A Heuristic Approach for Checking Containment of Generalized Tree-Pattern Queries

Pawel Placek
NJIT, USA
pp58@njit.edu

Dimitri Theodoratos
NJIT, USA
dth@cs.njit.edu

Stefanos Souldatos
NTUA, Greece
stef@dblab.ntua.gr

Theodore Dalamagas
IMIS, Greece
dalamag@imis.athena-innovation.gr

Timos Sellis
IMIS & NTUA, Greece
timos@imis.athena-innovation.gr

ABSTRACT

Query processing techniques for XML data have focused mainly on tree-pattern queries (TPQs). However, the need for querying XML data sources whose structure is very complex or not fully known to the user, and the need to integrate multiple XML data sources with different structures have driven, recently, the suggestion of query languages that relax the complete specification of a tree pattern. In order to implement the processing of such languages in current DBMSs, their containment problem has to be efficiently solved.

In this paper, we consider a query language which generalizes TPQs by allowing the partial specification of a tree pattern. Partial tree-pattern queries (PTPQs) constitute a large fragment of XPath that flexibly permits the specification of a broad range of queries from keyword queries without structure, to queries with partial specification of the structure, to complete TPQs. We address the containment problem for PTPQs. This problem becomes more complex in the context of PTPQs because the partial specification of the structure allows new, non-trivial, structural expressions to be inferred from those explicitly specified in a query. We show that the containent problem cannot be characterized by homomorphisms between PTPQs, even when PTPQs are put in a canonical form that comprises all derived structural expressions. We provide necessary and sufficient conditions for this problem in terms of homomorphisms between PTPQs and (a possibly exponential number of) TPQs. To cope with the high complexity of PTPQ containment, we suggest a heuristic approach for this problem that trades accuracy for speed. An extensive experimental evaluation of our heuristic shows that our heuristic approach can be efficiently implemented in a query optimizer.

1. INTRODUCTION

The increased popularity of XML has triggered the interest of the database community on the processing of XPath [1]. XPath lies at the core of W3C language proposals for XML querying (e.g. XQuery [1]). In this context, a challenging issue is the effective and efficient querying of tree-structured data on the web. This goal faces several obstacles: (a) a tree-structured data source may contain structured data (i.e., relational data) along with unstructured data (i.e., text), (b) the user may not know the (full) tree structure of a data source, and (c) the user needs to query in an integrated way several data sources that structure their information differently.

Traditional information integration approaches attempt to cope with the issue of querying multiple tree-structured data sources by providing a global structure. Mapping rules, e.g. node-to-node or path-to-path, are defined between the global structure and the local structures used in the sources [7]. Such approaches require extensive manual effort since the global schema is difficult to construct and the rules should be hard-coded in the integration application. Moreover, the user should have exact knowledge of the global structure in order to formulate queries. Approximation techniques can also be used to generate alternative forms of a query and search for answers in the data sources. For instance, tree-pattern query relaxation methods are presented in [3, 4], while methods for producing approximate answers to XML queries are suggested in [12, 24]. In the same direction, flexible and semiflexible semantics were introduced in [14] for tree and dag queries. Nevertheless, in all these cases the answer is not exact with respect to the initial query. Approaches based on keyword search [26, 13, 8, 18] avoid the issue of unknown structure or the issue of multiple differently structured data sources since knowledge of the structure is not required for the queries. However, the total absence of structure has a number of drawbacks: (a) the user cannot specify structural information within the query to accelerate computation of the answer, (b) the user cannot impose structural conditions to filter out undesirable answers, and (c) the user cannot specify the structure of the query result.

Some approaches aim at extending full-fledged XML query languages with keyword-based search techniques [11, 17]. However, these languages are too complex for the simple user. Approaches with languages based on tree patterns share a restrictive feature: it is not possible in a tree pattern to indicate that two nodes occur in the same path without specifying a structural relationship between them. Such a restriction causes problems both to the formulation and pro-

cessing of requests that do not specify an order among some nodes. Indeed, a number of tree-pattern queries (TPQs) that is exponential on the number of nodes of the query might need to be specified and processed.

Our approach. To face the previous issues, we use Partial Tree-Pattern Queries (PTPQs), a query language that generalizes TPQs by allowing a partial specification of a tree structure in the query. This language corresponds to a broad fragment of XPath [1] that contains TPQs (XPath expressions with predicates, and child and descendant axes) but also the reverse axes parent and anscestor and the node identity equality operator 'is'. The structure in a PTPQ can be flexibly specified *fully, partially or not at all*. PTPQs range from unstructured keyword-style queries to completely specified TPQs. This flexibility in specifying structural constraints allows their use for dealing with the issues mentioned above.

The problem. For a query language to be useful, it needs to be complemented with query processing and optimization techniques. Important issues in query optimization, including query satisfiability [16], query minimization [2, 30, 25], and query rewriting using views [23, 15], require solving the query containment problem. In this paper, we address the query containment for PTPQs. The query containment problem has been studied in the past for (fully specified) tree-pattern queries in the absence and in the presence of constraints. However, most of the previous work focuses almost exclusively on characterizing the complexity of the problem for tree patterns. Our goal here is to provide an efficient (sound but not necessarily complete) method for checking query containment that can be used for processing partial tree-pattern queries.

Contribution. We start by formally defining the containment problem for PTPQs. In order to allow PTPQ comparison we use a "canonical form" for queries, called *full form*. Intuitively, a full form of a PTPQ comprises all the structural expressions that can be inferred from those specified in the PTPQ. The main contributions of this paper are the following:

- We show that a PTPQ is equivalent to a set of TPQs called component TPQs. However, we also show that the number of these TPQs can be exponential on the number of nodes of the PTPQ.
- We show that homomorphisms cannot completely characterize PTPQ containment even if the full form of the PTPQ to be checked for containment is used. That is, the existence of a homomorphism implies containment but the opposite is not necessarily true.
- We provide necessary and sufficient conditions for PTPQ containment in terms of homomorphisms from PTPQs to component TPQs. Since the number of component TPQs can be exponential, this technique for checking PTPQ containment is of high complexity and cannot be used in practice.
- We explain why the containment of PTPQs cannot be fully characterized by homomorphisms. Based on this explanation we identify a subclass of PTPQs (which strictly contains TPQs) for which containment can be completely characterized by homomorphisms.
- In order to deal with the high complexity of PTPQ containment checking using the component TPQs, we design a sound but not complete heuristic technique which trades time for accuracy. This technique is based on equivalenty adding new (partial) paths to the PTPQ to be checked for containment in order to increase the possibility for a homomorphism to this PTPQ to exist.
- We have implemented our containment checking technique, and performed an extensive experimentation to evaluate it. Our experiments show that the heuristic technique is efficient compared to the non-heuristic one while maintaining high accuracy. These results show that it can be directly exploited in practice for the processing of PTPQs.

Outline. The next section reviews related work. Section 3 presents the data model and the query language, shows that PTPQs are equivalent to sets of component TPQs, and discusses inference issues. Section 4 provides necessary and sufficient conditions for PTPQ containment. In Section 5, we explain why homomorphisms do not always work and identify a subclass of PTPQs which is completly characterized by homomorphisms. Section 6 introduces our heuristic approache. Experimental results are shown in Section 7. We conclude in Section 8 and discuss future work. Because of lack of space, detailed proofs are omitted.

2. RELATED WORK

Because of their importance for query processing, a considerable amount of work has focused on the containment problem for tree-pattern queries in the presence and in the absence of schemas [9, 14, 19, 31, 20, 10, 6]. In particular, Neven and Schwentick [20] studied the complexity of tree-pattern queries (involving child and descendant relationships and wildcards) in the presence of disjunction and DTDs. As we show later in the paper, a partially specified tree-pattern query can be expressed as a set of tree-pattern queries. However, the containment problem addressed in [20] is different than that of partially specified tree-pattern queries because not every set of tree-pattern queries can be expressed as a partially specified tree-pattern query. Benedikt and Fundulaki [5] study query composition for different fragments of XPath under subquery semantics. None of these papers addresses query containment for partial tree-pattern queries. Most of the aforementioned works focus on studying the complexity of these problems for different classes of tree-pattern queries. Our goal in this paper is different. We are focusing on providing heuristic techniques for checking query containment that can be used for efficiently processing and optimizing partial tree-pattern queries.

Partially specified tree-pattern queries were initially introduced in [27]. Initial results on the query containment problem for partial tree-pattern queries were presented in [28, 29], where inference rules were also provided to completely characterize the inference of structural expressions. In both of these papers the class of partial tree-pattern queries considered is restricted to disallow cases of unordered node sharing expressions between two or three partial paths. Further, the focus of [29] was on heuristic approaches in the presence of indexes called dimension graphs. These results *cannot* be applied to heuristically check containment of PTPQs in the absence of dimension graphs. The present paper is the first one to provide necessary and sufficient conditions for partial tree-pattern query containment along with an elaborate (sound but not complete) heuristic approach for checking query containment. This approach can be exploited in query optimizers for processing partial tree-pattern queries.

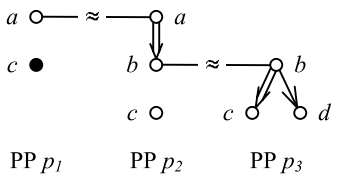

Figure 1: PTPQ Q_1

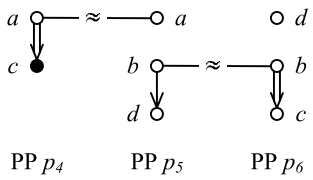

Figure 2: PTPQ Q_2

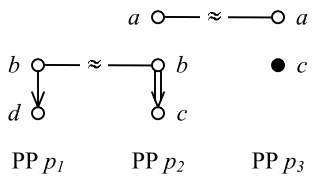

Figure 3: PTPQ Q_3

3. THE PARTIAL TREE-PATTERN QUERY LANGUAGE

We briefly present in this section the data model and our query language that allows generalizing tree patterns.

3.1 Data model and query language

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. For simplicity, we assume that the same element does not label two nodes on the same path.

DEFINITION 3.1. 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. □

We graphically represent PTPQs using graph notation. Each PP of a PTPQ Q is represented as a (not necessarily connected) graph of elements. The name of each PP is shown by the corresponding PP graph. The output node of Q is denoted by a filled black node. Child and descendant precedence relationships in a PP are depicted using single ($\rightarrow$) and double ($\Rightarrow$) arrows between the respective elements in the PP graph. In particular, descendant precedence relationships of the form $r \Rightarrow e$ in a PP are shown only with the presence of element e in the PP graph. 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 1, 2, and 3 show three PTPQs.

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

DEFINITION 3.2. *An* embedding *of a PTPQ Q to a database D is a mapping M of the elements of the PPs of Q to nodes in D such that: (a) an element e of Q is mapped by M to a node in D labeled by e; (b) the elements of a PP in Q are mapped by M to nodes in D that are on the same path; (c) $\forall\ e_i[p] \rightarrow e_j[p]$ (resp. $e_i[p] \Rightarrow e_j[p]$) in Q, $M(e_j[p])$ is a child (resp. descendant) of $M(e_i[p])$ in D; and (d) $\forall e[p_i] \approx e[p_j]$ in Q, $M(e[p_i])$ and $M(e[p_j])$ coincide.* □

We call *image* of a PP p in Q under M, denoted $M(p)$, the path from the root of D that comprises all the images of the elements of p under M and ends in one of them. Notice that more than one PP of Q may have their image on the same root-to-leaf path of D (M does not have to be a bijection). The *answer* of Q on D is the set of the images of the output node of Q under all possible embeddings of Q to D.

Clearly, the class of PTPQs encompasses TPQs: every TPQ can be represented by a PTPQ that returns the same answer on every database. 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.

A PTPQ is *satisfiable* if and only if it has a non-empty answer on some database. In the following, we assume that queries are satisfiable.

3.2 PTPQ containment and equivalence

In this paper we focus on the containment of PTPQs:

DEFINITION 3.3. *Let Q_1 and Q_2 be two PTPQs. Q_2* contains *Q_1 (denoted $Q_1 \subseteq Q_2$) if and only if for every database D, the answer of Q_1 on D is a subset of the answer of Q_2 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 $Q_2 \subseteq Q_1$.* □

EXAMPLE 3.1. *Consider the PTPQs Q_1 Q_2 and Q_3 of Figures 1, 2, and 3 respectively. One can see that $Q_1 \subseteq Q_2$. Observe, for instance, that the image of the output node of Q_1 under any embedding of Q_1 to a database D is also an image of the output node of Q_2 under some embedding of Q_2 to D. Similarly, $Q_2 \subseteq Q_3$. In contrast, $Q_2 \not\subseteq Q_1$. We will prove these claims formally later.* □

In the next sections of this paper, we study exact and heuristic methods for checking PTPQ containment.

3.3 Inference of structural expressions

Precedence relationships and node sharing expressions are collectively called *structural expressions*. Because the structure of tree patterns is partially specified in PTPQs, new, non-trivial structural expressions can be derived from those explicitly specified in the queries. Some of the derivations are relevant also to TPQs. Consider, for instance, query Q_1 of Figure 1. Let's use the symbol $\vdash$ to denote that the structural expressions in its left hand side derive the structural expression in its right hand side. Clearly, $b[p_3] \rightarrow d[p_3] \vdash$

$b[p_3] \Rightarrow d[p_3]$ Many other derivations are specific to PTPQs. For instance, $b[p_3] \rightarrow d[p_3],\ b[p_3] \Rightarrow c[p_3] \vdash d[p_3] \Rightarrow c[p_3]$.

More formally, let S be a set of structural expressions of a PTPQ Q, and s be a structural expression. We say that s can be *inferred* from S iff for every embedding M of Q to a database, M satisfies s. The inferred structural expressions can be computed efficiently using a set of inference rules [29].

To study properties of PTPQs, we introduce a "normal form" for PTPQs called *full form*.

DEFINITION 3.4. *A PTPQ Q is in full form if and only if the set S of its structural expressions comprises all the structural expressions that can be inferred from S.* □

Given a PTPQ Q, we can construct a PTPQ in full form by replacing its set S of structural expressions in Q by the closure of S. Clearly, this PTPQ is *unique* and *equivalent* to Q. Therefore, we refer, in the following, to *the* full form of a PTPQ.

Figures 4 and 5 show the PTPQs Q'_1 and Q'_2 which are the full forms of the PTPQs Q_1 and Q_2 of Figures 1 and 2 respectively. Query Q_3 of Figure 3 is in full form. For clarity of presentation, when graphically representing queries in full form, we do not depict structural expressions that can be trivially or transitively derived from the shown structural expressions. Observe that the full form of Q_1 shows that this PTPQ is also a TPQ.

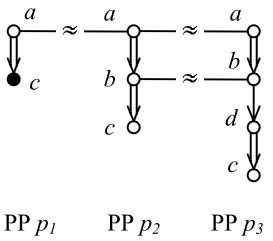

Figure 4: PTPQ Q'_1, the full form of Q_1

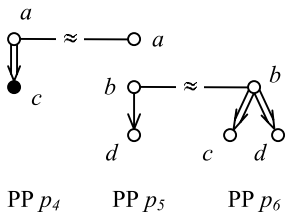

Figure 5: PTPQ Q'_2, the full form of Q_2

Clearly, the number of structural expressions that can be derived using the inference rules in a query is bounded by $O(n^2)$, where n is the product of the maximum number of distinct elements in the query and its number of PPs. Therefore, the full form of a query can be computed in polynomial time.

3.4 Component TPQs

We show now that the answer of a PTPQ Q 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 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 3.1. *Let Q be a PTPQ. If Q is safisfiable 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.* □

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

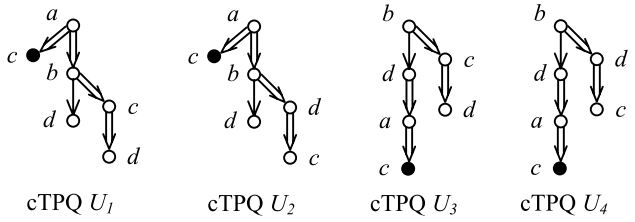

Figure 6: The four cTPQs for PTPQ Q_2

DEFINITION 3.5. *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. Consider the PTPQ Q_2 of Figure 2. Figure 6 shows the four component TPQs U_1, U_2, U_3 and U_4 of Q_2. The PTPQ Q_1 of Figure 1 has only one cTPQ since its full form shown in Figure 4 is a TPQ.

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

PROPOSITION 3.2. *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 $Q, U_1, \ldots, U_k$ 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 cTPQs.

The PTPQ language can express all expressions in the fragment of XPath [1] that allows predicates (branching) and child (/) and descendant (//) axes. This fragment corresponds to tree-pattern queries (without wildcards). PTPQs can also express the fragment of XPath that allows, in addition, the symmetric reverse axes parent (\) and ancestor (\\), and the node identity equality operator (is). However, PTPQs strictly contain this fragment of XPath. For instance, it is not difficult to see that a PTPQ that contains two nodes other than r in one PP without precedence relationship between them cannot be expessed by this fragment of XPath.

4. NECESSARY AND SUFFICIENT CONDITIONS FOR PTPQ CONTAINMENT

For TPQs that involve the descendant axis (that is, descendant precedence relationships), and branching ([]), but no wildcards (*), the existence of a homomorphism is a necessary and sufficient condition for containment [2, 19]. In order to check if a similar result holds for PTPQs we extend the concept of homomorphism for TPQs to homomorphism for PTPQs:

DEFINITION 4.1. *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 4.1. *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. The full form of a query Q is a query equivalent to Q and 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 4.1. *Consider the PTPQs* Q_1 *and* Q_2 *of Example 3.1 shown in Figures 1 and 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 4. This proves our claim of Example 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 and 3 respectively. PTPQ Q_2' is the full form of PTPQ Q_2 of Figure 2. One can easily see that there is no homomorphism from Q_3 to Q_2'. However, as we mentioned in Example 3.1, $Q_2 \subseteq Q_3$.

We elaborate in Section 5.2 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 homorphisms from a PTPQ to TPQs:

THEOREM 4.1. *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 an homomorphism from* Q_2 *to* U*.* □

EXAMPLE 4.2. *Consider again the PTPQs* Q_1*,* Q_2 *and* Q_3 *of Example 3.1 shown in Figures 1, 2, and 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 6. This proves our claim of Example 3.1 that* $Q_2 \subseteq Q_3$*.*

In contrast, it is easy to see that there no homorphism from Q_1 *to at least one of the cTPQs of* Q_2 *(in fact, there is no homomorhism to any one of the cTPQs of* Q_2*). This proves our claim of Example 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. [22, 21] 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 unfeasible to check general PTPQ containment by checking the existence of homomorphisms between a PTPQ and an exponential number of TPQs.

5. 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.

5.1 Why homomorphisms do not always work

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, a homomorphism from Q' to Q will not exist since these nodes and precedence relationships do not appear together in any PP of Q.

EXAMPLE 5.1. *Consider the PTPQ* Q_2 *of Figure 2 whose full form is shown in Figure 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. This PTPQ comprises the 3-path swing* $\{a[p_4] \approx a[p_5], b[p_5] \approx b[p_6]\}$*. 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 3 which does not have a homomorphism to the full form of PTPQ* Q_2 *even though, as proved in example 4.2,* $Q_2 \subseteq Q_3$*.* □

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 $a[p_1] \rightarrow a_1[p_1],\ a_1[p_1] \rightarrow a_2[p_1],\ \ldots,\ a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2],\ b_1[p_2] \rightarrow b_2[p_2],\ \ldots,\ b_{l-1}[p_2] \rightarrow b_l[p_2]$, 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 7(a). Then, every tree in which there is an embedding of Q comprises a path that satisfies the child precedence relationships $a[p_1] \rightarrow a_1[p_1],\ a_1[p_1] \rightarrow a_2[p_1],\ \ldots,\ a_{k-1}[p_1] \rightarrow a_k[p_1],\ b[p_2] \rightarrow b_1[p_2],\ b_1[p_2] \rightarrow b_2[p_2],\ \ldots,\ a_{l-1}[p_2] \rightarrow a_l[p_2]$ 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 5.2. *Consider the PTPQ Q_6 of Figure 7(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 7(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, $Q_6 \subseteq Q_7$.* □

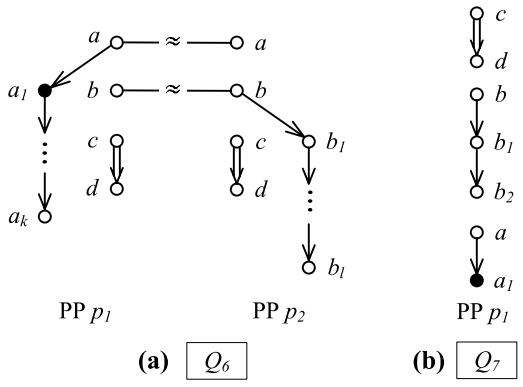

Figure 7: (a) PTPQ Q_6, a PTPQ with a 2-path swing, (b) PTPQ Q_7

5.2 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 5.1. *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 $a[p_1] \rightarrow a_1[p_1],\ a_1[p_1] \rightarrow a_2[p_1],\ \ldots,\ a_{k-1}[p_1] \rightarrow a_k[p_1]$ and $b[p_2] \rightarrow b_1[p_2],\ b_1[p_2] \rightarrow b_2[p_2],\ \ldots,\ b_{l-1}[p_2] \rightarrow b_l[p_2]$, then it also symmetrically contains the chains $a[p_2] \rightarrow a_1[p_2],\ a_1[p_2] \rightarrow a_2[p_2],\ \ldots,\ a_{k-1}[p_2] \rightarrow a_k[p_2]$ and $b[p_1] \rightarrow b_1[p_1],\ b_1[p_1] \rightarrow b_2[p_1],\ \ldots,\ b_{l-1}[p_1] \rightarrow b_l[p_1]$, and the node sharing expressions $a_1[p_1] \approx a_1[p_2], \ldots,\ a_k[p_1] \approx a_k[p_2]$, and $b_1[p_1] \approx b_1[p_2], \ldots,\ b_l[p_1] \approx b_l[p_2]$.* □

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 5.1. *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 an homomorphism from Q' to Q.* □

In [2, 19] 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 $\mathcal{C}$.

6. 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 homomorhism 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.

6.1 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 6.1. *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 6.1. *Consider query Q'_2 of Figure 5 (which is the full form of query Q_2 of Figure 2). Figure 8(a) shows the virtual PP v of PP p_5 w.r.t. PPs p_4 and p_6 in Q'_2.* □

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

DEFINITION 6.2. *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 6.2. *Figure 8(b) shows the adjustment Q''_2 of query Q'_2 of Figure 5 with the virtual PP v shown in Figure 8(a). The full form Q'''_2 of Q''_2 is shown in Figure 8(c). Figure 8(d) redraws query Q_3 of Figure 3. As proven in example 4.2, $Q_2 \subseteq Q_3$ (and $Q'_2 \subseteq Q_3$ since $Q_2 \equiv Q_2$). Also, as stated in Section 4 (and one can easily check), there is no homomorphism from Q_3 to Q'_2. However, Figures 8(c) and*

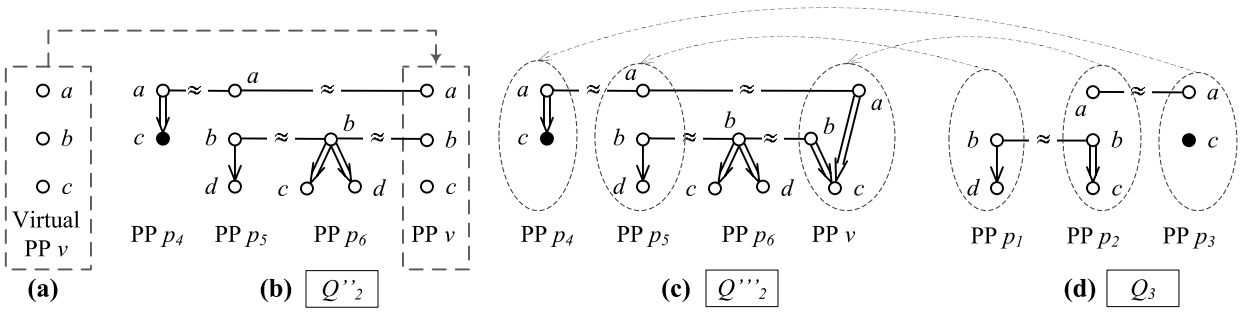

Figure 8: (a) The virtual PP v of p_5 w.r.t. p_4 and p_6 in Q_2, (b) The adjustment Q_2'' of Q_2' with v, (c) the full form Q_2''' of Q_2'', (d) PTPQ Q_3, and an "outline" of a homomorphism from Q_3 to Q_2'''.

(d), show a homomorphism from Q_3 to 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. □

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

DEFINITION 6.3. *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 $a \rightarrow a_1,\ a_1 \rightarrow a_2,\ \ldots,\ a_{k-1} \rightarrow a_k$ and $b \rightarrow b_1,\ b_1 \rightarrow b_2,\ \ldots,\ a_{l-1} \rightarrow a_l$, 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 6.4. *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 6.3. *Figure 9(b) shows the adjustment, Q_6', of query Q_6 of Figure 7(a) with the virtual PP v_1 shown in Figure 9(a). As shown in example 5.2, the PTPQ Q_7 of Figure 7(b) contains the PTPQ Q_6 of Figure 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 6.1. *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.* □

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'.

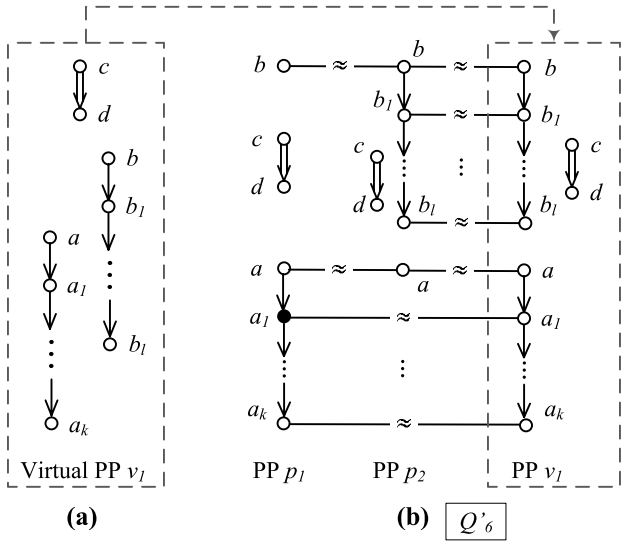

Figure 9: (a) The virtual PP v_1 of p_1 and p_2 in Q_6, (b) The adjustment Q_6' of Q_6 with v

6.2 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 homorphism, independent of the order of computation of the adjustments. PTPQ Q_a is called *adjustment* of Q. The formal process is shown in Figure 11.

Based on Theorem 6.1, 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 homo-

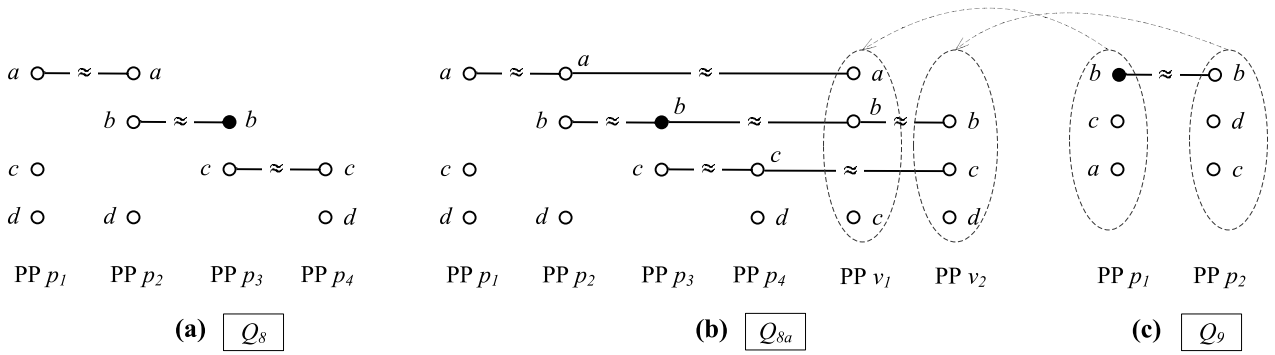

Figure 10: (a) PTPQ Q_8, (b) The adjustment Q_{8a} of Q_8, (c) PTPQ Q_9

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 11: Computation of the adjustment Q_a of a PTPQ Q

morphism 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 through a homomorphism.

EXAMPLE 6.4. *Consider the PTPQs Q_8 and Q_9 of Figures 10(a) and (c) respectively. PTPQ Q_8 is in full form. Clearly, there is no homomorphism from Q_9 to Q_8. Figure 10(b) shows the adjustment Q_{8a} of PTPQ Q_8 which comprises two vitual PPs v_1 and v_2. Figures 10(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.* □

7. EXPERIMENTAL EVALUATION

To study the effectiveness of our PTPQ containment checking technique, we ran a comprehensive set of experiments. Checking PTPQ containment using component TPQs is expected to be time consuming for queries that involve a large number of such TPQs. However, our experimental evaluation shows that the heuristic approach 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 and used for containment check. It is important to note that for each pair (Q_1, Q_2) of PTPQs used for containment check, $Q_1 \subseteq Q_2$, but there is no homomorphism from Q_2 to the full form of Q_1 due to the existence of 2-path and 3-path swings.

Experiments. In our experiments, we compared the execution time and the accuracy for PTPQ containment check in 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 (Adj), and (c) 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 12 and 13, 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 involve all PPs. Queries with 2 PPs include one 2-path swing (not shown due to lack of space), 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 homomorphism containment 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 examined in the containment check.

Our heuristic technique (Adj) clearly improves $ComTPQs$ check. The execution time of Adj is not generally affected by the number of elements in the queries. However, similarly to Hom, the execution time of Adj 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. For all measurements of this experiment, the accuracy of Adj is above 60%. Note that this percentage represents containment detection on pairs of contained PTPQs where containment cannot be detected through the ex-

istence of a homomorphism. Clearly, this percentage is much higher for random pairs of contained PTPQs since *Adj* can correctly detect containment when a homomorphism exists between the 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 PTPQs for different numbers of 3-path swings. In Figures 14 and 15, 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 homomorphism containment check (Hom) is not affected by the number of swings in the queries.

Similarly to the previous experiment, the execution time of $ComTPQs$ goes up as the number of elements in the queries increases. It also slightly decreases 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 the same image on the XML tree, thus a smaller number of component TPQs are generated and examined in the containment check.

Our heuristic technique again improves $ComTPQs$. When the number of swings in the queries goes up, the execution time of *Adj* increases. The reason is that the adjustment of queries with a large number of swings generally includes a larger number of virtual paths that need to be extracted and examined in the containment check. The accuracy of *Adj* in this experiment is above 60% for all measurements.

Remarks. Our heuristic technique clearly improves the time of PTPQ containment check. Although the technique is not complete, our experiments show clearly that it is orders of magnitude faster than checking PTPQ containment using component TPQs, while maintaining high accuracy.

8. CONCLUSION

We considered PTPQs which correspond to a large fragment of XPath strictly containing TPQs. 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.

We studied the containment problem for PTPQs, 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 approach that equivalently adds additional partial paths to PTPQs. A detailed experimental evaluation of our approach shows that it greatly improves the query containment checking execution time, and that it trades execution time for accuracy. These results allow its use for query processing and optimization.

We are currently working on using our heuristic technique to address minimization problems for PTPQs.

9. REFERENCES

[1] World Wide Web Consortium site (W3C), http://www.w3c.org.

[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD*, pages 497–508, 2001.

[3] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *EDBT*, 2002.

[4] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. Flexpath: Flexible structure and full-text querying for xml. In *SIGMOD*, pages 83–94, 2004.

[5] M. Benedikt and I. Fundulaki. XML subtree queries: Specification and composition. In *DBPL*, 2005.

[6] L. Chen and E. A. Rundensteiner. XQuery Containment in Presence of Variable Binding Dependencies. In *WWW*, 2005.

[7] S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale xml repository. In *VLDB*, 2001.

[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A Semantic Search Engine for XML. In *Proc. of VLDB*, 2003.

[9] A. Deutsch and V. Tannen. Containment and integrity constraints for xpath. In *KRDB*, 2001.

[10] X. Dong, A. Y. Halevy, and I. Tatarinov. Containment of Nested XML Queries. In *Proc. of VLDB*, 2004.

[11] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. *Computer Networks*, 33(1-6):119–135, 2000.

[12] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *SIGMOD*, pages 287–298, 2002.

[13] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. In *ICDE*, pages 367–378, 2003.

[14] Y. Kanza and Y. Sagiv. Flexible Queries Over Semistructured Data. In *PODS*, 2001.

[15] L. V. Lakshmanan, H. W. Wang, and Z. J. Zhao. Answering Tree Pattern Queries Using Views. In *VLDB*, 2006.

[16] L. V. S. Lakshmanan, G. Ramesh, H. W. Wang, and Z. J. Zhao. On Testing Satisfiability of Tree Pattern Queries. In *VLDB*, pages 120–130, 2004.

[17] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, pages 72–83, 2004.

[18] Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD*, pages 329–340, 2007.

[19] G. Miklau and D. Suciu. Containment and Equivalence for an XPath Fragment. In *PODS*, 2002.

[20] F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *ICDT*, pages 315–329, 2003.

[21] D. Olteanu. Forward node-selecting queries over trees. *ACM Trans. Database Syst.*, 32(1):3, 2007.

[22] D. Olteanu, H. Meuss, T. Furche, and F. Bry. Xpath: Looking forward. In *EDBT Workshops*, 2002.

[23] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *SIGMOD*, 1999.

[24] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Approximate XML query answers. In *SIGMOD*, 2004.

[25] P. Ramanan. Efficient Algorithms for Minimizing Tree Pattern Queries. In *SIGMOD*, pages 299–309, 2002.

[26] A. Schmidt, M. L. Kersten, and M. Windhouwer.

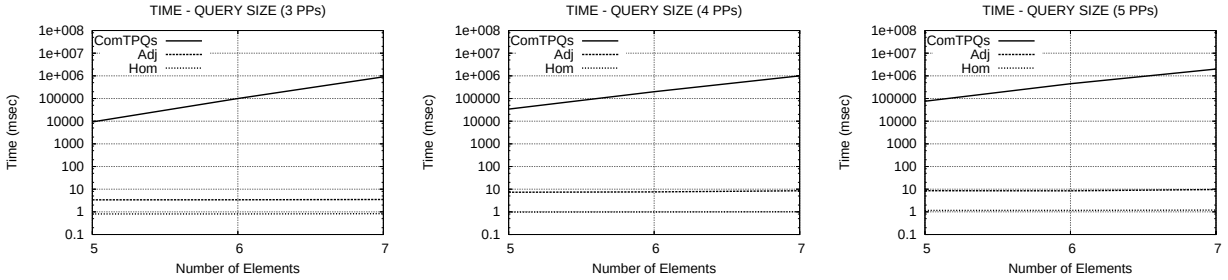

Figure 12: Execution time for checking PTPQ containment varying the size of the queries.

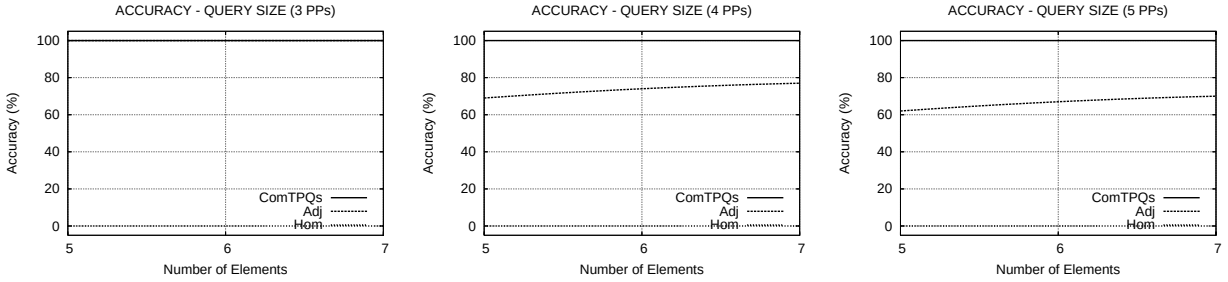

Figure 13: Percentage of correct answers in checking PTPQ containment varying the size of the queries.

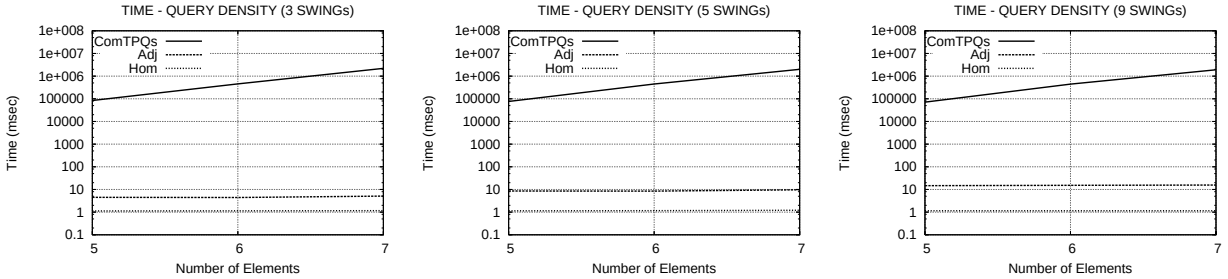

Figure 14: Execution time for checking PTPQ containment varying the density of swings in the queries.

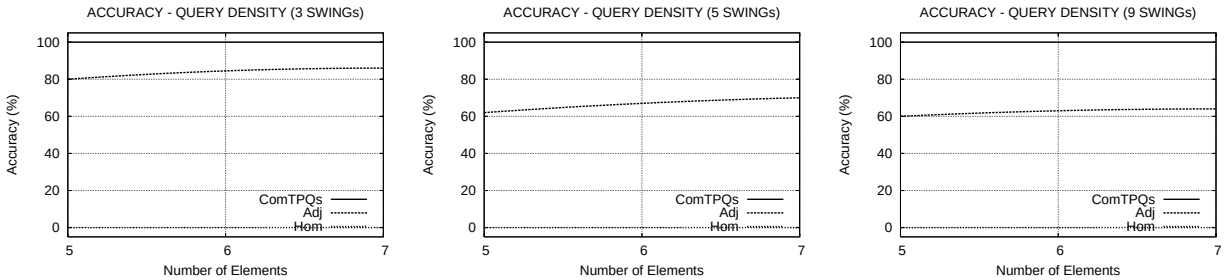

Figure 15: Percentage of correct answers in checking PTPQ containment varying the density of swings in the queries.

Querying XML Documents Made Easy: Nearest Concept Queries. In *ICDE*, pages 321–329, 2001.

[27] D. Theodoratos, T. Dalamagas, A. Koufopoulos, and N. Gehani. Semantic Querying of Tree-Structured Data Sources Using Partially Specified Tree-Patterns. In *CIKM*, pages 712–719, 2005.

[28] D. Theodoratos, T. Dalamagas, P. Placek, S. Souldatos, and T. Sellis. Containment of Partially Specified Tree-Pattern Queries. In *SSDBM*, 2006.

[29] 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*, pages 445–454, 2006.

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

[31] P. T. Wood. Containment for XPath Fragments under DTD Constraints. In *ICDE*, pages 300–314, 2003.