On the minimization of Xpath queries

S. Flesca, F. Furfaro
D.E.I.S. - Università della Calabria
Via P. Bucci
87036 - Rende (CS)
ITALY
{flesca, furfaro}@si.deis.unical.it

E. Masciari
ICAR - CNR
Via P. Bucci
87036 - Rende (CS)
ITALY
masciari@icar.cs.cnr.it

Abstract

XML queries are usually expressed by means of XPath expressions identifying portions of the selected documents. An XPath expression defines a way of navigating an XML tree and returns the set of nodes which are reachable from one or more starting nodes through the paths specified by the expression. The problem of efficiently answering XPath queries is very interesting and has recently received increasing attention by the research community. In particular, an increasing effort has been devoted to define effective optimization techniques for XPath queries. One of the main issues related to the optimization of XPath queries is their minimization. The minimization of XPath queries has been studied for limited fragments of XPath, containing only the descendent, the child and the branch operators. In this work, we address the problem of minimizing XPath queries for a more general fragment, containing also the wildcard operator. We characterize the complexity of the minimization of XPath queries, stating that it is NP-hard, and propose an algorithm for computing minimum XPath queries. Moreover, we identify an interesting tractable case and propose an ad hoc algorithm handling the minimization of this kind of queries in polynomial time.

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

1 Introduction

Extracting information using an incomplete knowledge of the data structure is the main issue that has to be dealt with when extending classical techniques for querying databases to the field of semistructured data, and in particular of XML data. The user always knows what kind of information he is interested in, but rarely knows where this information is placed or how it is structured. Therefore, answering a query over an XML database can make it necessary to explore the data in several directions.

XML queries are usually expressed by means of XPath expressions [4], which define a way of navigating an XML tree (corresponding to some document) and return the set of nodes which are reachable from one or more starting nodes through the paths specified by the expressions.

An XPath expression can be represented graphically by means of a *tree pattern* defining some structural properties of the nodes belonging to the specified path.

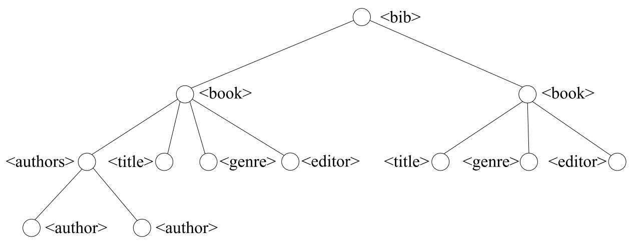

Figure 1: An XML tree

For instance, consider the document represented in Fig. 1 containing some information about a collection of books, and the query: "*find the titles of all the books for which at least one author is known*". This query can be formulated with the XPath expression `bib/book[//author]/title` which defines the following navigation: starting from an element `bib`, consider its children `book` from which we can reach an element `author` by means of any path, and return the title of these books. This expression is equivalent to the following tree pattern:

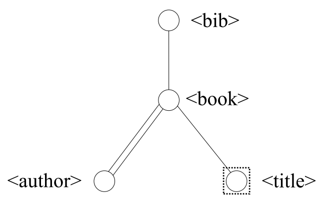

Figure 2: A tree pattern

The boxed node in the above tree pattern defines the *output node* (corresponding to the element `title`), i.e. the information that must be returned. The edge represented with a double line corresponds to the symbol '`//`' in the original expression and is called *descendant edge*. The condition on the element book (we are interested in books having at least one author) corresponds to the *branch* in the tree pattern at the node `book`. The answer to XPath queries is built by matching the tree pattern representing the query against a document. In our example, when the tree pattern is matched on the document in Fig. 1, the content of the element `title` on the left-hand side of the tree is returned.

The efficiency of the matching operation greatly depends on the size of the pattern [8], so it is crucial to have queries of minimum size. To achieve this goal we should re-formulate queries avoiding "redundant" conditions. For instance, consider the following query: "*retrieve the editors that published thrillers and whose authors have written a thriller*". Looking at the structure of this query we observe that the first condition makes the second one redundant. Thus, an equivalent (and minimal) query can be formulated as: "*retrieve the editors that published a thriller*".

Observe that the problem of minimizing the tree pattern corresponding to a given query is strictly related to the problem of checking whether there are two "subpatterns" (corresponding to some conditions on intermediate nodes) which are contained one into the other. That is, it can be reduced to finding a condition expressed in the query which can be subsumed by another condition specified in the same query. This problem is called *query containment*, and has received a great deal of attention by the research community, originally for relational queries [3, 9, 10], and, more recently, for XML queries [5, 12, 13, 15, 17].

The minimization of XPath queries was first studied in [16], where *simple* XPath expressions (i.e. without the use of the symbol '`//`') were considered. The complexity of minimizing queries expressed using this restricted fragment of XPath (called $XP^{\{/,[\,],*\}}$) was shown to be polynomial w.r.t. the size of the query. In [1], a different fragment of XPath (called $XP^{\{/,//,[\,]\}}$) has been studied, showing that queries containing the operators '`/`', '`//`', '`[]`' but without any occurrence of the wildcard symbol '`*`' can be also minimized in polynomial time (a node marked with '`*`' in a tree pattern can be matched to a node with any label in a document). More efficient algorithms for minimizing tree patterns in the same fragment $XP^{\{/,//,[\,]\}}$ have been recently proposed in [14].

We point out that the minimization problem for both the XPath fragments analyzed in [16] and [1] can be efficiently solved as: 1) it can be reduced to solve a number of instances of containment between pairs of tree patterns; 2) for these fragments, the containment between two tree patterns can be decided in polynomial time, as it can be reduced to find a *homomorphism* between them [12]. For more general fragments of XPath the containment problem is *co*NP-complete [12, 13, 17], as it cannot be reduced to find a homomorphism between two tree patterns. Moreover, the technique used in [16] and [1] for minimizing a tree pattern is based on the property that, for $XP^{\{/,[\,],*\}}$ and $XP^{\{/,//,[\,]\}}$, a tree pattern of minimum size equivalent to a given tree pattern p can be found among the subpatterns of p, i.e. it can be computed by pruning "redundant" nodes from p. The validity of this property for more general XPath fragments has never been proved.

Main Contribution. In our work we show some fundamental results on minimization:

1. we show that given a tree pattern p belonging to the fragment of XPath $XP^{\{/,//,[\,],*\}}$ (containing branches, descendant edges and the wildcard symbol), a minimum tree pattern can be found among the subpatterns of p. This result allows us to design a sound and complete algorithm for tree pattern minimization;

2. we show that the decisional problem "*given a cardinal k and a tree pattern p in $XP^{\{/,//,[\,],*\}}$, does there exist a tree pattern p' (equivalent to p) whose size is less than or equal to k?*" is coNP-complete;

3. we identify an interesting subclass of $XP^{\{/,//,[\,],*\}}$ which can be minimized efficiently (i.e. in polynomial time).

We point out that the containment problem has been already characterized for the whole $XP^{\{/,//,[\,],*\}}$, and its restricted fragments $XP^{\{/,//,[\,]\}}$, $XP^{\{/,//,*\}}$ and $XP^{\{/,[\,],*\}}$. On the other side, the complexity of the minimization problem has been characterized only for the above restricted fragments, but not for the whole $XP^{\{/,//,[\,],*\}}$.

Plan of the paper In Section 2 we introduce basic notions about tree patterns and define the notations that will be used throughout the rest of the paper. In Section 3 we illustrate in detail our goal, and state the main theoretical results of this work. In Section 4 we introduce a framework for reasoning about the minimization of XPath queries, and use it for defining an algorithm for minimization. In Section 5 we analyze the complexity of the minimization problem and, finally, in Section 6 we introduce a form of XPath expressions which can be minimized efficiently.

2 Preliminaries

In this paper we model XML documents as unordered node labelled trees over an infinite alphabet. We point out that, even if by choosing this model we disregard the order of XML documents, this is not a limitation since the fragment of XPath we use ignores the order of the document. We assume the presence of an alphabet $\mathbb{N}$ of nodes and an alphabet Σ of node labels.

Trees and Tree patterns

A *tree* t is a tuple $(r_t, N_t, E_t, \lambda_t)$, where $N_t \subseteq \mathbb{N}$ is the set of nodes, $\lambda_t : N_t \rightarrow \Sigma$ is a node labelling function, $r_t \in N_t$ is the distinguished root of t, and $E_t \subseteq N_t \times N_t$ is an (acyclic) set of edges such that starting from any node $n_i \in N_t$ it is possible to reach any other node $n_j \in N_t$, walking through a sequence of edges $e_1, \ldots, e_k$.

Given a tree $t = (r_t, N_t, E_t, \lambda_t)$, we say that a tree $t' = (r_{t'}, N_{t'}, E_{t'}, \lambda_{t'})$ is a subtree of t if the following conditions hold:

1. $N_{t'} \subseteq N_t$;
2. the edge (n_i, n_j) belongs to $E_{t'}$ iff $n_i \in N_{t'}$, $n_j \in N_{t'}$ and $(n_i, n_j) \in E_t$.

The set of trees defined on the alphabet of node labels Σ will be denoted as T_Σ.

Definition 1 *A* tree pattern *p is a pair $\langle t_p, o_p \rangle$, where:*

1. *$t_p = (r_p, N_p, E_p, \lambda_p)$ is a tree;*
2. *E_p is partitioned into the two disjoint sets C_p and D_p denoting, respectively, the* child *and* descendent *branches;*
3. *$o_p \in N_p$ is a distinguished output node* [1].

[1]We do not consider tree patterns with a set of output nodes (called *k-ary tree patterns*) since a unique output node (*unary tree patterns*) suffice to express XPath queries. However, it can be shown that the containment (and equivalence) problem between k-ary tree patterns is equivalent to the containment (and equivalence) between unary tree patterns.

Observe that, the alphabet of labels can include the wildcard symbol '*';

Given a set $\mathcal{F} \subseteq \{/, //, [\,], *\}$, we shall denote by $XP^{\mathcal{F}}$ the fragment of XPath which uses only operators in $\mathcal{F}$. The class of tree patterns used in our framework corresponds to a fragment of XPath studied in [12], denoted $XP^{\{[\,],*,/,//\}}$, consisting of the expressions which can be defined recursively by the following grammar:

$$exp \rightarrow exp/exp \mid exp//exp \mid exp[exp] \mid \sigma \mid * \mid .$$

where σ is a symbol in Σ, and the symbol '.' stands for the "*current node*".

Given an $XP^{\{/,//,[\,],*\}}$ expression e, a tree pattern p corresponding to e can be trivially defined. For instance, the XPath expression `a[b/*//c]//d` can be represented by the tree pattern shown in Fig. 3.

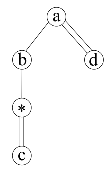

Figure 3: A pattern corresponding to `a[b/*//c]//d`

Given a tree t and a tree pattern p, an *embedding* e of p into t is a total function $e : N_p \rightarrow N_t$, such that:

1. $e(r_p) = r_t$,
2. $\forall (x, y) \in C_p$, $e(y)$ is a child of $e(x)$ in t,
3. $\forall (x, y) \in D_p$, $e(y)$ is a descendant of $e(x)$ in t, and
4. $\forall x \in N_p$, if $\lambda_p(x) = a$ (where $a \neq *$) then $\lambda_t(e(x)) = a$.

Given a tree pattern p and a tree t, $p(t)$ denotes the unary relation $p(t) = \{x \in N_t \mid \exists$ an embedding e of p into t s.t. $e(o_p) = x\}$.

Fig. 4 shows two examples of embedding of the tree pattern of Fig. 3 into two distinct trees.

Models and Canonical Models of Tree Patterns

The models of a tree pattern p defined over the alphabet Σ are the trees of T_Σ which can be embedded by p. The set of models of p is $Mod(p) = \{t \in T_\Sigma \mid p(t) \neq \emptyset\}$.

Canonical models of a tree pattern p are models having the same *shape* as p. That is, a canonical model of p is a tree which can be obtained from p by substituting descendant edges with chains of *-marked nodes of any length, and then replacing every * label (both * labels which were originally in the pattern and those which have been obtained transforming descendant edges) with any symbol in Σ. The set of canonical models of a pattern p will be denoted as $m(p)$. The subset of canonical models of p obtained by expanding descendant edges into chains of *-labelled nodes of length at most ω, and replacing the $*$ with a *new* symbol z (i.e. z is not used for labelling any node of p) will be denoted as $m^z_\omega(p)$.

In Fig. 4 two examples of model and canonical model of the tree pattern p of Fig. 3 are shown.

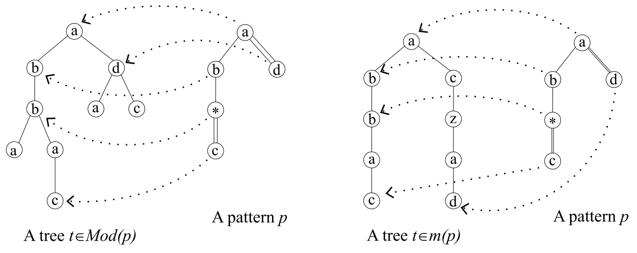

Figure 4: A model and a canonical model of a tree pattern

Containment and equivalence between tree patterns

Given two tree patterns p_1, p_2, we say that p_1 *is contained in* p_2 ($p_1 \subseteq p_2$) iff $\forall t\ p_1(t) \subseteq p_2(t)$.

We say that p_1 and p_2 are *equivalent* ($p_1 \equiv p_2$) iff $p_1 \subseteq p_2$ and $p_2 \subseteq p_1$ (i.e. $\forall t\ p_1(t) = p_2(t)$). The set of patterns which are equivalent to a given pattern p will be denoted as $Eq(p)$.

The containment and equivalence problems are basically identical (equivalence between two tree patterns is a two way containment), and their complexity has been widely studied. In the table shown in Fig. 5 we report some results about the complexity of the containment problem for some fragments of XPath.

Fragment	***Complexity***
$XP^{\{/,//,[\,],*\}}$	co-NP complete
$XP^{\{/,//,*\}}$	P
$XP^{\{/,[\,],*\}}$	P
$XP^{\{/,//,[\,]\}}$	P

Figure 5: Complexity of the containment problem

An important result on containment which is not reported in the above table is that for a pair of subpattern $p \in XP^{\{/,//,*\}}$ and $q \in XP^{\{/,//,[\,],*\}}$ checking whether $p \supseteq q$ can be done in polynomial time [12].

Boolean tree patterns

A boolean tree pattern is a "*nullary*" tree pattern, that is a tree pattern with no output node. A pattern p with this property is called "boolean" since $p(t)$ can be seen as a boolean function which evaluates to *true* if an embedding between p and t exists (the *true* value corresponds to a set with an empty singleton), *false* otherwise (the *false* value corresponds to an empty set).

The notions of model and canonical model can be extended to boolean tree patterns. That is, the models of a boolean tree pattern p are the trees of T_Σ on which p evaluates to *true*: $Mod(p) = \{t \in T_\Sigma | p(t) \text{ is true}\}$. Analogously, the canonical models of p are models having the same "shape" as p.

Also the notions of containment and equivalence between tree patterns can be trivially extended to boolean patterns. In particular, for boolean tree patterns the containment problem reduces to implication: $p_1 \subseteq p_2$ iff $\forall t\ p_1(t) \Rightarrow p_2(t)$, whereas the equivalence problem reduces to verifying whether $\forall t\ p_1(t) \equiv p_2(t)$.

As shown in [12], the containment and equivalence problems for "general" tree patterns and boolean tree patterns are equivalent. That is, two tree patterns p_1, p_2 can be always translated into two boolean patterns p'_1, p'_2 such that $p_1 \subset p_2$ iff $p'_1 \subset p'_2$. The same property holds for the equivalence problem, which can be seen as a two way containment. Therefore, for the sake of simplicity, in the following we will use boolean tree patterns for studying the tree pattern minimization problem (we shall not care about output nodes).

Given a boolean tree pattern $p = \langle t_p, \emptyset \rangle$, we say that the boolean tree pattern $p' = \langle t_{p'}, \emptyset \rangle$ is a subpattern of p if the following conditions hold:

1. $N_{p'} \subseteq N_p$;
2. the edge (n_i, n_j) belongs to $C_{p'}$ iff $n_i \in N_{p'}$, $n_j \in N_{p'}$ and $(n_i, n_j) \in C_p$;
3. the edge (n_i, n_j) belongs to $D_{p'}$ iff $n_i \in N_{p'}$, $n_j \in N_{p'}$ and $(n_i, n_j) \in D_p$;

Given a pattern p, we define $size(p) = |N_p|$ and $minsize(p) = min_{p' \in Eq(p)}(size(p'))$.

Notations on tree patterns

In the following we denote the subpattern of p rooted in any node n and containing all the descendant nodes of n as sp_n. The following figure explains this notation.

Moreover, given a tree pattern p whose root r has m children $c_1, \ldots, c_m$, we will denote as $sp_1, \ldots, sp_m$ the subpatterns $sp_{c_1}, \ldots, sp_{c_m}$ (i.e. the subpatterns of p directly connected to r by either a child or descendant edge containing all the descendant nodes). We will denote as p_j the subpattern of p obtained from sp_j adding r to sp_j and connecting it to the root of sp_j in same way as it was connected in p. Obviously, for any pair p_i and sp_i it holds that $minsize(p_i) =$

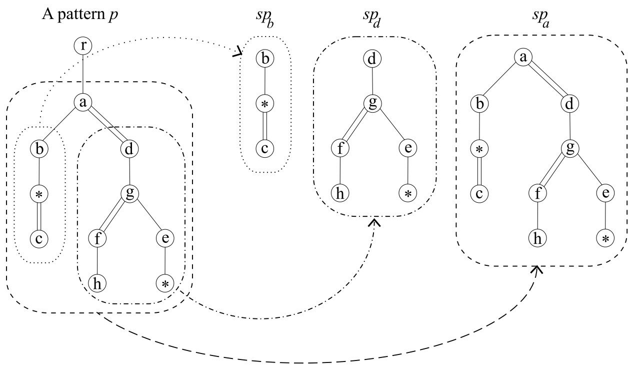

Figure 6: A pattern p and its subpatterns sp_b, sp_d, sp_a

$minsize(sp_i) + 1$. Furthermore, we will denote as $SP(p)$ and $P(p)$, respectively, the sets of all the sp_i and p_i in p.

Fig. 7 shows the meaning of this notation for a tree pattern p whose root has 2 children. In the examined case, $P(p) = \{p_1, p_2\}$ and $SP(p) = \{sp_1, sp_2\}$.

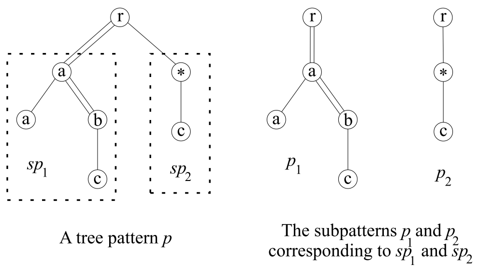
sp_1
sp_2
p_1
p_2
A tree pattern p
The subpatterns p_1 and p_2 corresponding to sp_1 and sp_2

Figure 7: Examples of subpatterns

Given a pattern p and a subpattern p' of p, we denote as $p - p'$ the pattern obtained from p by pruning p'. Given a pattern p and a node n of p, we denote as $p - n$ the pattern obtained from p after pruning the subpattern sp_n. Analogously, given a set of nodes $N' = \{n_1, \ldots, n_l\}$, we denote as $p - N'$ the pattern obtained from p by pruning all the subpatterns $sp_{n_1}, \ldots, sp_{n_l}$. Finally, given a set of positive integers $X = \{x_1, \ldots, x_l\}$, we denote as $p - X$ the pattern obtained from p by pruning all the subpatterns $sp_{x_1}, \ldots, sp_{x_l}$.

Reasoning about containment using models

We can reason about containment between tree patterns using the notions of model and canonical model described above. In particular, the following result holds [12]:

Fact 1 *For any (boolean) tree patterns p and q, the following assertions are equivalent: 1) $p \subseteq q$, 2) $Mod(p) \subseteq Mod(q)$, 3) $m(p) \subseteq Mod(q)$, 4) $m^z_\omega(p) \subseteq Mod(q)$, where ω is one plus the longest chain of *-labelled nodes not containing descendant edges in q.*

As $m^z_\omega(p)$ is a finite set of trees, the equivalence between 1) and 4) permits us to test the containment $p \subseteq q$ by generating all the trees in $m^z_\omega(p)$ and checking whether they all belong to $Mod(q)$.

3 Problem statement

The problem of minimizing a tree pattern can be formulated as follows:

Given a tree pattern p, construct a tree pattern p_{min} which is equivalent to p and having minimum size (i.e. $size(p_{min}) = minsize(p)$).

This problem has been recently investigated for different fragments of XPath expressions. In particular, in [16] it has been shown that the tree pattern minimization problem can be solved in polynomial time for $XP^{\{[\,],*\}}$, and in [1] a sound and complete polynomial time algorithm minimizing a pattern in $XP^{\{[\,],//\}}$ has been defined.

The latter cases are tractable as, for the above fragments of XPath, the following two properties hold:

1. a minimum size tree pattern equivalent to p can be found among the subpatterns of p;

2. the containment between two tree patterns p, q ($p \subseteq q$) is equivalent to the problem of finding a *homomorphism* from q to p. A homomorphism h from a pattern q to a pattern p is a total mapping from the nodes of q to the nodes of p such that:
   - h preserves node types (i.e. $\forall u \in N_q\ \lambda_q(u) \neq$ '*' $\Rightarrow \lambda_q(u) = \lambda_p(h(u))$);
   - h preserves structural relationships (i.e. whenever v is a child (resp. descendant) of u in q, $h(v)$ is a child (resp. descendant) of $h(u)$ in p).

The former property ensures that a tree pattern of minimum size can be obtained from p by "pruning" some of its nodes, until no node can be further removed preserving the equivalence of the obtained pattern w.r.t. p. The latter property can be used for checking whether a node of a pattern is redundant (i.e. it can be removed) efficiently, as finding a homomorphism can be done in polynomial time. Fig. 8 shows a pair of patterns p, q such that there exists a homomorphism from q to p proving that $p \subseteq q$.

Unfortunately, property 2 does not hold for more general XPath fragments. In particular, for $XP^{\{/,//,[\,],*\}}$, the existence of a homomorphism between q and p suffices for asserting $p \subseteq q$, but is not a necessary condition. Fig. 9 shows a pair of patterns

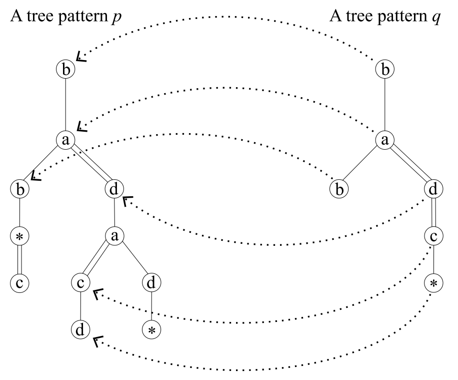

Figure 8: A homomorphism between two tree patterns

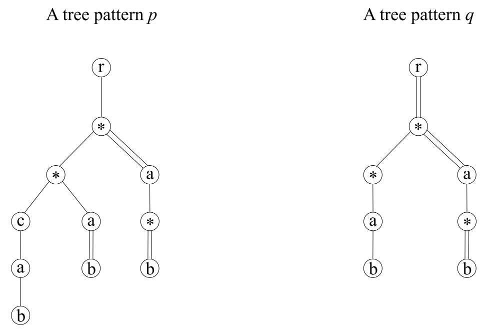

Figure 9: Two tree patterns which are not related via a homomorphism

p, q such that no homomorphism from q to p exists, although p is contained into q.

No homomorphism between the tree patterns q and p of Fig. 9 exists as, even if the right-hand side branch of q can be mapped onto the right-hand side branch of p, the node b of the left-hand side branch of q cannot be mapped onto any node of p.

However, it can be proved that, although no homomorphism between p and q exists, $p \subseteq q$ holds. In fact, it is easy to see that any canonical model t of p is a model of q. This can be shown by considering that canonical models of p are obtained by expanding descendant edges of p into (possibly empty) chains of *-labelled edges, and then “reasoning by cases”.

Canonical models of p obtained by expanding the descendant edge connecting the nodes a and b in the left-hand side branch of p into a child edge can be embedded by q in the same way as the tree t' in Fig. 10. In this case, q maps the portion of t' corresponding to the “right-hand side” portion of p. Otherwise, when expanding the descendant edge connecting the same pair of nodes a and b into a chain of at least one *-labelled node, the canonical model can be embedded by q in the same way as the tree t'' in Fig. 10. In this case, q maps the portion of t' corresponding to the “right-hand side” portion of p. This phenomenon can be seen as a form of disjunction, which is not caught by homomorphism and makes the containment problem harder.

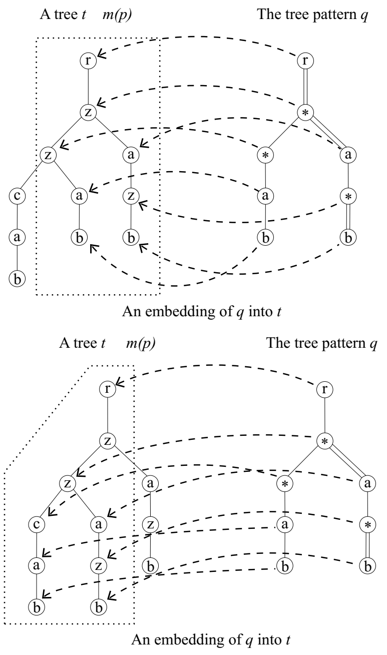

Figure 10: Embedding of the pattern q of Fig. 9

Thus, deciding whether $p \subseteq q$ by searching for a homomorphism between q and p leads to a sound but not complete algorithm.

As regards property 1, observe that, if property 1 does not hold we cannot minimize a query by simply pruning some of its parts, since it is necessary to consider also queries having a completely different structure. For instance, consider the query $Q = (\sigma_{Name=`a'}R \cup \sigma_{Name=`b'}R)$ on a relational schema $\{R\}$ expressed using named relational algebra. This query can be completely reformulated, changing the selection condition, obtaining a query $Q' = (\sigma_{Name=`a' \vee Name=`b'}R)$, that requires only one selection operation. Obviously the possibility of obtaining in this way a minimum size query makes the minimization problem harder.

The work of [12] shows that the presence of both // and $*$ in our XPath fragment adds a limited form of disjunction. Indeed, they show that, considering the XPath fragment $XP^{\{/,//,[\,],*\}}$, the containment of a pattern p in the disjunction of patterns $p_1, \ldots, p_k$ can be reduced to the containment of two pattern p' and p''. Thus, while for both the fragment $XP^{\{/,[\,],*\}}$ (analyzed in [16]) and $XP^{\{/,//,[\,]\}}$ (analyzed in [1]) it is easy to show that property 1 holds, it is not straightforward to prove that the same property holds for the fragment $XP^{\{/,//,[\,],*\}}$.

In our work we provide two main contributions:

1. we show that property 1 still holds for $XP^{\{/,//,[\,],*\}}$;

2. on the basis of the latter property, we investigate the problem of minimizing tree patterns in $XP^{\{/,//,[\,],*\}}$, and show that it is NP-hard[2].

Moreover, we provide an algorithm for minimizing a tree pattern.

4 A framework for minimizing XPath queries

In this section we provide two fundamental contributions. First, we prove that property 1 holds for $XP^{\{/,//,[\,],*\}}$ (i.e. a minimum size tree pattern equivalent to $p \in XP^{\{/,//,[\,],*\}}$ can be found among the subpatterns of p). Then, we define an algorithm for minimizing a tree pattern query.

In order to prove that property 1 still holds for $XP^{\{/,//,[\,],*\}}$ we have to introduce various lemmas. Although the "partial" results stated in these lemmas are not of practical use for the definition of algorithms for minimization of tree patterns, they constitute a general framework for reasoning about tree patterns.

Our first result regards the containment of two patterns p and q. In particular, we prove that if p is contained in q, then each subpattern q_j of q contains at least one subpattern p_i of p.

Lemma 1 *Let p and q be two patterns with root r, such that $p \subseteq q$. Then, for each subpattern $q_j \in P(q)$ there exists a subpattern $p_i \in P(p)$ s.t. $p_i \subseteq q_j$.*

An application of Lemma 1 is sketched in Fig. 1, where two patterns p, q s.t. $p \subseteq q$ are reported. In this case, q_1 contains p_3, q_2 contains p_3, and q_1 contains p_1. Note that the subpattern p_2 of p is not contained in any subpattern q_i of q.

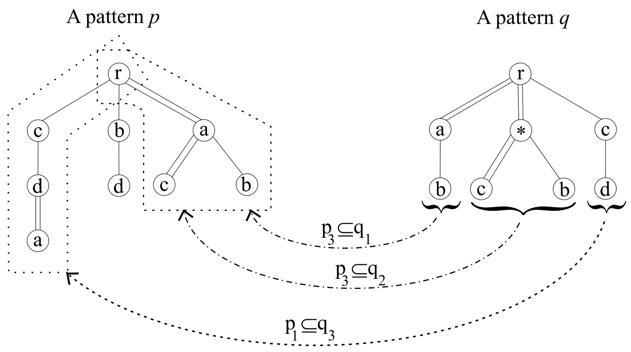

Figure 11: Explaining Lemma 1

[2]A more precise characterization of the optimization problem states that it is in FP^{NP}

The above lemma allows us to reason about the containment of two patterns in terms of the containment of their subpatterns. We can use this lemma to derive a first result about equivalent patterns: If two patterns p and q are equivalent, but the root of p has more children than the root of q, then some subpatterns p_i are "redundant".

Lemma 2 *Let p and q be two patterns rooted in r s.t. $p \equiv q$, and let m and n, with $m > n$, be the number of children of r in p and, respectively, q. Then, there exists a set $S \subset SP(p)$ consisting of $m-n$ subpatterns sp_i such that $p - S \equiv p$.*

The above lemma can be applied to the patterns p and q of Fig. 12. These two patterns are equivalent, but the root of p has more children than the root of q. As stated by Lemma 2 one of the subpatterns p_i is redundant. In this case, the subpattern p_2 can be removed from p obtaining an equivalent subpattern.

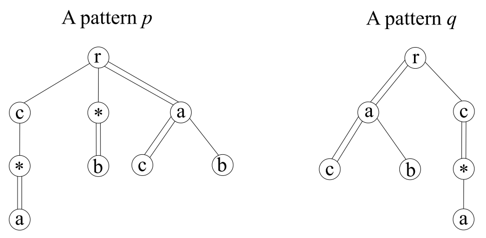

Figure 12: Two equivalent patterns with a different "shape"

The following Lemma states another important result. It implies that all the patterns which have minimum size and are equivalent to a given pattern p have a common structural property: their roots have the same number of children.

Lemma 3 *Let p and q be two equivalent patterns rooted in r having the same number of child and descendant nodes of r, and let q be of minimum size. Then, there not exists a subpattern $sp_k \in SP(p)$ such that $p - sp_k \equiv p$.*

The above Lemma states that, if a pattern has minimum size, the conditions expressed by the subpatterns connected to its root cannot be expressed using a smaller set of subpatterns (i.e. conditions). The following Lemma strengthens this result, as it ensures that, given a pair of patterns p, q such that $p \equiv q$ and q has minimum size, if the root of p has the same number of children as the root of q, then every subpattern p_i expresses a condition equivalent to some subpattern q_j in q. This result makes it possible to associate each p_i in p with a unique q_j in q.

Lemma 4 *Let p and q be two equivalent patterns whose roots have the same number of child and descendant nodes, and let q be of minimum size. For each subpattern $p_i \in P(p)$ there exists a unique subpattern $q_j \in P(q)$ directly connected to r_q s.t $p_i \equiv q_j$.*

Another important result regarding patterns minimality is stated by the following Lemma, which indicates the conditions that might lead a pattern to be not minimal. More formally, a pattern has not minimum size if at least one of its subpatterns p_i is redundant (i.e. it expresses a condition which can be subsumed by another subpattern p_j) or has not minimum size (i.e. the conditions expressed by this subpattern can be reformulated in a more coincise form).

Lemma 5 *A pattern p in $XP^{\{[\,],/,//,*\}}$ is not of minimum size iff at least one of the following conditions hold:*

1. *there exists a pair of subpatterns p_i,p_j s.t. $p_i \subseteq p_j$;*
2. *there exists a subpattern p_i of p which is not of minimum size.*

The above lemmas suffice to show the following theorem, which states that, given a tree pattern p, a pattern $p_{min} \in Eq(p)$ can be found among the subpatterns of p.

Theorem 1 *Given a pattern p in $XP^{\{/,//,[\,],*\}}$ if $minsize(p) = k$ then there exists a subpattern p_{min} of p such that $p \equiv p_{min}$ and $size(p_{min}) = k$.*

Proof. As $minsize(p) < size(p)$, from Lemma 5 we have that either there exists at least a pair of subpatterns p_i, p_j s.t. $p_i \subseteq p_j$, or there exists at least one subpattern p_i of p which is not minimum. Therefore we can remove from p all the subpatterns sp_j (corresponding to some p_j containing another p_i) thus obtaining a subpattern p' which is equivalent to p. The subpattern p' can possibly coincide to p if there weren't any pairs p_i, p_j s.t. $p_i \subseteq p_j$.

If, after pruning all the redundant subpatterns, $minsize(p) = size(p')$ then we have proven the theorem, as p' is a subpattern of p and has minimum size. Otherwise, from Lemma 5 we know that, as there is no pair p'_i, p'_j s.t. $p'_i \subseteq p'_j$, there exists a set $NotMin$ (with cardinality at least 1) of subpatterns p'_i of p' which are not minimum. Each of these subpatterns consists of a tree pattern having the same root as p', and whose root is connected to a unique child. It is trivial to show that each p'_i is not minimum iff sp'_i (obtained from p'_i removing its root) is not minimum. We can apply iteratively the same reasoning to each non minimum $sp'_i \in NotMin$, replacing it in p' with a minimum subpattern sp''_i of sp'_i obtained from sp'_i as shown above. At the end of this process, p' will be a subpattern of p s.t. neither there is a pair p_i, p_j s.t. $p_i \subseteq p_j$, nor there is one subpattern p_i of p which is not minimum. Therefore, for Lemma 5, p' has minimum size. □

4.1 An Algorithm for tree pattern minimization

Theorem 1 suggests a technique for minimizing a tree pattern, as it states that a minimum tree pattern equivalent to a given tree pattern p can be found among the subpatterns of p. The following algorithm implements the idea used for the proof of Theorem 1.

Algorithm 1 works as follows. First, it checks whether there is any subpattern p_i of p which is "redundant" w.r.t. the remainder of p. That is, it checks whether $p_i \supseteq p - sp_i$, where sp_i is obtained from p_i by removing its root, for each p_i. Then, if such a "redundant" pattern is found, it is removed from p. After removing all the "redundant" subpatterns in $SP(p)$, the algorithm is recursively executed on the not pruned subpatterns sp_i. Finally, every minimized pattern sp_i^{min} is connected to the root in the same way as the corresponding sp_i was connected to the root using the function *assemble*.

```
Algorithm 1
FUNCTION Minimize
Input:   $p$ (a tree pattern)
Output:  $p_{min}$ (a minimum tree pattern equivalent to $p$)
begin
  $p_{min} = p$;
  For each $p_i \in P(p_{min})$ do
    if ($p_i \supseteq p_{min} - sp_i$)
      $p_{min} = p_{min} - sp_i$;
  $SP_{new} = \emptyset$;
  For each $sp_i \in SP(p_{min})$ do
    $SP_{new} = SP_{new} \cup Minimize(sp_i)$;
  $p_{min} = assemble(p_{min}, SP_{new})$;
  return $p_{min}$;
end
```

Figure 13: An algorithm minimizing a tree pattern

For deciding the containment between pairs of patterns we can use the sound and complete algorithm introduced in [12], that is to our knowledge the only one defined for the fragment $XP^{\{/,//,[\,],*\}}$. In the latter work an upper bound on the complexity of this algorithm has been stated: given two patterns $p, p' \in XP^{\{/,//,[\,],*\}}$ deciding whether $p \subseteq p'$ requires at most $O(|p| \cdot |p'| \cdot (w' + 1)^{d+1})$ steps, where $|p|$ is the size of p, $|p'|$ is the size of p', d is the number of descendant edges in p and w' is one plus the longest chain of '$*$' in p'.

Using this result, we can state an upper bound for the complexity of Algorithm 1. We denote the number of branches of p as b, the maximum degree of any node of p as r, the length of the longest chain of '$*$' in p plus one as w, and the number of descendant edges of p as d.

Proposition 1 (Upper bound) *Algorithm 1 works in* $O(b \cdot r \cdot |p|^2 \cdot (w+1)^{d+1})$.

Proof. For each branching node b_i of p, Algorithm 1 calls the subroutine for checking containment as many times as the number of children of b_i. Therefore, the algorithm performs at most $b \cdot r$ containment checking step, and each of these steps has a cost bounded by $O(|p|^2 \cdot (w+1)^{d+1})$ (as shown in [12]). □

Observe that the efficiency of Algorithm 1 can be improved by speeding up the containment test. Lemma 1 ensures that checking if $p_i \supseteq p - sp_i$ is equivalent to testing the containment of p_i in any of the subpatterns p_j with $j \neq i$, that is: $p_i \supseteq p - sp_i$ iff $\exists j \neq i | p_i \supseteq p_j$. The upper bound on the number of operations that must be executed using this strategy is smaller than the upper bound expected on the number of operations that should be performed if the containment test were executed between p_i and the whole $p - sp_i$. We can show this, for the sake of simplicity, considering a pattern p consisting of three tree patterns p_1, p_2, p_3, such as the one on the left-hand side of Fig. 12. We check whether p_1 is redundant using both the two described approaches. First, we check whether $p_1 \supseteq p_{2,3}$, where $p_{2,3} = p - p_1$; then, we decide $p_1 \supseteq p_2$ and $p1 \supseteq p_3$ separately. In former case we have the following bound: $B_1 = |p_1| \cdot |p_{2,3}| \cdot (w_1+1)^{d_{23}+1}$. In the latter case, checking the containment will have the following bound: $B_2 = |p_1| \cdot |p_2| \cdot (w_1+1)^{d_2+1} + |p_1| \cdot |p_3| \cdot (w_1+1)^{d_3+1}$, where $d_{23} = d_2 + d_3$. It is easy to prove that this bound is better than the first one. In fact, $B_2 = \frac{|p_1| \cdot |p_2| \cdot (w_1+1)^{d_{23}+1}}{(w_1+1)^{d_3}} + \frac{|p_1| \cdot |p_3| \cdot (w_1+1)^{d_{23}+1}}{(w_1+1)^{d_2}} \leq \frac{|p_1| \cdot |p_2| \cdot (w_1+1)^{d_{23}+1} + |p_1| \cdot |p_3| \cdot (w_1+1)^{d_{23}+1}}{(w_1+1)^{d_{min}}} = \frac{|p_1| \cdot (w_1+1)^{d_{23}+1} \cdot (|p_2|+|p_3|)}{(w_1+1)^{d_{min}}} = \frac{|p_1| \cdot (w_1+1)^{d_{23}+1} \cdot (|p_{23}|+1)}{(w_1+1)^{d_{min}}} = \frac{B_1}{(w_1+1)^{d_{min}}} + \frac{|p_1| \cdot (w_1+1)^{d_{23}+1}}{(w_1+1)^{d_{min}}} = \frac{B_1}{(w_1+1)^{d_{min}}} + \frac{B_1}{|p_{23}| \cdot (w_1+1)^{d_{min}}} = B_1 \cdot \frac{|p_{23}|+1}{|p_{23}| \cdot ((w_1+1)^{d_{min}})} \leq B_1$, where $d_{min} = min\{d_2, d_3\}$.

The above considerations can be easily extended to a pattern p consisting of a generic number of patterns $p_1, p_2, \ldots, p_n$.

Remark We point out that Algorithm 1 is based on a top-down strategy. Obviously, we could define an analogous algorithm for minimization based on a bottom-up approach. However the asymptotic complexity would not change. The main difference between the two approaches is that using a bottom-up strategy we are guaranteed that when we test the containment between two subpatterns, these subpatterns have minimum size. As the cost of deciding the containment between two patterns depends on their size, this could possibly lead to an improvement of efficiency. However, if a subpattern is redundant and is rooted "closely" to the root, then Algorithm 1 removes it without performing any minimization step. In contrast, a bottom-up algorithm would first minimize this subpattern and then check whether it is redundant. This strategy can be inefficient, especially when the redundant subpattern is already of minimum size.

An algorithm exploiting a bottom-up strategy for minimization is given in Section 6. This algorithm is specialized for the minimization of a particular form of tree patterns, for which the bottom-up strategy is optimal.

5 Complexity results

Algorithm 1 works in exponential time w.r.t. the size of the pattern to be minimized, as stated in Proposition 1. In this section we analyze the complexity of the problem of minimizing XPath queries in $XP^{\{/,//,[\,],*\}}$, showing that unfortunately it is not possible to define an algorithm performing much better than ours. In fact, we will show that the decisional problem "given a cardinal k and a tree pattern p in $XP^{\{/,//,[\,],*\}}$, does there exist a tree pattern p' (equivalent to p) whose size is less than or equal to k?" is coNP-complete. In order to characterize the complexity of this problem, we first characterize the complexity of the following decisional problem.

Lemma 6 *Let p be a pattern in $XP^{\{/,//,[\,],*\}}$ and k a positive integer. The problem of testing if $minsize(p) > k$ is NP-complete.*

Proof. (Sketch)
(Membership) Due to space limitations, we only provide the intuition underlying this part of the proof. A polynomial size certificate proving that $minsize(p) > k$ should contain a set X of k nodes of p that cannot be removed obtaining an equivalent pattern. However, verifying whether this set of nodes can be removed from p yielding an equivalent pattern cannot be done in polynomial time, as checking the equivalence between patterns is in coNP. Therefore, the certificate should contain for each node $x \in X$ a "sub-certificate" (a canonical model of p) showing that the tree pattern

obtained after removing x is more general than the original tree pattern (i.e. $p \in m(p) - Mod(p - sp_x)$).
(Completeness) We prove that the problem is complete for the class NP by showing a reduction of the problem of checking that a pattern q_1 is not contained into a pattern q_2.

Given two patterns q_1 and q_2, we build a pattern p that consists of two chains of n nodes both attached to the root of p. The nodes of the first chain are all labelled with a new symbol "x", whereas the nodes of the second chain are labelled with "*". We attach the pattern q_1 at the end of the first chain, and the pattern q_2 at the end of the second chain, as shown in Fig. 14.

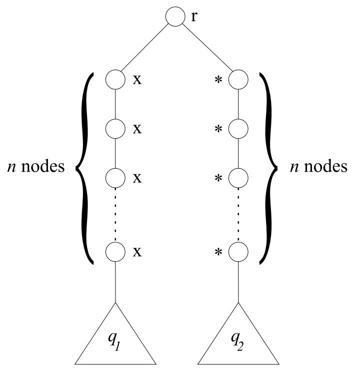

Figure 14: The tree pattern p

We choose $n > 2\ max(size(q_1), size(q_2))$ and test whether $minsize(p) \geq 2 \cdot n$. Clearly, $minsize(p) > 2 \cdot n$ iff $q_1 \not\subseteq q_2$. Indeed, if $q_1 \not\subseteq q_2$ then $p_1 \not\subseteq p_2$. Furthermore $p_2 \not\subseteq p_1$ by construction, since p_2 consists of a chain of $*$ nodes and p_1 consists of a chain of nodes labelled with the symbol "x". This implies that neither sp_1 nor sp_2 can be removed from p yielding an equivalent pattern, and, since $minsize(sp_1) > n$ and $minsize(sp_2) > n$, then $minsize(p) \geq 1 + minsize(sp_1) + minsize(sp_2) > 2 \cdot n$. Suppose now that $q_1 \subseteq q_2$ this implies that $p_1 \subseteq p_2$ and then $p_2 \equiv p$. Thus $minsize(p) < 2 \cdot n$ since $size(p_2) \leq n + size(q_2)$, and $size(q_2) \leq \frac{n}{2}$. □

Theorem 2 *Let p be a pattern in $XP^{\{/,//,[\ ],*\}}$ and k a positive integer. The problem of testing if there exits a pattern p' equivalent to p such that $size(p') \leq k$ is $coNP$-complete.*

Proof. (Sketch) It straightforwardly follows from Lemma 6 and Theorem 1. □

6 Tractability Results

Theorem 6 states that the problem of minimizing a tree pattern query in $XP^{\{/,//,[\ ],*\}}$ is NP-Hard. In this section we will discuss a form of tree pattern queries which can be minimized efficiently (i.e. in polynomial time). That is, we will describe some limitations on the "shape" of a tree pattern which make this problem easier.

Definition 2 *A* limited branched *tree pattern p is a tree pattern in $XP^{\{/,//,[\ ],*\}}$ such that:*

1. *every non leaf node of p may have any number of children;*

2. *if a node n has k children $n_1, \ldots, n_k$, then at least $k - 1$ of the patterns sp_{n_i} (where $i \in [1..k]$) are linear (i.e. $sp_{n_i} \in XP^{\{/,//,*\}}$).*

In the following figure we show some examples of patterns satisfying Definition 2.

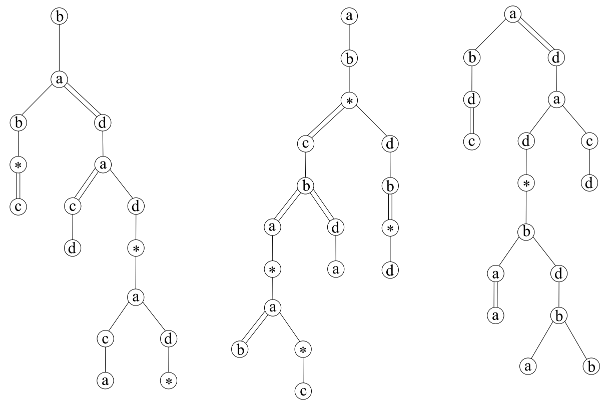

Figure 15: Tree patterns satisfying the normal form of Definition 15

The three patterns in Fig. 15 correspond, respectively, to the following expressions:

1. `b/a[b/*//c]//d/a[d/*/a[c/a]/d/*]//c/d`;

2. `a/b/*[d/b//*/d]//c/b[//d/a]//a/*/a[*/c]//b`;

3. `a[b/d//c]//d/a[c/d]/d/*/b[a//a]/d/b[a]/b`;

Theorem 3 *Let p be a limited branched tree pattern. A minimum pattern p_{min} equivalent to p can be found in polynomial time (w.r.t. the size of p).*

Proof. Lemma 5 implies that p can be minimized by checking the containment between each subpattern rooted in a branching node and the other subpatterns rooted in the same node, for every branching node. Let $b_1, \ldots, b_m$ be the m branching nodes of p ordered according to their depth (i.e. b_1 is the nearest to the root, whereas b_m is the deepest).

We can minimize p starting from b_m. This node is the root of only linear subpatterns. Applying Lemma 5 on sp_{b_m} we have that the subpattern sp_{b_m} can be minimized in polynomial time, as 1) linear patterns have minimum size, and 2) the containment between pairs of linear patterns can be decided in polynomial time (see [12]). Let $sp_{b_m}^{min}$ be a pattern of minimum size equivalent to sp_{b_m}, and let p^1 be the pattern obtained from p by replacing sp_{b_m} with $sp_{b_m}^{min}$.

Next, we consider b_{m-1} in p^1. The pattern $sp_{b_{m-1}}$ consists of k subpatterns such that $k-1$ of these subpatterns are linear and the remainder one is composed of a linear pattern connecting b_{m-1} to $sp_{b_m}^{min}$. From Lemma 5 we have that sp_{b_m} can be minimized in polynomial time, as 1) linear patterns have minimum size, 2) the subpattern consisting of a linear pattern connecting b_{m-1} to $sp_{b_m}^{min}$ has minimum size, and 3) the containment between a linear pattern and a pattern in $XP^{\{/,//,[\,],*\}}$ can be decided in polynomial time (see [12]).

We can apply the same reasoning iteratively. After the m-th iteration we have a pattern $p^m \in Eq(p)$ having minimum size. □

Following the schema of the proof of Theorem 3 we can define an algorithm which minimizes a limited branched pattern efficiently. This algorithm is shown in Fig. 16.

```
Algorithm 2
FUNCTION Minimize
Input:   p (a bounded branched tree pattern)
Output:  $p_{min}$ (a minimum tree pattern equivalent to p)
begin
  $p_{min} = p$;
  $B = \{b_1, \ldots, b_m\}$; //the set of branching nodes of p
  while $(B \neq \emptyset)$
    $b = deepest(B)$;
    $q = sp_b$;
    $Red_q = \emptyset$;    //"redundant" subpatterns of q;
  For each $q_i \in P(q)$ do
    For each $q_j \in P(q)$ do
      if $(i \neq j) \wedge (q_i \text{ is linear}) \wedge (q_j \notin Red) \wedge (q_i \supseteq q_j)$
        $Red_q = Red_q \cup \{q_i\}$;
    $q = q - Red_q$;
    $p_{min} = replace(p_{min}, sp_b, q)$;
    $B = B - \{b\}$;
  end while;
  return $p_{min}$;
end
```

Figure 16: An algorithm minimizing a limited branched tree pattern

We point out that Algorithm 2 has some differences w.r.t the algorithm for minimization presented in the previous section. In fact, it is based on a bottom-up schema. Instead of visiting the pattern starting from the root, it considers all of its branching nodes starting from the deepest one. Therefore, at each step it operates on patterns of minimum size (every subpattern rooted in a branching node either is linear or has been minimized at some previous step), so that it must never decide the containment of a linear pattern into a non linear one (as a non linear pattern of minimum size can never contain a linear one). Viceversa, it must decide the containment between linear patterns and possibly the containment of non linear patterns into linear ones, which can be done in polynomial time (as shown in [12]). If we used Algorithm 1 for minimizing a limited branched tree pattern, we should possibly check the containment between linear patterns and non linear ones in both directions, so we could not be guaranteed on the polynomial bound.

7 Conclusions and Future Works

In this paper we have studied the minimization problem for tree patterns belonging to the fragment of XPath $XP^{\{/,//,[\,],*\}}$ (i.e. the fragment containing branches, descendant edges and the wildcard symbol) and have provided some relevant contributions. First, we have proved the *global minimality* property: a minimum tree pattern equivalent to a given tree pattern p can be found among the subpatterns of p, and thus obtained by pruning "redundant" branches from p. On the basis of this result, we have designed a sound and complete algorithm for tree pattern minimization which works, in the general case, in time exponential w.r.t. the size of the input tree pattern. Secondly, we have characterized the complexity of the minimization problem, showing that the corresponding decisional problem is $coNP$-complete, and have studied a "tractable" form of tree pattern which can be minimized in polynomial time, providing an ad-hoc algorithm for the efficient minimization of this class of tree patterns.

Currently, we are investigating the possibility to extend our minimization framework to deal with XPath queries that must satisfy some constraints such as join conditions on tree pattern nodes. An example of join condition is shown on the left-hand side of Fig. 17. In this case, the join condition involves the two nodes of p with label a and says that they should be the same node. The tree pattern p_{min} on the right-hand side of Fig. 17 is a minimum tree pattern equivalent to p, but it is not a sub pattern of p. Therefore, the introduction of these constraints makes the minimization problem harder, as the global minimality property does not hold.

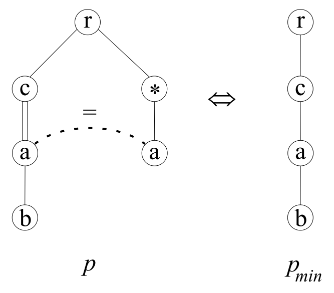

Figure 17: Two equivalent tree patterns

References

[1] S. Amer-Yahia, S. Cho, L. K. S. Lakshmanan, D. Srivastava, Minimization of tree pattern queries, *Proc. of the 2001 ACM SIGMOD Conf. on Management of Data*, Santa Barbara, California, USA, May 21-24, 2001.

[2] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Y. Vardi, Containment of Conjunctive Regular Path Queries with Inverse, *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, Breckenridge, Colorado, USA, April 11-15, 2000.

[3] A. K. Chandra, P. M. Merlin, Optimal implementation of conjunctive queries in relational databases, *Proc. of ACM Symp. on Theory of Computing (STOC)*, Boulder, Colorado, USA, May 2-4, 1977.

[4] J. Clark, XML path language (XPath), *http://www.w3.org/TR/xpath.*

[5] A. Deutsch, V. Tannen, Containment and Integrity constraints for Xpath fragments, *Proc. of the 8th Int. Work. on Knowledge Representation meets Databases (KRDB)*, Rome, Italy, September 15, 2001.

[6] A. Deutsch, V. Tannen, Reformulation of XML Queries and Constraints, *Proc. of the 9th Int. Conf on Database Theory (ICDT)*, Siena, Italy, January 8-10, 2003.

[7] D. Florescu, A. Levy, D. Suciu, Query containment for disjunctive queries with regular expressions, *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, Seattle, Washington, June 1-3, 1998.

[8] G. Gottlob, C. Koch, R. Pichler, Efficient algorithms for processing XPath queries, *Proc. of the 28th International Conference on Very Large Data Bases (VLDB)* Hong Kong, China August 20-23, 2002.

[9] P. G. Kolaitis, M. Vardi, Conjunctive-query containment and constraint satisfaction, *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, Seattle, Washington, USA, June 1-3, 1998.

[10] P. G. Kolaitis, D.L. Martin, M.N. Thakur, On the complexity of the containment problem for conjunctive queries with built-in predicates, *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, Seattle, Washington, USA, June 1-3, 1998.

[11] A.Y. Levy, D. Suciu, Deciding containment for queries with complex objects, *Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, May 12-14, 1997, Tucson, Arizona.

[12] G. Miklau, D. Suciu, Containment and Equivalence for an XPath Fragment, *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, Madison, Wisconsin, USA, June 3-5, 2002.

[13] F. Neven, T. Schwentick, XPath Containment in the Presence of Disjunction, DTDs, and Variables, *Proc. of the 9th Int. Conf on Database Theory (ICDT)*, Siena, Italy, January 8-10, 2003.

[14] P. Ramanan, Efficient algorithms for minimizing tree pattern queries, *Proc. of the 2002 ACM SIGMOD Int. Conf. on Management of Data*, Madison, Wisconsin, June 3-6, 2002.

[15] P. T. Wood, On the equivalence of XML patterns, *Proc. of the 1st Int. Conf. on Computational Logic (CL)*, London, UK, July 24-28, 2000.

[16] P. T. Wood, Minimizing simple xpath expressions, *Proc. of the 4th Int. Workshop on the Web and Databases (WebDB)*, Santa Barbara, California, USA, May 21-24, 2001.

[17] P. T. Wood, Containment for XPath Fragments under DTD Constraints, *Proc. of the 9th Int. Conf. on Database Theory (ICDT)*, Siena, Italy, January 8-10, 2003.