' can be obtained by removing the "redundant" PP p_2 from Q. In general, we can define redundant parts (PPs or nodes) in a PTPQ as follows:

Definition 6.1.2. *Let Q be a PTPQ. A part (PP or node) is redundant in Q if the PTPQ resulting by removing this part from Q is equivalent to Q.*

Removing a node from a PTPQ implies that all the precedence relationships and node sharing expressions that involve this node and the other nodes in its equivalence class are also removed. Removing a PP from a PTPQ implies that all the node sharing expressions that involve this PP are also removed. Clearly, the PP p_2 of Q in Figure 6.2 is redundant because there is a homomorphism from Q the PTPQ

obtained by removing p_2 from Q (that is, PTPQ Q') which guarantees the equivalence of Q and Q'.

In the example of Figure 6.3 the minimal equivalent PTPQ Q' is obtained by removing the node c from the PP p_2 of PTPQ Q which is in full form.

One might wonder whether, for a given PTPQ, a minimal equivalent one can always be computed by removing redundant parts. The next example shows that this is not possible even if the PTPQ is in full form.

Example 6.1.3. *Consider the PTPQs Q and Q' of Figure 6.5. One can see that Q is in full form and that Q and Q' are equivalent. Clearly, Q' is minimal. Observe that the single PP of Q' is different (and in fact it has more nodes) than any PP of Q.*

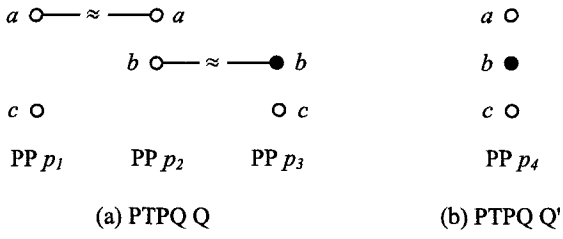

Figure 6.5 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q).

Similar observation can be made about the PTPQs Q and Q' of Figure 6.6. Q is in full form, and Q and Q' are equivalent. Clearly, Q' is minimal. Observe again that the single PP of Q' is different (and in fact it has more nodes) than any PP of Q. □

Another important question when minimizing PTPQs is whether a PTPQ has a unique equivalent minimal PTPQ. For instance, in [3] it is shown that for a TPQ involving descendant relationships, there is a minimal equivalent TPQ which is unique up to isomorphism (an isomorphism is a 1-1 homomorphism whose inverse mapping is also a homomorphism). The following example shows that unfortunately, this is not the case with PTPQs.

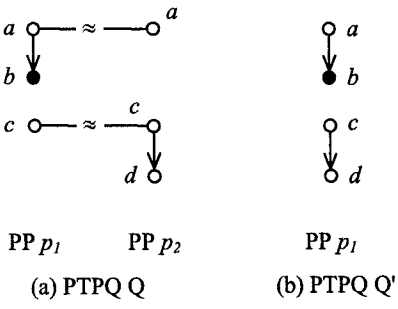

Figure 6.6 (a) PTPQ Q, (b) PTPQ Q' (minimal and equivalent to Q).

Example 6.1.4. *Consider the PTPQs Q_1 and Q_2 of Figure 6.7. One can see that Q_1 and Q_2 are equivalent and both of them are minimal. However, there is no isomorphism between Q_1 and Q_2.*

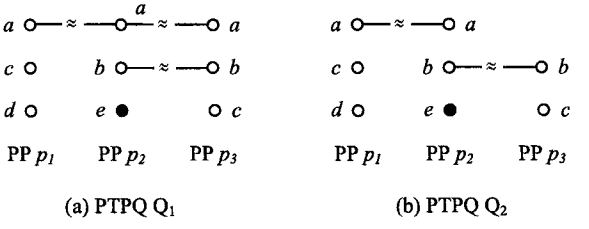

Figure 6.7 Two equivalent minimal queries.

For a more complex case consider the three PTPQs Q_1, Q_2 and Q_3 of Figure 6.8. These PTPQs are equivalent and all of them are minimal. Again, there are no pairwise isomorphisms among these queries. □

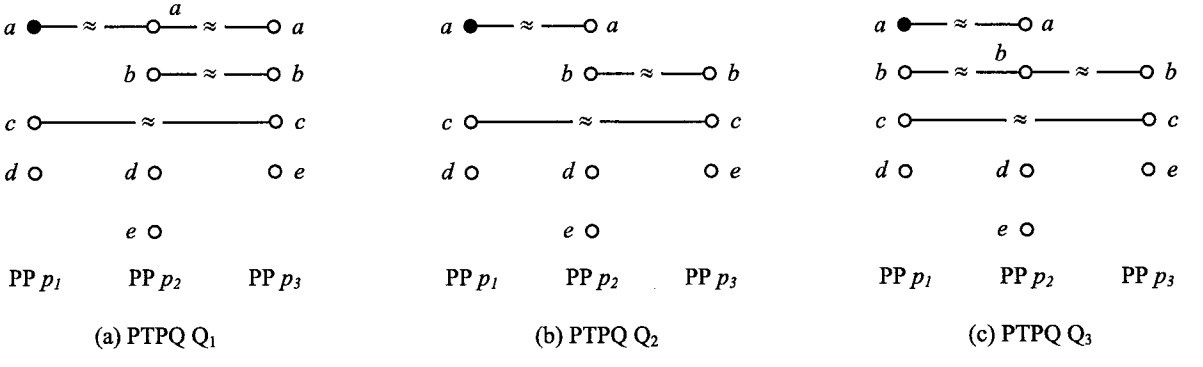

Figure 6.8 Three minimal and equivalent queries.

Therefore, a minimal equivalent PTPQ cannot be computed by identifying and removing "redundant" PPs and/or (equivalent classes) of nodes, and an equivalent

minimal PTPQ might not be unique. These facts make the PTPQ minimization problem more complex.

6.2 Minimizing a Subclass of PTPQs

In this section, we consider a subclass of PTPQs called component PTPQs, and we show that a PTPQ is equivalent to a set of component PTPQs. We then show that PTPQ containment (and equivalence) can be fully characterized through the existence of homomorphisms from PTPQs to component PTPQs. We use this result to show that minimization for component PTPQs can be achieved by removing redundant PPs identified by a homomorphism.

6.2.1 Component PTPQs

A PTPQ Q is called *component* PTPQ (or cPTPQ for short) if in the full form of Q, there is a descendant precedence relationship between any two nodes that participate in a node sharing expression and lie on the same PP. Clearly, the class of cPTPQs contains TPQs. Observe that by adding descendant precedence relationships between every two nodes that participate in node sharing expressions and lie in the same PP of a PTPQ and by taking the full form of the resulting PTPQ until no two nodes in the same PP that participate in node sharing expressions are not linked through a precedence relationship, the resulting PTPQ is an unsatisfiable PTPQ or a satisfiable cPTPQ.

A PTPQ can be associated to a set of cPTPQs. The cTPQs of a given PTPQ Q is the set of satisfiable cPTPQs resulting by this process. It is not difficult to see that if no satisfiable cPTPQ can be obtained for Q from this process Q is unsatisfiable.

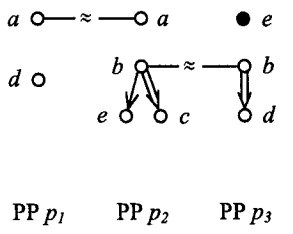

Figure 6.9 A PTPQ Q.

Consider the PTPQ Q of Figure 6.9. Figure 6.10 shows the two cPTPQs U_1 and U_2 of Q.

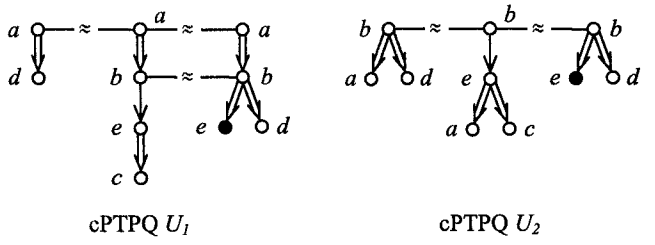

Figure 6.10 The two cPTPQs for the PTPQ Q of Figure 6.9.

The cPTPQs of a PTPQ Q can be used to compute the answer of Q as follows:

Proposition 6.2.1. *Let $\{P_1, \ldots, P_k\}$ be the set of cPTPQs of a PTPQ Q and D be a database. Let also $A, A_1, \ldots, A_k$ be the answers of $Q, P_1, \ldots, P_n$ respectively on D. Then, $A = \bigcup_{i=1,\ldots,k} A_i$.* □

The previous proposition states that the answer of a PTPQ is the union of the answers of its cPTPQs. This result generalizes a result from Chapter 5 where the answer of a PTPQ is computed from a set of TPQs called component TPQs. Clearly, the number of cPTPQs of a given PTPQ is not greater than the number of its cTPQs, and in general is expected to be much smaller. Even though the number of cPTPQs of a given query can be exponential in the number of its nodes, it is possible that for a given query the number of its cPTPQs is polynomial while the number of its cTPQs is exponential in the number of its nodes.

6.2.2 Using cPTPQs to Characterize PTPQ Containment

As shown in Chapter 5 the existence of a homomorphism from a PTPQ Q' to a PTPQ Q even though a sufficient condition is not a necessary one for the containment of Q into Q'. This is so even if the full form of Q is considered instead of Q.

We now provide necessary and sufficient conditions for PTPQ containment in terms of homomorphisms from a PTPQ to cPTPQs.

Theorem 6.2.1. *Let Q_1 and Q_2 be two PTPQs. Let also $\mathcal{U}_1$ be the set of component cPTPQs for Q_1 in full form. $Q_1 \subseteq Q_2$ iff for every component TPQ $U \in \mathcal{U}_1$, there is a homomorphism from Q_2 to U.* □

For instance, one can see that any PTPQ that contains the PTPQ Q of Figure 6.9 has a homomorphism to each one of the cPTPQs of Q shown in Figure 6.10 (which are in full form).

A consequence of Theorem 6.2.1 is that the containment for cPTPQs is fully characterized by the existence of homomorphisms provided that the contained cPTPQ is in full form. We use this result to discuss minimization for cPTPQs.

6.2.3 cPTPQ Minimization

Given a cPTPQ, a minimal equivalent one can be obtained by removing from its full form iteratively all redundant parts in any order until no more redundant parts can be removed. Note that based on the results of the previous subsection, a redundant part p in a cPTPQ Q can be identified by the existence of a homomorphism from Q to the full form of $Q - \{p\}$.

6.3 Heuristic Algorithms for PTPQ Minimization

In this section, we first elaborate on the reason PTPQs cannot be minimized using homomorphisms to identify and remove redundant parts even though this is the case with TPQs involving branching and descendant relationships [3]. We use these observations to show how a PTPQ can be equivalently put in different forms (called adjustments of the PTPQ) that gradually comprise more PPs and are more likely to be minimized through the removal of redundant parts. Finally, we exploit PTPQ adjustments to design heuristic algorithms for minimizing PTPQs.

6.3.1 Why PTPQ Minimization Cannot Be Achieved Using Homomorphisms to Identify Redundant Parts

The presence of two node sharing expressions $a[p_1] \approx a[p_2]$ and $b[p_2] \approx b[p_3]$ in a PTPQ Q when no precedence relationship can be derived between $a[p_2]$ and $b[p_2]$ forces the image of Q under any embedding to comprise a path that involves $r \Rightarrow a$ and $r \Rightarrow b$ and all relationships which are common between p_1 and p_3. This is true even though these precedence relationships together cannot be derived in any PP of Q. Such a set $\{a[p_1] \approx a[p_2],\ b[p_2] \approx b[p_3]\}$ of two node sharing expressions is called a *3-path swing* and it was described in detail in Chapter 5.

In the presence of a 3-path swing, it is possible that a PTPQ is not minimal even though it does not comprise redundant parts in which case, of course, homomorphisms cannot help us in minimizing the PTPQ. Further, in the presence of a 3-path swing, it is possible that a redundant PP exists in the PTPQ, even though no homomorphism exists from this PP to any other PP of the PTPQ. Again in this case homomorphisms fail to help us in minimizing the PTPQ.

Example 6.3.1. *Consider the PTPQ Q of Figure 6.5(a) which is in full form. PTPQ Q is not minimal (a minimal equivalent one is shown in Figure 6.5(b)). PTPQ Q comprises the 3-path swing $\{a[p_1] \approx a[p_2], b[p_2] \approx b[p_3]\}$. Consequently, the image of Q under any embedding comprises a path that involves the nodes a, b, and c. Clearly, Q does not comprise redundant parts, and therefore homomorphisms cannot be used to compute a minimal equivalent PTPQ.*

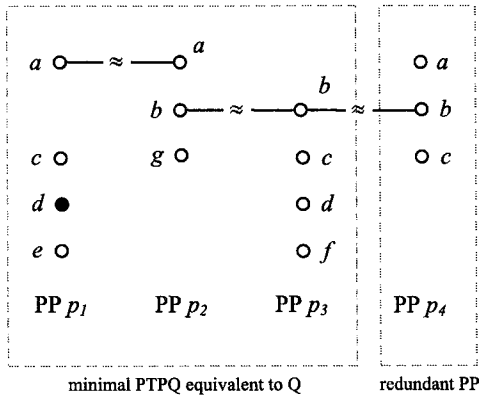

Figure 6.11 A PTPQ Q with a redundant PP.

Consider also the PTPQ Q of Figure 6.11 which again is in full form. PTPQ Q is not minimal. A minimal equivalent one can be obtained by removing PP p_4. Observe that, even though p_4 is redundant, its redundancy cannot be established using homomorphisms. There is no homomorphism from p_4 to any one of p_1, p_2, and p_3. The reason is that the 3-path swing $\{a[p_1] \approx a[p_2], b[p_2] \approx b[p_3]\}$ forces the nodes a, b, c, and d to appear in a path of the image of Q under any embedding of Q, and this makes p_4 redundant. □

A similar behavior can be observed when two node sharing expressions $a[p_1] \approx a[p_2]$ and $b[p_1] \approx b[p_2]$ appear in a PTPQ Q along with the "chains" of child precedence relationships $a[p_1] \rightarrow a_1[p_1]$, $a_1[p_1] \rightarrow a_2[p_1]$, ..., $a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2]$, $b_1[p_2] \rightarrow b_2[p_2]$, ..., $b_{l-1}[p_2] \rightarrow b_l[p_2]$, when no precedence relationship can be derived between $a[p_2]$ and $b[p_2]$. Then, every tree in which there is an embedding of Q comprises a path that satisfies the child precedence relationships

$a[p_1] \rightarrow a_1[p_1]$, $a_1[p_1] \rightarrow a_2[p_1]$, ..., $a_{k-1}[p_1] \rightarrow a_k[p_1]$, $b[p_2] \rightarrow b_1[p_2]$, $b_1[p_2] \rightarrow b_2[p_2]$, ..., $b_{l-1}[p_2] \rightarrow b_l[p_2]$ even if these child precedence relationships together cannot be derived in any PP of Q. We call such a set $\{a[p_1] \approx a[p_2], b[p_1] \approx b[p_2]\}$ of two node sharing expressions a *2-path swing*. Similarly to 3-path swings, 2-path swings are discussed in detail in Chapter 5.

An example of a 2-path swing ($\{a[p_1] \approx a[p_2], c[p_1] \approx b[p_2]\}$) is shown in the PTPQ of Figure 6.12.

As with the case of 3-path swings, homomorphisms may fail to help us in minimizing a PTPQ either because there are redundant PPs in the PTPQ that cannot be identified through homomorphisms or because, even though the PTPQ is not minimal, there are no redundant parts in which case homomorphisms cannot, of course, be of help.

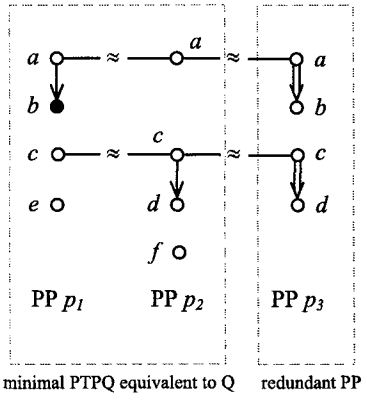

Figure 6.12 A PTPQ Q with a redundant PP.

Example 6.3.2. *Consider the PTPQ Q of Figure 6.12 which is in full form. PTPQ Q is not minimal. A minimal equivalent one can be obtained by removing PP p_3. Observe that, even though p_3 is redundant, its redundancy cannot be established using homomorphisms since there is no homomorphism from p_3 to any one of p_1 and p_2. The reason is that the 2-path swing $\{a[p_1] \approx a[p_2], c[p_1] \approx c[p_2]\}$ forces the precedence relationships $a \rightarrow b$ and $c \rightarrow d$ to appear in the same database path of the image of Q under any embedding of Q to this database, and this makes p_3 redundant.* □

6.3.2 Virtual Paths and Adjustments of PTPQs

In order to design heuristic algorithms for PTPQ minimization we use the concept of virtual path and we slightly modify the concept of PTPQ adjustment introduced in Chapter 5. We present these concepts below.

Definition 6.3.1. *Let Q be a PTPQ that comprises a 3-path swing $\{a[p_1] \approx a[p_2],\ b[p_2] \approx b[p_3]\}$. A* virtual PP *for p_2 w.r.t. p_1 and p_3 in Q is a PP (set of precedence relationships) that comprises: (a) a precedence relationship $r \Rightarrow a$ for every node a participating in a swing involving PPs p_1, p_2 and p_3, and (b) all precedence relationships that are common to p_1 and p_3.* □

Therefore, the virtual PP v comprises precedence relationships from two different PPs.

For an example of the virtual path consider the PTPQ Q of Figure 6.11. A virtual path v for PP p_2 w.r.t to PPs p_1 and p_3 of Q is shown in Figure 6.13.

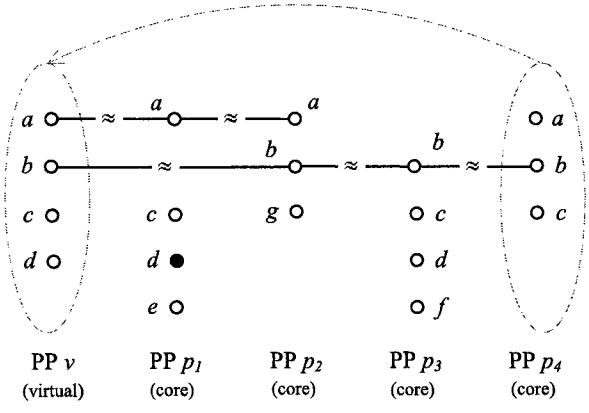

Figure 6.13 The adjustment of the PTPQ Q of Figure 6.11 with a virtual PP v.

We use the concept of virtual PP to define the adjustment of a PTPQ. For this definition, we assume that each PP in a PTPQ is marked as *core* or *virtual*.

Definition 6.3.2. *Let v be a virtual PP for PP p_2 w.r.t. PPs p_1 and p_3 in a PTPQ Q. The* adjustment *of Q with v is a PTPQ Q' such that: (a) Q' contains all the PPs of Q marked as they are marked in Q and all node sharing expressions of Q, (b) Q' contains the PP v with a name (say v) that does not occur in Q marked as virtual, (c) for every node sharing expression $a[p_i] \approx a[p_j]$, $i, j \in [1, 3]$, Q' contains the node sharing expressions $a[v] \approx a[p_i]$ and $a[v] \approx a[p_j]$, and (d) there is no homomorphism from Q' to Q, which maps v to p_1, p_2, or p_3.* □

Figure 6.13 shows the adjustment of the PTPQ Q of Figure 6.11 with the virtual path v.

Similarly to 3-path swings, 2-path swings can be used to define virtual paths:

Definition 6.3.3. *Let Q be a PTPQ that comprises a 2-path swing $\{a[p_1] \approx a[p_2],\ b[p_1] \approx b[p_2]\}$ in Q. Let also $a[p_1] \rightarrow a_1[p_1],\ a_1[p_1] \rightarrow a_2[p_1],\ \ldots,\ a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2],\ b_1[p_2] \rightarrow b_2[p_2],\ \ldots,\ b_{l-1}[p_2] \rightarrow b_l[p_2]$ be the chains of child precedence relationships attached to $a[p_1]$ and $b[p_2]$. A* virtual PP *for p_1 and p_2 in Q is a PP (set of precedence relationships) that comprises: (a) the precedence relationships $a \rightarrow a_1,\ a_1 \rightarrow a_2,\ \ldots,\ a_{k-1} \rightarrow a_k$ and $b \rightarrow b_1,\ b_1 \rightarrow b_2,\ \ldots,\ b_{l-1} \rightarrow b_l$, and (b) all precedence relationships that are common to p_1 and p_2.* □

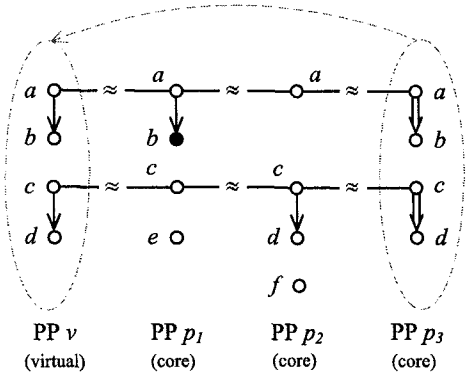

Figure 6.14 The adjustment of the PTPQ Q of Figure 6.12 with a virtual PP v.

For an example of the virtual path consider the PTPQ Q of Figure 6.12. A virtual path v for PPs p_1 and p_2 in Q is shown in Figure 6.14.

A PTPQ can be adjusted with virtual paths which are based on 2-path swings.

Definition 6.3.4. *Let v be a virtual PP for PPs p_1 and p_2 in a PTPQ Q. The* adjustment *of Q with v is a query Q' such that: (a) Q' contains all the PPs of Q marked as they are marked in Q and all node sharing expressions of Q, (b) Q' contains the PP v with a name (say v) that does not occur in Q marked as virtual, (c) for every node sharing expression $a[p_1] \approx a[p_2]$, Q' contains the node sharing expressions $a[v] \approx a[p_1]$ and $a[v] \approx a[p_2]$, and (d) there is no homomorphism from Q' to Q, which maps v to p_1 or p_2.* □

Figure 6.14 shows the adjustment of the PTPQ Q of Figure 6.12 with the virtual path v. As we shown in Theorem 5.5.1 adjustment of a PTPQ Q with a virtual PP is equivalent to Q.

6.3.3 Heuristic Algorithms for Minimizing PTPQs

Our heuristic algorithms are based on the following remarks. Consider the PTPQ of Figure 6.11, PP p_4 in Q is redundant, but Q does not have a homomorphism to the PTPQ resulting by removing PP p_4 from Q. In contrast, if we consider the adjustment Q' of Q with v shown in Figure 6.13, we can see that Q' has a homomorphism to the PTPQ resulting by removing PP p_4 from Q'. Therefore, we can remove the redundant PP p_4 and the virtual PP v from Q' to obtain a PTPQ, which has less PPs then Q and is equivalent to Q. Incidentally, this PTPQ is also minimal. Consider also the PTPQ of Figure 6.12, PP p_3 in Q is redundant, but Q does not have a homomorphism to the PTPQ resulting by removing PP p_3 from Q. In contrast, if we consider the adjustment Q' of Q with v shown in Figure 6.14, we can see that Q' has a homomorphism to the PTPQ resulting by removing PP p_3 from Q'. Therefore, we can remove the redundant PP p_3 and the virtual PP v from Q' to obtain a PTPQ equivalent to Q, which has less PPs then Q. In this case too, the new PTPQ is also minimal.

We call *adjustment* of a PTPQ Q the PTPQ Q_a obtained by adding virtual paths to the full form of Q based on all triplets or pairs of core paths that have swings and by taking the full form of the resulting PTPQ. Clearly the adjustment of the PTPQ Q is unique and equivalent to Q.

Heuristic Algorithm 1: Minimization using PTPQ adjustment.
Input: a PTPQ Q
Output: a PTPQ Q'.

Step 1
compute the adjustment Q_a of Q.

Step 2
for every PP v marked as virtual in Q_a do
  for every core PP p_i involved in the generation of v do
    if there is a homomorphism from Q_a to $Q_a - \{p_i\}$ that
    maps p_i to v and all other PPs of Q_a to themselves then
      remove p_i from Q_a and mark v as core PP

Step 3
for every core PP p_j of Q_a do
  if there is a homomorphism from Q_a to $Q_a - \{p_j\}$, then
    remove p_j from Q_a

Step 4
for every 3-path swing of Q_a involving the core PPs p_1, p_2, and p_3, with
p_2 being the axis of the swing, do
  let set S contain all precedence relationships of p_2 not involving nodes
  participating in node sharing expressions with nodes from p_1 or p_3,
    if no node in the precedence relationships in S participates in a node sharing
    expression in Q_a and all precedence relationships in S appear in both
    p_1 and p_3, then
      remove the nodes appearing in precedence relationships in S from p_2

Step 5
compute Q' by removing all virtual PPs from Q_a

Figure 6.15 Heuristic algorithm 1.

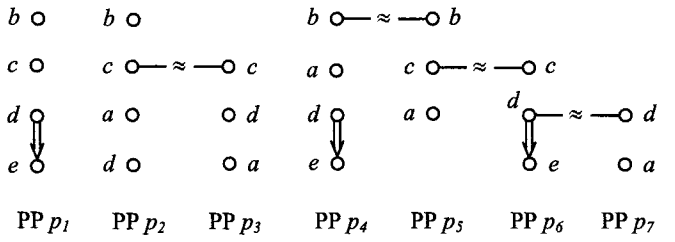

Figure 6.16 A PTPQ Q.

Example 6.3.3. *Consider the PTPQ Q of Figure 6.16. PTPQ Q is in full form. PPs p_1, p_2 and p_3 are redundant but this cannot be identified by homomorphisms. Figure 6.17 shows the adjustment of Q. Our first heuristic algorithm on Q is able to identify the PP p_1 of Q as redundant since there is a homomorphism from Q_a to $Q_a - \{p_1\}$ mapping the PP p_1 to the virtual PP and removes it from Q.* □

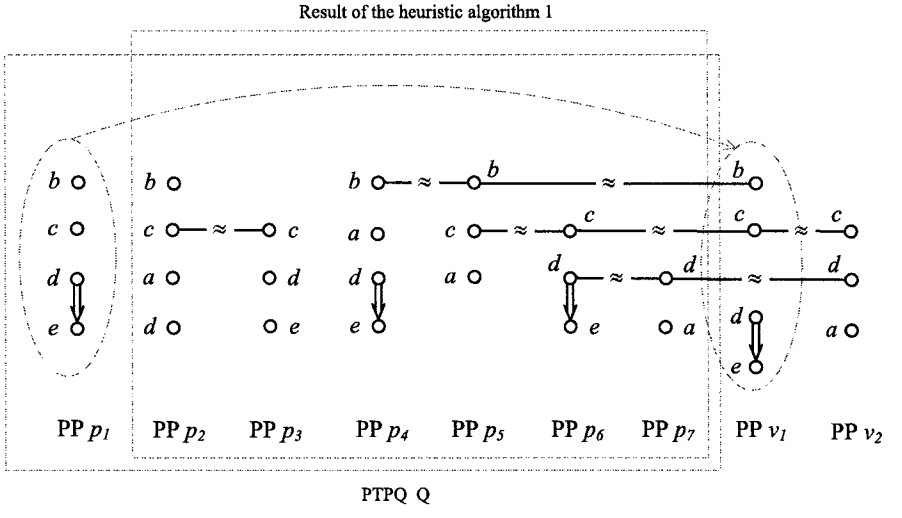

Figure 6.17 The adjustment of the PTPQ Q of Figure 6.16.

We can show the following theorem for Heuristic Algorithm 1.

Theorem 6.3.1. *Heuristic Algorithm 1 on a PTPQ Q produces a PTPQ Q' which is equivalent to Q and does not have more (equivalent classes of) nodes or PPs than Q.* □

Proof: Clearly, the algorithm does not add any additional PPs. Any time the virtual PP is marked as core, at least one other core PP is removed from the PTPQ. The fact that the PTPQ obtained by this step is equivalent to the original PTPQ can be derived from the fact that the adjustment of a PTPQ Q with virtual path is equivalent to Q. Whenever redundant PP p is removed from the current PTPQ Q in Step 3, the equivalence is guaranteed by the existence of a homomorphism from Q to Q - $\{p\}$. Finally, the nodes and precedence relationships removed by the algorithm from the axis of a 3-path swing in Step 4 appear in the virtual path, and therefore the resulting PTPQ is equivalent to the initial one. □

As shown in the Example 6.3.3, the first heuristic algorithm is able to identify and remove redundant PPs or nodes, which cannot be shown to be redundant by simply using homomorphisms on the PTPQ or the full form of the PTPQ. In this sense, the algorithm is able to produce minimal equivalent PTPQs Q' from all the PTPQs Q shown in Figures 6.3, 6.4, 6.11, and 6.12. Moreover, the first heuristic algorithm is able to minimize PTPQs, whose minimal equivalent ones cannot be obtained by removing redundant parts. For instance, the algorithm will produce minimal equivalent PTPQs Q' for the PTPQs Q of Figures 6.5 and 6.6.

Our second heuristic algorithm exploits an extension of the concept of adjustment of a PTPQ called complete adjustment: the *complete adjustment* of a PTPQ Q is defined similarly to the adjustment of Q except that in the complete adjustment virtual PPs that are defined recursively based on swings involving other virtual PPs are also added to the adjustment.

Clearly, the second heuristic algorithm consumes more time since it has to recursively compute the complete adjustment of Q instead of the "simple" adjustment but it is expected to be able to identify and remove more core PPs. A theorem similar to Theorem 6.3.1 can be shown for Heuristic Algorithm 2.

Heuristic Algorithm 2: Minimization using PTPQ complete adjustment.
Input: a PTPQ Q
Output: a PTPQ Q'.

Step 1
compute the complete adjustment Q_{ca} of Q.

Step 2 to Step 5:
are similar to the corresponding steps of Heuristic Algorithm 1.

Figure 6.18 Heuristic algorithm 2.

Example 6.3.4. *Consider again the PTPQ Q of Figure 6.16 which is in full form. PPs p_1, p_2 and p_3 are redundant but this cannot be identified by homomorphisms. Figure 6.19 shows the complete adjustment of Q. Our second heuristic algorithm on Q is able to identify the redundancy of all three PPs p_1, p_2 and p_3 of Q, since there is a homomorphism from Q_a to $Q_a - \{p_1, p_2, p_3\}$ mapping the PPs p_1, p_2 and p_3 to virtual PPs as shown in Figure 6.19, and removes them from Q.* □

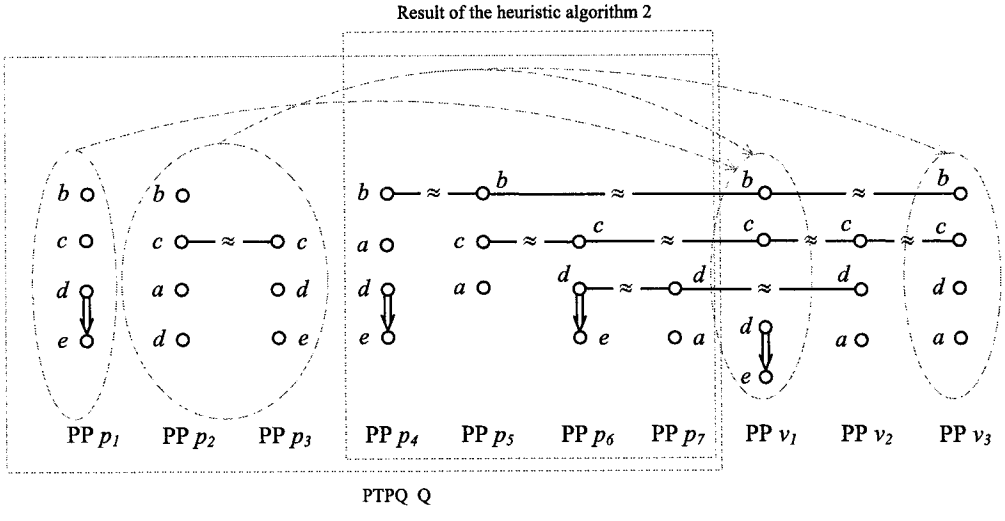

Figure 6.19 The complete adjustment of the PTPQ Q of Figure 6.16.

6.4 Conclusion

In this chapter we studied the minimization problem for PTPQ in the absence of dimension graphs. We showed that PTPQs cannot be minimized by removing redundant parts as is the case with TPQs involving branching and descendant relationships. We demonstrated that, in general, a PTPQ does not have a unique minimal equivalent PTPQ. We also showed that a PTPQ can be equivalently represented by a set of partial tree-pattern queries called component PTPQs, which can be minimized by removing redundant parts. Finally we devised sound, but not complete, heuristic techniques for PTPQ minimization, which gradually trade execution time for accuracy. They identify and remove redundant parts, which cannot by identified as redundant using homomorphisms even if the full form of the PTPQ is used. Moreover, these techniques are able to compute minimal PTPQs, even in cases where minimal one cannot be obtained by removal of redundant parts.

CHAPTER 7

CONCLUSION

We have considered partial tree-pattern queries (PTPQs), which correspond to a large fragment of XPath strictly containing tree-pattern queries. A central feature of this type of queries is that the structure can be specified fully, partially, or not at all in a query. Therefore, they can be used to query data sources whose structure is not fully known to the user, or to query multiple data sources which structure information differently. In this thesis we dealt with the processing of partial tree-pattern queries. This issue has not been addressed previously for this class of queries.

In order to handle structural expression inference and to allow query comparison, we defined a set of inference rules and a "normal form" for PTPQs, called *full form*. Using the concept of the full form we provided necessary and sufficient conditions for PTPQ satisfiability. We introduced the concept of *homomorphism* between two PTPQs to characterize their containment. Further, we defined a construct that summarizes the structural information of the database called *dimension graph*. A dimension graph can be automatically extracted from a database and supports the processing and evaluation of the PTPQs.

We studied the problem of PTPQ containment in the presence of dimension graphs, and we provided necessary and sufficient conditions for PTPQ containment. We further devised sound but not complete heuristic approaches that exploit structural information extracted from the dimension graph either in advance or at query time. A detailed experimental evaluation of our approaches shows that they greatly improve the PTPQ containment checking execution time, and that they gradually trade execution time for accuracy.

Next we focused on processing PTPQs in the absence of dimension graphs. We studied the containment problem for PTPQs and we proved that a PTPQ can be equivalently represented by a set of tree-pattern queries. We provided necessary and sufficient conditions for PTPQ containment. We also identified a subclass of PTPQs where containment can be fully characterized by the existence of homomorphisms. Further, we devised two sound but not complete heuristic approaches that equivalently add virtual partial paths to PTPQs. These heuristics gradually trade execution time for accuracy. Empirical implementation of both approaches confirms that they greatly improve the query containment checking execution time while maintaining very high accuracy.

In the framework of this thesis we also studied the problem of minimization of PTPQs. We demonstrated that PTPQs cannot be minimized by removing redundant parts as is the case with tree pattern queries involving branching and descendant relationships. Further, we showed that for some PTPQs there is no unique minimal equivalent PTPQ. We also proved that a PTPQ can be equivalently represented by a set of partial tree-pattern queries called component PTPQs. Finally, we devised sound but not complete heuristic approach for minimizing PTPQs.

The presented results show that our techniques can be used by query optimizers in real world applications for the processing and optimization of partially specified tree-pattern queries.

BIBLIOGRAPHY

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1997.

[2] V. Aguilera, S. Cluet, T. Milo, P. Veltri, and D. Vodislav. Views in a Large-scale XML Repository. *The VLDB Journal*, 11(3):238–255, 2002.

[3] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. *SIGMOD Rec.*, 30(2):497–508, 2001.

[4] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*, pages 496–513, London, UK, 2002. Springer-Verlag.

[5] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 83–94, New York, NY, USA, 2004. ACM.

[6] A. Barta, M. P. Consens, and A. O. Mendelzon. Benefits of Path Summaries in an XML Query Optimizer Supporting Multiple Access Methods. In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pages 133–144. VLDB Endowment, 2005.

[7] M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. In *PODS '05: Proceedings of the Twenty-Fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 25–36, New York, NY, USA, 2005. ACM.

[8] M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. *Journal ACM*, 55(2):1–79, 2008.

[9] M. Benedikt and I. Fundulaki. XML Subtree Queries: Specification and Composition. In *DBLP 2005*, pages 138–153. Springer-Verlag, 2005.

[10] B. Cautis, A. Deutsch, and N. Onose. XPath Rewriting Using Multiple Views: Achieving Completeness and Efficiency. In *WebDB*, 2008.

[11] D. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *WebDB (Informal Proceedings)*, pages 53–62, 2000.

[12] S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 271–280, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[13] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: a Semantic Search Engine for XML. In *vldb'2003: Proceedings of the 29th International Conference on Very Large Databases*, pages 45–56. VLDB Endowment, 2003.

[14] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *WWW '99: Proceedings of the Eighth International Conference on World Wide Web*, pages 1155–1169, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[15] A. Deutsch and V. Tannen. Containment and Integrity Constraints for XPath. In *KRDB*, 2001.

[16] L. Feng and T. Dillon. An XML-Enabled Data Mining Query Language: XML-DMQL. *Int. J. Bus. Intell. Data Min.*, 1(1):22–41, 2005.

[17] S. Flesca, F. Furfaro, and E. Masciari. On the Minimization of XPath Queries. In *VLDB*, pages 153–164, 2003.

[18] S. Flesca, F. Furfaro, and E. Masciari. On the Minimization of XPath Queries. *J. ACM*, 55(1):1–46, 2008.

[19] D. Florescu, D. Kossmann, and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *BDA*, 2000.

[20] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 436–445, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[21] G. Gottlob, C. Koch, and R. Pichler. The Complexity of XPath Query Evaluation. In *PODS '03: Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 179–190, New York, NY, USA, 2003. ACM.

[22] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The Complexity of XPath Query Evaluation and XML Typing. *J. ACM*, 52(2):284–335, 2005.

[23] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Integrating XML Data Sources Using Approximate Joins. *ACM Trans. Database Syst.*, 31(1):161–207, 2006.

[24] A. Y. Halevy. Answering Queries Using Views: A Survey. *The VLDB Journal*, 10(4):270–294, 2001.

[25] J. Hidders. Satisfiability of XPath Expressions. In *DBPL*, pages 21–36, 2003.

[26] Y. Kanza and Y. Sagiv. Flexible Queries over Semistructured Data. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 40–51, New York, NY, USA, 2001. ACM.

[27] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering Indexes for Branching Path Queries. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 133–144, New York, NY, USA, 2002. ACM.

[28] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, Washington, DC, USA, 2002. IEEE Computer Society.

[29] B. Kimelfeld and Y. Sagiv. Revisiting Redundancy and Minimization in an XPath Fragment. In *EDBT '08: Proceedings of the 11th International Conference on Extending Database Technology*, pages 61–72, New York, NY, USA, 2008. ACM.

[30] L. V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. Zhao. On Testing Satisfiability of Tree Pattern Queries. In *VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 120–131. VLDB Endowment, 2004.

[31] Y. Li, C. Yu, and H. V. Jagadish. Schema-Ffee XQuery. In *VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Databases*, pages 72–83. VLDB Endowment, 2004.

[32] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 329–340, New York, NY, USA, 2007. ACM.

[33] G. Miklau and D. Suciu. Containment and Equivalence for a Fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

[34] T. Milo and D. Suciu. Index Structures for Path Expressions. In *Proceedings of the 9th International Conference on Database Theory*, pages 277–295, 1999.

[35] F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, pages 315–329, London, UK, 2002. Springer-Verlag.

[36] F. Neven and T. Schwentick. On the Complexity of XPath Containment in the Presence of Disjunction, DTDs, and Variables. *LMCS Journal*, pages 1–30, 2006.

[37] D. Olteanu. Forward Node-Selecting Queries Over Trees. *ACM Trans. Database Syst.*, 32(1):3, 2007.

[38] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking Forward. In *EDBT Workshops*, pages 109–127, 2002.

[39] P. Placek, D. Theodoratos, S. Souldatos, T. Dalamagas, and T. Sellis. A Heuristic Approach for Checking Containment of Generalized Tree-Pattern Queries. In *CIKM '08: Proceedings of the 17th ACM International Conference on Information and Knowledge Management*, pages 551–560, 2008.

[40] N. Polyzotis and M. Garofalakis. Statistical Synopses for Graph-Structured XML Databases. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 358–369, New York, NY, USA, 2002. ACM.

[41] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Approximate XML Query Answers. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 263–274, New York, NY, USA, 2004. ACM.

[42] P. Ramanan. Efficient Algorithms for Minimizing Tree Pattern Queries. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 299–309, New York, NY, USA, 2002. ACM.

[43] J. Robie, J. Lapp, and D. Schach. *XML Query Language (XQL).* http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[44] T. Schwentick. XPath Query Containment. *SIGMOD Rec.*, 33(1):101–109, 2004.

[45] S. Souldatos, X. Wu, D. Theodoratos, T. Dalamagas, and T. Sellis. Evaluation of Partial Path Queries on Xml Data. In *CIKM '07: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, pages 21–30, New York, NY, USA, 2007. ACM.

[46] B. ten Cate and C. Lutz. The Complexity of Query Containment in Expressive Fragments of XPath 2.0. In *PODS '07: Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 73–82, New York, NY, USA, 2007. ACM.

[47] D. Theodoratos, T. Dalamagas, A. Koufopoulos, and N. Gehani. Semantic Querying of Tree-Structured Data Sources Using Partially Specified Tree Patterns. In *CIKM '05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pages 712–719, New York, NY, USA, 2005. ACM.

[48] D. Theodoratos, T. Dalamagas, P. Placek, S. Souldatos, and T. Sellis. Containment of Partially Specified Tree-Pattern Queries. In *SSDBM '06: Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[49] D. Theodoratos, P. Placek, T. Dalamagas, S. Souldatos, and T. K. Sellis. Containment of partially specified tree-pattern queries in the presence of dimension graphs. *The VLDB Journal*, 18(1):233–254, 2009.

[50] D. Theodoratos, S. Souldatos, T. Dalamagas, P. Placek, and T. Sellis. Heuristic Containment Check of Partial Tree-Pattern Queries in the Presence of Index Graphs. In *CIKM '06: Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 445–454, New York, NY, USA, 2006. ACM.

[51] D. Theodoratos and X. Wu. An Original Semantics to Keyword Queries for XML Using Structural Patterns. In *DASFAA '07: Proceedings of the 12th International Conference on Database Systems for Advanced Application*, 2007.

[52] D. Theodoratos and X. Wu. Assigning Semantics to Partial Tree-Pattern Queries. *Data Knowl. Eng.*, 64(1):242–265, 2008.

[53] D. Theodoratos and X. Wu. Eager Evaluation of Partial Tree-Pattern Queries on XML Streams. In *DASFAA '09: Proceedings of the 14th International Conference on Database Systems for Advanced Applications*, pages 241–246, 2009.

[54] P. T. Wood. Minimising Simple XPath Expressions. In *WebDB*, pages 13–18, 2001.

[55] P. T. Wood. Containment for XPath Fragments under DTD Constraints. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, pages 300–314, London, UK, 2002. Springer-Verlag.

[56] World Wide Web Consortium site, http://www.w3.org/TR/XLink/. *XML Linking Language (XLink).*

[57] World Wide Web Consortium site, http://www.w3.org/TR/XPath20. *XML Path Language (XPath).*

[58] World Wide Web Consortium site, http://www.w3.org/TR/XQuery/. *XQuery 1.0: An XML Query Language.*

[59] World Wide Web Consortium site, http://www.w3.org/TR/XSLT/. *XSL Transformations (XSLT).*

[60] World Wide Web Consortium site, http://www.w3.org/XML/. *Extensible Markup Language (XML).*

[61] X. Wu, S. Souldatos, D. Theodoratos, T. Dalamagas, and T. Sellis. Efficient Evaluation of Generalized Path Pattern Queries on XML Data. In *Proceedings of the 17th International World Wide Web Conference (WWW'08).* ACM, 2008.

[62] X. Wu and D. Theodoratos. Evaluating Partial Tree-Pattern Queries on XML Streams. In *CIKM '08: Proceeding of the 17th ACM Conference on Information and Knowledge Management*, pages 1409–1410, New York, NY, USA, 2008. ACM.

[63] X. Wu, D. Theodoratos, S. Souldatos, T. Dalamagas, and T. K. Sellis. Efficient Evaluation of Generalized Tree-Pattern Queries with Same-Path Constraints. In *SSDBM '09: Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, pages 361–379, 2009.

[64] W. Xu and Z. M. Özsoyoglu. Rewriting XPath Queries Using Materialized Views. In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pages 121–132. VLDB Endowment, 2005.

[65] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 527–538, New York, NY, USA, 2005. ACM.