Rewriting XPath Queries Using Materialized Views

Wanhong Xu Z. Meral Özsoyoglu

Center for Computational Genomics
Department of Electrical Engineering and Computer Science
Case Western Reserve University, Cleveland, OH
{wanhong, meral}@case.edu

Abstract

As a simple XML query language but with enough expressive power, XPath has become very popular. To expedite evaluation of XPath queries, we consider the problem of rewriting XPath queries using materialized XPath views. This problem is very important and arises not only from query optimization in server side but also from semantic caching in client side. We consider the problem of deciding whether there exists a rewriting of a query using XPath views and the problem of finding minimal rewritings. We first consider those two problems for a very practical XPath fragment containing the descendent, child, wildcard and branch features. We show that the rewriting existence problem is coNP-hard and the problem of finding minimal rewritings is Σ_3^p. We also consider those two rewriting problems for three subclasses of this XPath fragment, each of which contains child feature and two of descendent, wildcard and branch features, and show that both rewriting problems can be polynomially solved. Finally, we give an algorithm for finding minimal rewritings, which is sound for the XPath fragment, but is also complete and runs in polynomial time for its three subclasses.

1 Introduction

Recently, more and more data are represented and exchanged as XML documents over Internet. XPath [11], recommended by W3C, is a simple but popular language to navigate XML documents and extract information from them. XPath is also used as sub-languages of other XML query languages such as XQuery [5] and XSLT [12].

Since this language is popular, there has been a lot of work done to speedup evaluation of XPath queries, for example: index techniques [10, 29], structural join algorithms [1, 6] and minimization of XPath queries [2, 30, 28, 15]. More recently, the problem of rewriting queries using materialized XML views has begun to attract more attention.

This rewriting problem has been first discussed for semantic caching because semantic caching can improve performance significantly in traditional client-server databases and Web-based information systems. Hence, [9, 32] intuitively consider using cached XML views to answer XML queries and have obtained noticeable advantage on performance. Moreover, authors in [3] also consider this problem but in XML query processing using materialized XPath views. It points out that most of new proposed indexing schemes can be modelled as materialized views such that the rewriting problem could be essential to efficient evaluation of XPath queries. In this paper, we consider this problem in formal theoretical aspects, which is not exploited in previous works to best of our knowledge.

We begin by giving some examples to describe the motivation of studying this problem. Consider the following XML document t stored in an XML server, which partially describes enzyme information of a biological pathway:

```
<Pathway name = "PA1">
  <Reaction name = "RE1">
    <Enzymes>
      <Protein name = "PR1" EC# ="1.0.0.1"/>
      <RNA name = "RN1"/>
    </Enzymes>
  </Reaction>
  <Reaction name = "RE2">
    <Enzymes>
      <RNA name = "RN2">
    </Enzymes>
  </Reaction>
</Pathway>
```


Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

Let's assume that a client issues to the server an XPath query v :

$$/Reaction/Enzymes$$

which retrieves **Enzymes** subelements of all **Reaction** elements. The server evaluates this query and sends back to the client its result as follows:

```
<Enzymes>
  <Protein name = "PR1" EC# ="1.0.0.1"/>
  <RNA name = "RN1"/>
</Enzymes>
<Enzymes>
  <RNA name = "RN2">
</Enzymes>
```

Suppose the client caches the above result. When the client issues another XPath query p_1 :

$$/Reaction/Enzymes[/Protein]$$

which retrieves all **Reaction** elements' **Enzymes** subelements that have at least a **Protein** subelement. It's obvious that the result of p_1 is a subset of the cached result and we can issue an XPath query p_1' :

$$Enzymes[/protein]$$

which retrieves **Enzymes** elements having at least a **Protein** subelement, over the result of query v to compute the result of p_1 without sending p_1 to the server. We say that p_1' together with query v is a *rewriting* of p_1, and p_1' is a *compensation query* of p using v.

Let's consider another XPath query p_2 :

$$/Reaction/Enzymes/Protein$$

which retrieves all **Protein** subelements of **Enzymes** subelements of **Reaction** elements. The result of p_2 is not a subset of the cached result of query v. But, because of nested structures of XML documents, each **Protein** element in the result of p_2 is a subelement of an **Enzymes** element in the cached result of v. We still can issue an XPath query p_2' :

$$Enzymes/Protein$$

which retrieves all **Protein** subelements of **Enzymes** elements, over the cached result to compute the result of p_2.

However, for some XPath queries, we can't compute their results by using the cached result even if we know their results are a subset or subelements of the cached result. For example, consider the following XPath query p_3 :

$$/Reaction[@name = \text{"RN2 "}]/Enzymes$$

which only retrieves **Enzymes** subelements of all **Reaction** elements with name "RN2". We know the result of p_3 is a subset of the cached result. But, we don't know which **Enzymes** element in the cached result should be included in p_3's result. Thus, there is no rewriting of p_3 using v, i.e., we can't issue a query over the result of v to answer p_3.

In general, given an XPath query(view) v which is materialized (i.e., its result is pre-computed or cached) and a new XPath query p to be answered, the first problem studied in this paper is the rewriting existence problem, i.e., whether a compensation query of p using v exists such that we can evaluate the compensation query over the pre-computed or cached result of v to answer p. In case there are multiple compensation queries, we are interested in the compensation query which needs minimum cost to evaluate. According to the theoretical analysis of [16, 17], the evaluation efficiency of XPath queries greatly depends on the size of them. Same to [30, 15, 2], we also consider the size of XPath queries as a measure for their costs. Hence, the second problem studied in this paper is to find the compensation query with minimum size(also called as finding minimal rewritings problem).

The rest of this paper is organized as follows. In Section 2, we introduce basic notations and definitions about tree patterns which are simple XPath queries but used frequently in practice. Two rewriting problems are formulated in Section 3. Section 4 and 5 discuss the complexities of those two rewriting problems for tree patterns, and give an algorithm to find minimal rewritings. Finally, we describe related work in Section 6 and give the conclusion in Section 7.

2 Preliminaries

2.1 Trees and Tree Patterns

Generally, an XML database consists of a set of XML documents. We model each XML document as an unordered rooted node-labelled tree (called XML tree) over an infinite alphabet Σ, where the label of each internal node corresponds to an XML element, an attribute name or a data value. We denote all possible XML trees over Σ as T_Σ.

Definition 2.1 *An XML document is a tree $t\langle V_t, E_t, r_t\rangle$ over Σ called **XML tree**, where*

- *V_t is the node set and E_t is the edge set;*
- *$r_t \in V_t$ is the root of t;*
- *Each node n in V_t has a label from Σ(denoted as $n.label$).*

Given an XML tree $t\langle V_t, E_t, r_t\rangle$, we say that $t'\langle V_{t'}, E_{t'}, r_{t'}\rangle$ is a **subtree** of t if $V_{t'} \subseteq V_t$ and $E_{t'} = (V_{t'} \times V_{t'}) \cap E_t$. For any node n in t, we denote the subtree rooted at n and exactly containing all its descendants as $(t)^n_{sub}$. We let n be the root of $(t)^n_{sub}$, such that $(t)^n_{sub}$ can also be seen as an XML tree. For instance, Fig. 1(c) shows the subtree rooted at 'd'-labelled node of an XML tree t, which is shown in Fig. 1(a).

In this paper, we discuss a fragment of XPath queries, first studied in [23]. This fragment consists of label tests, child axes(/), descendant axes(//), branches([]) and wildcards(*). It can be recursively represented by the following grammar:

$$xp \rightarrow l \mid * \mid xp/xp \mid xp//xp \mid xp[xp]$$

where l is a node label from Σ. We denote this fragment as $XP^{\{/,//,*,[]\}}$. Three subclasses of $XP^{\{/,//,*,[]\}}$ are also specially discussed: $XP^{\{/,//,[]\}}$, $XP^{\{/,*,[]\}}$ and $XP^{\{/,//,*\}}$, which only use two of the three features: '//', '[]' and '*' in addition to '/'.

As said in [23], any XPath query from $XP^{\{/,//,*,[]\}}$ can be trivially represented as a labelled tree(called tree pattern) with the same semantics.

Definition 2.2 *A **tree pattern** p is a tree $\langle V_p, E_p, r_p, o_p\rangle$ over $\Sigma \cup \{'*'\}$, where V_p is the node set and E_p is the edge set, and:*

- *Each node n in V_p has a label from $\Sigma \cup \{'*'\}$, denoted as $n.label$;*
- *Each edge e in E_p has a label from $\{'/','//'\}$, denoted as $e.label$. The edge with label / is called child edge, otherwise called descendent edge;*
- *$r_p, o_p \in V_p$ are the root and output node of p respectively.*

For example, an XPath query $a[*/b]/c//d$ is represented as a tree pattern shown in Fig. 1(b), where the dark node is the output node. The size of a tree pattern, written as $\|p\|$, is defined as the number of its nodes. Without loss of generality, we refer to tree patterns as patterns in the rest of this paper.

Given a pattern $p\langle V_p, E_p, r_p, o_p\rangle$, we say that $p'\langle V_{p'}, E_{p'}, r_{p'}, o_{p'}\rangle$ is a **subpattern** of p if the following conditions hold: (1)$V_{p'} \subseteq V_p$; (2)$E_{p'} = (V_{p'} \times V_{p'}) \cap E_p$; (3)If the node $o_p \in V_{p'}$, then o_p is also the output node of p'. For any node n in p, we denote as $(p)^n_{sub}$ the subpattern with n as the root and exactly containing all its descendants. As an example, the pattern $p = a[// * /d]/b[*][//d]$ is shown in (1) of Fig. 6. Let n^* be the '*'-labelled node which is a child of p's root. The subpattern $(p)^{n^*}_{sub}$ is given in (2).

We now define an embedding(also called pattern match) from a pattern to an XML tree as follows:

Definition 2.3 *Given an XML tree $t\langle V_t, E_t, r_t\rangle$ and a pattern $p\langle V_p, E_p, r_p, o_p\rangle$, an **embedding** from p to t is a function $e : V_p \to V_t$, with following properties:*

- *Root preserving: $e(r_p) = r_t$;*
- *Label preserving: $\forall n \in V_p$, if $n.label \neq$ '*', $n.label = e(n).label$;*
- *Structure preserving: $\forall e = (n_1, n_2) \in E_p$, if $e.label =$ '/', $e(n_2)$ is a child of $e(n_1)$ in t; otherwise, $e(n_2)$ is a descendent of $e(n_1)$ in t.*

The embedding maps the output node o_p of p to a node n in t. We say that the subtree $(t)^n_{sub}$ of t is the result of this embedding. As an example, dashed lines between Fig. 1(a) and (b) shows an embedding, and its result is shown in Fig. 1(c). Actually, there could be more than one embedding from p to t. We define the result of p over t, denoted as $p(t)$, as the union of results of all embeddings, i.e.,

$$\cup_{e \in EB}\{(t)^{e(o_p)}_{sub}\}$$

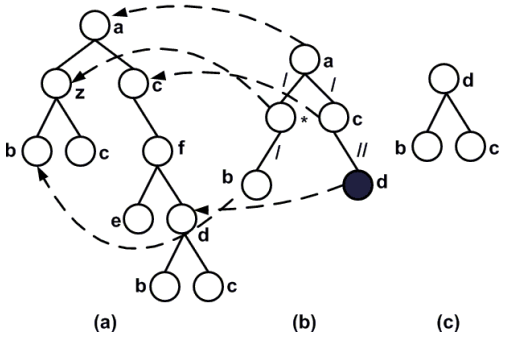

Figure 1: (a)An XML tree t; (b)A pattern p; and (c)A subtree of t

where EB is the set including all embeddings from p to t, and $e(o_p)$ is a node of t, mapped by the output node o_p of p through an embedding e.

In addition, we define an empty pattern denoted as ε: the result of evaluating ε over any XML tree is empty.

2.2 Containment and Minimization of Patterns

For any two patterns p_1 and p_2, p_1 is said to be *contained* in p_2($p_1 \sqsubseteq p_2$) iff $\forall t \in T_\Sigma\ p_1(t) \subseteq p_2(t)$, and p_1 is said to be *equivalent* to p_2 (denoted as $p_1 \equiv p_2$) iff $\forall t \in T_\Sigma\ p_1(t) = p_2(t)$. Obviously, the equivalence problem can be seen as a two-way containment problem because $p_1 \equiv p_2$ iff $p_1 \sqsubseteq p_2$ and $p_2 \sqsubseteq p_1$.

The complexity of the pattern containment problem has been well studied for $XP^{\{/,//,[],*\}}$ and also for its three subclasses. The problem is in coNP-complete [23] for $XP^{\{/,//,[],*\}}$ and in P for its three subclasses [2, 30, 25].

Minimizing a pattern p is to find an equivalent pattern $p'(\equiv p)$ with minimum size, i.e., no other equivalent pattern $p''(\equiv p)$ having $\|p''\| < \|p'\|$ exists. As shown in [15], the minimization problem is coNP-hard. However, a pattern can be minimized in polynomial time in the case of $XP^{\{/,//,[]\}}$ [2] and $XP^{\{/,*,[]\}}$ [30]. Any pattern from $XP^{\{/,//,*\}}$ is already minimized.

3 Problem Formulation

Let t be an XML tree. We use v to denote a materialized pattern whose result $v(t)$ is pre-computed or cached, and we use p to denote a pattern to be answered. Our goal is to find a pattern p' such that we can answer p by evaluating p' over the result of v, i.e., $p'(v(t))$ is equal to $p(t)$. Note that $v(t)$ may include a set of subtrees of t, and $p'(v(t))$ is defined as the union of results of evaluating p' over all subtrees in $v(t)$.

By observation, for any XML tree t, the result of evaluating a pattern p' over $v(t)$ can be viewed as the result of directly evaluating a pattern over t. Actually, this pattern can be obtained from p' and v.

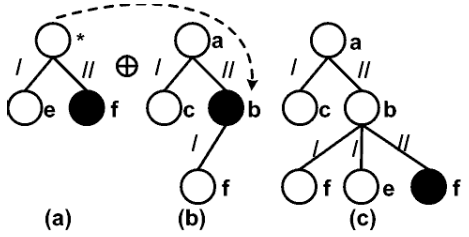

Figure 2: (a)p'; (b) v and (c)$p' \oplus v$.

We define an asymmetric concatenation operator, denoted as $\oplus$, between two patterns as below: given two patterns $p'\langle V_{p'}, E_{p'}, r_{p'}, o_{p'}\rangle$ and $v\langle V_v, E_v, r_v, o_v\rangle$, the concatenation from p' to v is a pattern, denoted as $p' \oplus v$, which is constructed from p' and v by merging $r_{p'}$(the root of p') and o_v(the output node of v) into one node. r_v and $o_{p'}$ are the root and output node of $p' \oplus v$ respectively. The merged node is denoted as $n_{p'\oplus v}$, and it has both the children of r_v in v and the children of $o_{p'}$ in p' as its children. When the two nodes $r_{p'}$ and o_v have different labels, we choose the more "restrictive" one as the label of $n_{p'\oplus v}$. That is, the label of merged node $n_{p'\oplus v}$ is chosen as

- $r_{p'}.label$ if $r_{p'}.label = o_v.label$;
- $r_{p'}.label$ if $o_v.label = *$ and $r_{p'}.label$ is from Σ;
- $o_v.label$ if $r_{p'}.label = *$ and $o_v.label$ is from Σ.

If both of the two labels are from Σ and different, then we let $p' \oplus v = \varepsilon$, i.e., the concatenation is an empty pattern.

Example 3.1 Fig. 2 shows an example: we can get pattern $p' \oplus v = a[c]//b[f][e]//f$(shown in (c)) by concatenating $p' = *[e]//f$(shown in (a)) to $v = a[c]//b[f]$ (shown in (b)). The root of p' has label *, and the output node of v has label 'b'. The merged node of the above two nodes has label 'b'. p' is a subpattern of $p' \oplus v$ under 'b'-labelled node.

We have the following result for the $\oplus$ operator.

Lemma 3.2 *Let p' and v be two patterns. $(p' \oplus v)(t)$ is equal to $p'(v(t))$ for any XML tree $t \in T_\Sigma$.*

Obviously, the fragment $XP^{\{/,//,*,[]\}}$ is closed under concatenation. Notice that the construction of $p' \oplus v$ based on p' and v doesn't introduce new wildcards or descendant edges. Hence, two subclasses $XP^{\{/,//,[]\}}$ and $XP^{\{/,*,[]\}}$ are also closed under concatenation. Moreover, if p' and v are from $XP^{\{/,//,*\}}$, p' and v are linear patterns with their leaves as output nodes. It's obvious that $p' \oplus v$ is in $XP^{\{/,//,*\}}$.

Lemma 3.3 *The fragment $XP^{\{/,//,*,[]\}}$ and its three subclasses are closed under concatenation.*

In addition, it's straightforward to show that the concatenation operator $\oplus$, considered as a binary operator, satisfies that given three patterns v, p' and p'',

1. $p'' \oplus v \sqsubseteq (or\ \equiv) p' \oplus v$ if $p'' \sqsubseteq (or\ \equiv) p'$, and
2. $(p'' \oplus p') \oplus v \equiv p'' \oplus (p' \oplus v)$ (i.e., $\oplus$ is associative)

Based on the concatenation operator and Lemma 3.2, we formally define (minimal) ***compensation patterns*** and (minimal) ***rewritings*** as follows:

Definition 3.4 *Let v be a materialized view and p be a pattern. We say that a pattern p' is a **compensation pattern** and $p' \oplus v$ is a **rewriting** of p using v if $p' \oplus v$ is equivalent to p. We also say that p' is a minimal compensation pattern, and $p' \oplus v$ is a minimal rewriting of p using v if there is no other compensation pattern p'' of p using v such that the size of p'' is less than that of p'.*

The two problems studied in this paper can now be restated as follows:
Rewriting Existence Problem: Given a pattern v and a pattern p, we check whether there exists a compensation pattern p' such that $p' \oplus v \equiv p$ or not; and
Finding Minimal Rewritings Problem: If a rewriting of p using v exists, find the minimal compensation pattern p' such that $p' \oplus v \equiv p$.

4 Rewriting Existence Problem

We discuss the complexity of rewriting existence problem in the case of $XP^{\{/,//,*,[]\}}$ in this section. Our first observation is that the rewriting existence problem is closely related to the pattern containment problem. More specifically, our next result shows that for patterns with their roots as output nodes, these two problems are equivalent.

Lemma 4.1 *Let p and v be two patterns with output nodes as roots, $p \sqsubseteq v$ iff there exists a rewriting of p using v.*

Hence, in the rest of this section, we first describe current techniques on containment of patterns and then discuss the complexity of rewriting existence problem.

4.1 Techniques on Containment of Patterns

Many techniques have been used to obtain the complexity results of the pattern containment problem, like homomorphisms [8], canonical models [23] and so on.

The homomorphism technique is first used in the containment problem of conjunctive queries [8]. The existence of a homomorphism between two patterns implies the containment relationship between them. That is, for two patterns p_1 and p_2, $p_2 \sqsubseteq p_1$ if a homomorphism from p_1 to p_2 exists.

Definition 4.2 *Given two patterns $p_1\langle V_{p_1}, E_{p_1}, r_{p_1}, o_{p_1}\rangle$ and $p_2\langle V_{p_2}, E_{p_2}, r_{p_2}, o_{p_2}\rangle$, a **homomorphism** is a function h: $V_{p_1} \to V_{p_2}$, with following properties:*

- *Root and output node preserving:* $h(r_{p_1}) = r_{p_2}$, *and* $h(o_{p_1}) = o_{p_2}$;
- *Label preserving:* $\forall n \in V_{p_1}$ $n.label =$ '*', *or* $n.label = h(n).label$;
- *Structure preserving:* $\forall e = (n_1, n_2) \in E_{p_1}$, *if* e *is a child edge,* $(h(n_1), h(n_2))$ *is also a child edge in* E_{p_2}; *otherwise,* $(h(n_1), h(n_2))$ *is a path in* p_2 *including at least a child or descendent edge.*

As an example, dashed lines in Fig. 3 represent a homomorphism from $p_1 = a[//b]/*//d$ to $p_2 = a[*/b]/c//d$.

The containment of two patterns can also imply the homomorphism existence between them in the case of $XP^{\{/,//,[]\}}$ and $XP^{\{/,*,[]\}}$, but unfortunately not in the case of $XP^{\{/,//,*,[]\}}$.

For the case of $XP^{\{/,//,*\}}$, [25, 23] propose a method to rewrite patterns in $XP^{\{/,//,*\}}$ to a new representation such that this implication still holds. For the convenience to discuss our rewriting problems, we describe this method as below but with small change, and also call it as **pattern standardization**.

The standardization works as follows. For any path consisting a chain of nodes in a pattern p of $XP^{\{/,//,*\}}$: $(v_1, v_2, ..., v_n)$, we replace the label of edge (v_i, v_{i+1}) with '//' for $i = 1, ..., n-1$ if the following conditions are satisfied: (1)v_1 is the root or its label is from Σ; v_n is the leaf(output node) or its label is from Σ; (2) The label of v_i for $i = 2, 3, ..., n-1$ is '*' (3)$\exists i$, the label of edge (v_i, v_{i+1}) is '//'. For example, a pattern p is $a//*/*/b//*/*$. Two paths $a//*/*/b$ and $b//*/*$ of p satisfy the above conditions such that p is standardized to $a//*//*//b//*//*$. Obviously, a pattern standardization can be done in linear time. A pattern p after standardization is denoted as $std(p)$. The following properties [25, 23] hold:

Lemma 4.3 *(1)For a pattern* p *in* $XP^{\{/,//,*\}}$, $p \equiv std(p)$; *(2)For two patterns* p_1 *and* p_2 *in* $XP^{\{/,//,*\}}$, *if* $p_2 \sqsubseteq p_1$, *a homomorphism exists from* $std(p_1)$ *to* p_2.

Finding a homomorphism between two patterns p_1 and p_2 can be done in polynomial time, specifically in $O(\|p_1\| \cdot \|p_2\|)$ [24]. Hence, the pattern containment problem is in P for the three subclasses of $XP^{\{/,//,*,[]\}}$.

However, the containment problem is not in P for the whole fragment $XP^{\{/,//,*,[]\}}$. [23] proposes the canonical model method to obtain its complexity. This method first introduces **boolean patterns**, which are patterns without specifying output nodes. Given an XML tree t and a boolean pattern q, we say $q(t)$ is true if an embedding exists between them; otherwise false. For two boolean patterns q_1 and q_2, we say that $q_1 \sqsubseteq q_2$ iff $\forall t \in T_\Sigma$, $q_1(t)$ implies $q_2(t)$. Then, this method translates the containment problem of two patterns to that of two boolean patterns. Finally, it shows the boolean pattern containment problem is in coNP-complete for $XP^{\{/,//,*,[]\}}$.

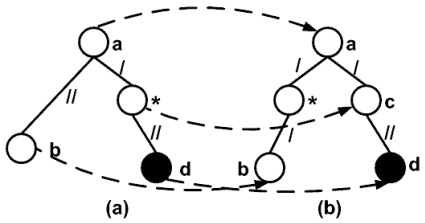

Figure 3: A homomorphism from p_1(a) to p_2(b)

4.2 Complexity

In Lemma 4.1, we have shown that the rewriting existence problem is equivalent to the pattern containment problem for patterns with their roots as output nodes. However, this special pattern containment problem is still in coNP-complete, as shown below.

Notice that for a pattern p whose output node is its root, if there is an embedding from p to an XML tree t, $p(t) = \{t\}$; otherwise $p(t) = \phi$. Obviously, patterns, with roots as output nodes, have the similar behavior as boolean patterns. Not surprisingly, we have the following complexity result by reducing the boolean pattern containment problem:

Lemma 4.4 *In case of* $XP^{\{/,//,*,[]\}}$, *the containment problem of two patterns with roots as output nodes is coNP-complete.*

Since a special case of the rewriting existence problem is coNP-complete, we have:

Theorem 4.5 *The rewriting existence problem is coNP-hard in case of* $XP^{\{/,//,*,[]\}}$.

4.3 Tractable Results

In this subsection, we show the rewriting existence problem can be solved in polynomial time for the three subclasses of $XP^{\{/,//,*,[]\}}$. Our idea is based on the fact that the existence of a homomorphism is sufficient and necessary for containment of two patterns in the case of $XP^{\{/,//,*,[]\}}$'s three subclasses.

We first consider the subclass $XP^{\{/,//,[]\}}$. The following example illustrates our intention about how to check whether a rewriting exists or not.

Example 4.6 Consider two patterns $v = a[//f]/b[c/e]$ and $p = a/b[c/e]/f$. There is a compensation pattern $p' = b[//e]/f$ of p using v such that $p' \oplus v = a[//f]/b[c/e][//e]/f$, which is equivalent to p. p', v, $p' \oplus v$ and p are shown in (a), (b), (c) and (d) of Fig. 4 respectively. The 'b'-labelled node in $p' \oplus v$ shown in (c) is the merged node of p' and v, which is denoted as $n_{p' \oplus v}$. Because $p' \oplus v \equiv p$, there is a homomorphism h_1 from p to $p' \oplus v$, represented by dashed lines between (c) and (d). h_1 maps a 'b'-labelled node in p, denoted as n_p, to $n_{p' \oplus v}$. We show next that the subpattern $(p)^{n_p}_{sub}$ rooted at n_p of

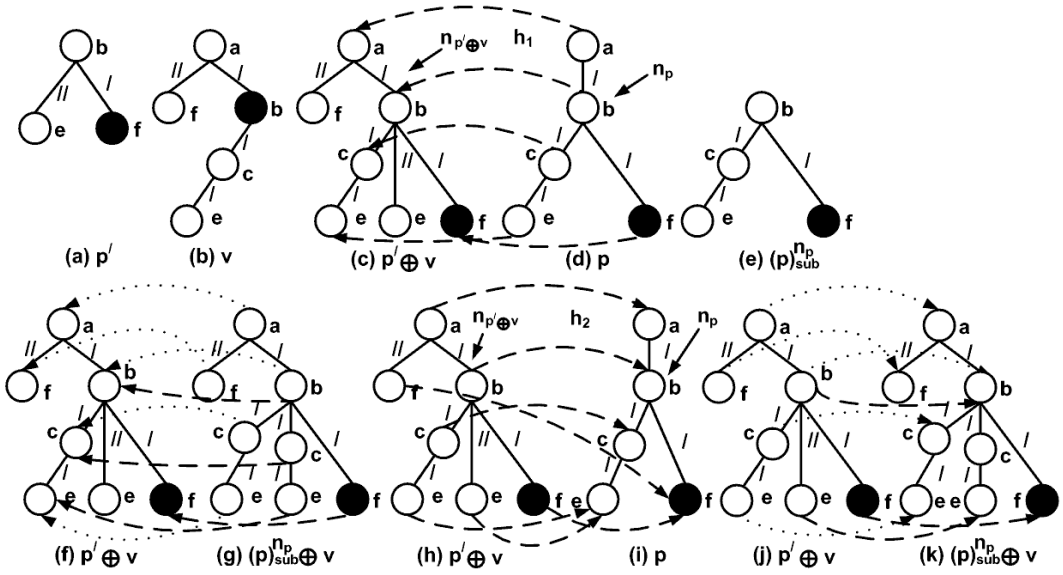

Figure 4: Constructing homomorphisms in case of $XP^{\{/,//,[]\}}$

p, shown in (e), is also a compensation pattern of p using v, i.e., $(p)^{n_p}_{sub} \oplus v \equiv p' \oplus v \equiv p$.

First, based on h_1 (refer to Fig. 4), we can construct a homomorphism h'_1 from $(p)^{n_p}_{sub} \oplus v$ (g) to $p' \oplus v$ (f) as follows: h'_1 maps each node from v part of $(p)^{n_p}_{sub} \oplus v$ to the same node from v part of $p' \oplus v$, represented by dotted lines from (g) to (f); h'_1 maps the rest nodes from $(p)^{n_p}_{sub}$ part of $(p)^{n_p}_{sub} \oplus v$, to corresponding nodes of $p' \oplus v$ as h_1 does, represented by dashed lines from (g) to (f). Hence, we have that $p' \oplus v \sqsubseteq (p)^{n_p}_{sub} \oplus v$.

Second, because $p' \oplus v \equiv p$, there is also a homomorphism h_2 from $p' \oplus v$ to p, represented by dashed lines from $p' \oplus v$ (h) to p (i) in Fig. 4. We denote the node mapped by the merged node $n_{p' \oplus v}$ under h_2 as n'_p. In our example, n'_p and n_p are the same node in p('b'-labelled node). Then, based on h_2, we can construct a homomorphism h'_2 from $p' \oplus v$ (j) to $(p)^{n_p}_{sub} \oplus v$ (k) as follows: h'_2 maps each node from v part of $p' \oplus v$ to the same node from v part of $(p)^{n_p}_{sub} \oplus v$, represented by dotted lines from (j) to (k); h'_2 maps the rest nodes from p' part of $p' \oplus v$, to corresponding nodes of $(p)^{n_p}_{sub} \oplus v$ as h_2 does, represented by dashed lines from (j) to (k). Hence, we have that $(p)^{n_p}_{sub} \oplus v \sqsubseteq p' \oplus v$.

In summary, we can construct two homomorphisms between $(p)^{n_p}_{sub} \oplus v$ and $p' \oplus v$ in both ways. Hence, $(p)^{n_p}_{sub} \oplus v \equiv p' \oplus v \equiv p$, i.e., $(p)^{n_p}_{sub}$ is a compensation pattern of p using v. □

The above example shows us that if a compensation pattern of p using v exists, there is a subpattern of p which is also a compensation pattern of p using v. Is this always true for any possible patterns p and v? The answer is yes. In the above example, we can construct a homomorphism from $(p)^{n_p}_{sub} \oplus v$ to $p' \oplus v$ if there is a node n_p of p mapped by a homomorphism(from p to $p' \oplus v$) to the merged node of $p' \oplus v$. We can also construct a homomorphism from $p' \oplus v$ to $(p)^{n_p}_{sub} \oplus v$ if the node n'_p of p mapped from the merged node of $p' \oplus v$ by a homomorphism(from $p' \oplus v$ to p) is the node n_p. Our next discussion and result can guarantee that n_p always exists and $n_{p'}$ must be n_p.

We say that the path from a pattern p's root to output node is the selection path of p. Notice that a homomorphism between two patterns always maps one pattern's root and output node to the other's root and output node respectively. We show that if two patterns are equivalent, the sizes of their selection paths are the same.(Note that this result not only holds for $XP^{\{/,//,[]\}}$ but also for $XP^{\{/,//,*,[]\}}$.)

Lemma 4.7 *Let p_1 and p_2 be two equivalent patterns. If $p_1 \equiv p_2$, the selection path of p_1 has the same size as that of p_2 in case of $XP^{\{/,//,*,[]\}}$.*

Since two equivalent patterns' selection paths have the same size, any homomorphism between them must map nodes in the selection path of one pattern to nodes in that of the other pattern sequentially one by one. Let two patterns be p and v and there is a pattern p' such that $p' \oplus v$ is a rewriting of p using v. Obviously, the merged node $n_{p' \oplus v}$ of $p' \oplus v$ is in the selection path of $p' \oplus v$. There is a unique node n_p in the selection path of p such that any homomorphism from p to $p' \oplus v$(or $p' \oplus v$ to p) maps n_p to $n_{p' \oplus v}$(or $n_{p' \oplus v}$ to n_p). Moreover, n_p has the same position in the selection path of p as the merged node $n_{p' \oplus v}$ in that of $p' \oplus v$, i.e., if $n_{p' \oplus v}$ is the i-th node in the selection path of $p' \oplus v$ starting from the root, then n_p is also the i-th node in that of p starting from the root. Since $n_{p' \oplus v}$ is merged from the output node of v and the root of p', then n_p also has the same position as the output node of v. We have the following conclusion for the subclass $XP^{\{/,//,[]\}}$.

Lemma 4.8 *Let v and p be two patterns, and let n_p be the node in the selection path of p with the same position as the output node of v in that of v. If a compensation pattern p' of p using v exists, the subpattern $(p)^{n_p}_{sub}$ of p is a compensation pattern of p using v.*

The above lemma directly implies that we only need to consider one compensation pattern candidate $(p)^{n_p}_{sub}$ to check whether a rewriting of p using v exists, because no compensation pattern of p using v exists if $(p)^{n_p}_{sub}$ is not.

Now, we discuss two other subclasses $XP^{\{/,//,*\}}$ and $XP^{\{/,*,[]\}}$. Notice that the existence of homomorphisms in both ways between $p' \oplus v$ and p is the only condition to make the above lemma work. Hence, the above lemma can easily apply to $XP^{\{/,*,[]\}}$, and the following result holds:

Corollary 4.9 *Lemma 4.8 holds for $XP^{\{/,*,[]\}}$.*

However, in the case of $XP^{\{/,//,*\}}$, $p' \oplus v$ is maybe not standardized such that there is no homomorphism from $p' \oplus v$ to p even if p' and v are standardized and $p' \oplus v \equiv p$. For example, let $p' = a/*$ and $v = *//b$, but $p' \oplus v = a/*//b$ is not standardized.

We still can construct a homomorphism from $(p)^{n_p}_{sub} \oplus v$ to $p' \oplus v$ because a homomorphism exists from p to $p' \oplus v$, and we can't construct a homomorphism from $p' \oplus v$ to $(p)^{n_p}_{sub} \oplus v$. But, in the following example, we show that a homomorphism can be constructed from p to $(p)^{n_p}_{sub} \oplus v$ such that $(p)^{n_p}_{sub} \oplus v$ is still equivalent to p, i.e., $(p)^{n_p}_{sub}$ still is a compensation pattern of p using v.

Example 4.10 Let two patterns $v = a/*$ and $p = a//*//*//b$. There is a pattern $p' = *//*/b$ such that $p' \oplus v \equiv p$, where $p' \oplus v = a//*//*//b$. p is standardized, so there is a homomorphism h from p to $p' \oplus v$, represented by dashed lines between (d) and (f). The node $n_{p' \oplus v}$ in (d) is the merged node of p' and v. h maps a '*'-labelled node in p, denoted as n_p, to $n_{p' \oplus v}$. We show next that the subpattern $(p)^{n_p}_{sub}$ rooted at n_p of p, shown in (c), is also a compensation pattern of p using v, i.e., $(p)^{n_p}_{sub} \oplus v \equiv p$. $(p)^{n_p}_{sub} \oplus v$ is shown in (e).

First, we also can construct a homomorphism h' from $(p)^{n_p} \oplus v$ to $p' \oplus v$ based on h in the same way as Example 4.6. Hence, $p' \oplus v \sqsubseteq (p)^{n_p}_{sub} \oplus v$.

Second, instead of constructing a homomorphism from $p' \oplus v$ to $(p)^{n_p}_{sub} \oplus v$ like Example 4.6, we construct a homomorphism h'' from p to $(p)^{n_p}_{sub} \oplus v$ in the following way: h'' maps each node from $(p)^{n_p}_{sub}$ part of p to the same node from $(p)^{n_p}_{sub}$ part of $(p)^{n_p}_{sub} \oplus v$, represented by dotted lines from (f) to (e); h'' maps the rest nodes of p to nodes from v part of $(p)^{n_p}_{sub} \oplus v$ as h does, represented by dashed lines from (f) to (e). Hence, $(p)^{n_p}_{sub} \oplus v \sqsubseteq p$.

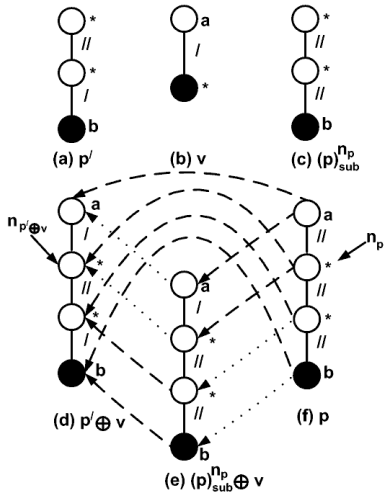

Figure 5: Constructing homomorphisms in case of $XP^{\{/,//,*\}}$

Finally, we have that $(p)^{n_p}_{sub} \oplus v \equiv p' \oplus v$ because $p \equiv p' \oplus v$, i.e., $(p)^{n_p}_{sub}$ is a compensation pattern of p using v.□

As illustrated by the the above example, we have:

Corollary 4.11 *Lemma 4.8 holds for $XP^{\{/,//,*\}}$*

Finally, we have the following complexity result for the rewriting existence problem in the case of the three subclasses of $XP^{\{/,//,*,[]\}}$ to conclude this subsection.

Theorem 4.12 *For three subclasses of $XP^{\{/,//,*,[]\}}$, the rewriting existence problem is in P.*

Proof. Directly from Lemma 4.8 and its corollaries, and the fact that testing equivalence of two patterns is in P for the three subclasses.

5 Finding Minimal Rewritings Problem

In this section, we consider the problem of finding minimal rewritings, i.e., finding the minimal compensation pattern of p using v, where p and v are two patterns. Obviously, this problem is related to minimization of patterns. In this section, we first introduce the minimization algorithm, and then discuss this problem's complexity and finally design an algorithm for it.

5.1 Algorithm on Minimization of Patterns

We first introduce several definitions and notations on patterns.

Given a pattern $p\langle V_p, E_p, r_p, o_p\rangle$ and any node n in p, we denote as $(p)^n$ the subpattern of p constructed from $(p)^n_{sub}$ by adding the root of p and connecting it to n using the same path between them in p. As an example, the pattern $p = a[//*/d]/b[*][//d]$ is shown in (1) of Fig. 6. Let n^* be the '*'-labelled node which

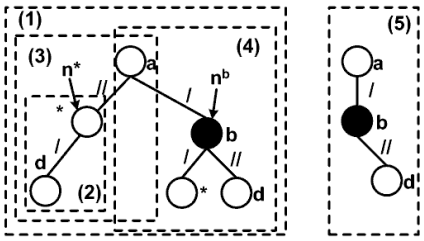

Figure 6: (1)p; (2)$(p)^{n^*}_{sub}$; (3)$(p)^{n^*}$; (4)$p - n^*$ or $(p)^{n^b}$; and (5)$min(p)$

is a child of p's root. The subpatterns $(p)^{n^*}_{sub}$ and $(p)^{n^*}$ are given in (2) and (3) respectively.

Given a pattern p, we define C_p be a node set including all children of p's root. Assume that $C_p = \{n_1, n_2, ..., n_m\}$. We denote a set of subpatterns $\{(p)^{n_1}, (p)^{n_2}, ..., (p)^{n_m}\}$ as $P(p)$. In Fig. 6, the root of p shown in (1) has two children with labels '*'and 'b', denoted as n^* and n^b, respectively. Then, $C_p = \{n^*, n^b\}$, and $P(p)$ includes two subpatterns $(p)^{n^*}$ and $(p)^{n^b}$ of p shown in (3) and (4) separately.

Given a pattern p and a node n of p, we denote as $p - n$ the subpattern obtained from p by pruning the subpattern $(p)^n_{sub}$ rooted at n. Moreover, let N be a set of nodes of p and we denote as $p-N$ the subpattern obtained from p by pruning all subpatterns rooted at nodes in N. For example, $p - n^*$ is shown in (4) of Fig. 6.

The above definitions can also be applied to boolean patterns in general. Using these definitions, the minimization problem is discussed below.

Given a boolean pattern q, [15] shows that for two nodes n_i and $n_j \in C_q$, if $(q)^{n_i} \sqsubseteq (q)^{n_j}$, n_j is redundant in q, because the subpattern $q - n_j$ is equivalent to q. Moreover, [15] also shows that q is not minimized if and only if a subpattern in $P(q)$ is not minimized or there exist two subpatterns $(q)^{n_i}$ and $(q)^{n_j}$ in $P(q)$ having $(q)^{n_i} \sqsubseteq (q)^{n_j}$. This result leads to the boolean pattern minimization algorithm in [15], which works as follows: For a node n_j in C_q, it checks whether n_j is redundant, i.e., whether there exists $n_i \in C_q$ s.t. $(q)^{n_i} \sqsubseteq (q)^{n_j}$. If yes, it prunes $(q)^{n_j}_{sub}$ and updates q to $q - n_j$. Then, it continues the above pruning procedure until C_q has no redundant nodes. Finally, for every node n_i in C_q which isn't pruned, it recursively minimizes $(n)^{n_i}_{sub}$.

The above results and minimization algorithm for boolean patterns can be trivially extended to patterns in general. As an example, the pattern shown in (1) of Fig. 6 after minimization is given in (5).

We have the following two results for patterns based on works in [15], which we will use for the finding minimal rewritings problem later.

Lemma 5.1 *Let $p\langle V_p, E_p, r_p, o_p\rangle$ be a pattern and n be a node in the path from r_p to o_p, $(min(p))^n_{sub}$ is isomorphic to $min((p)^n_{sub})$.*

Proof. Directly from the minimization algorithm and the fact that $(p)^n_{sub}$, who includes the output node of p, can not be pruned away during minimization.

Lemma 5.2 *Let p_1 and p_2 be two equivalent patterns, and p_2 is minimized. Then, for each subpattern $(p_2)^{n_j} \in P(p_2)$, there exists a subpattern $(p_1)^{n_i} \in P(p_1)$ such that $(p_1)^{n_i} \equiv (p_2)^{n_j}$.*

Proof omitted. Our proof is based on the Lemma 1 of [15]. We restate it by using our notations: let q_1 and q_2 be two boolean patterns such that $q_1 \sqsubseteq q_2$. Then, for each subpattern $(q_2)^{n_j} \in P(q_2)$, there exists a subpattern $(q_1)^{n_i} \in P(q_1)$ s.t. $(q_1)^{n_i} \sqsubseteq (q_2)^{n_j}$. □

5.2 Complexity

Not surprisingly, our first result is that the compensation pattern doesn't introduce new labels. That is, if a pattern p' is a compensation pattern of a pattern p using a pattern v, then p' doesn't have any label from Σ which doesn't appear in p.

Lemma 5.3 *Let p and v be two patterns. p' doesn't introduce new labels from Σ if p' is a compensation pattern of p using v.*

Our second result is that the minimal compensation pattern doesn't increase size, i.e., if p' is a minimal compensation pattern of p using v, then $\|p'\| \leq \|p\|$.

Lemma 5.4 *Let p and v be two patterns. If a compensation pattern of p using v exists, the minimal compensation pattern of p using v has at most size $\|p\|$.*

The proof is based on the following two important claims which follow from the definitions.

Claim 5.5 *Let v be a pattern and o_v be its output node. $(v)^{o_v}_{sub}$ is the subpattern rooted at o_v of v. The concatenation from $(v)^{o_v}_{sub}$ to v, $(v)^{o_v}_{sub} \oplus v$, is equivalent to v.*

Claim 5.6 *Let p' and v be two patterns. Let o_v be the output node of v and $n_{p'\oplus v}$ be the merged node of $p' \oplus v$. The concatenation from p' to $(v)^{o_v}_{sub}$, i.e., $p' \oplus (v)^{o_v}_{sub}$, is isomorphic to, the subpattern rooted at $n_{p'\oplus v}$ of $p' \oplus v$, i.e., $(p' \oplus v)^{n_{p'\oplus v}}_{sub}$.*

Proof. (Lemma 5.4): Assume that p' is a compensation pattern of p using v. Our idea is that we can construct a pattern based on p' such that it is a compensation pattern with size less than $\|p\|$.

Let o_v be the output node of v and $(v)^{o_v}_{sub}$ is the subpattern rooted at o_v of v. We show that $p' \oplus (v)^{o_v}_{sub}$ is also a compensation pattern of p using v. For $\oplus$ is associative, $(p' \oplus (v)^{o_v}_{sub}) \oplus v \equiv p' \oplus ((v)^{o_v}_{sub} \oplus v)$. $(v)^{o_v}_{sub} \oplus v \equiv v$ according to Claim 5.5. Thus, $(p' \oplus (v)^{o_v}_{sub}) \oplus v \equiv p' \oplus v \equiv p$. It means that $p' \oplus (v)^{o_v}_{sub}$ is a compensation pattern.

Since $p' \oplus (v)^{o_v}_{sub}$ is a compensation pattern, then $min(p' \oplus (v)^{o_v}_{sub})$ is a compensation pattern too. We show that the size of $min(p' \oplus (v)^{o_v}_{sub})$ is less than that of p. First, $p' \oplus (v)^{o_v}_{sub}$ is isomorphic to the subpattern $(p' \oplus v)^{n_{p'\oplus v}}_{sub}$ rooted at the merged node $n_{p'\oplus v}$ of $p' \oplus v$ according to Claim 5.6. Thus, the size of $min(p' \oplus (v)^{o_v}_{sub})$ is equal to the size of $min((p' \oplus v)^{n_{p'\oplus v}}_{sub})$. Second, based on Lemma 5.1, $min((p' \oplus v)^{n_{p'\oplus v}}_{sub})$ is isomorphic to $(min(p' \oplus v))^{n_{p'\oplus v}}_{sub}$. Obviously, the size of $(min(p' \oplus v))^{n_{p'\oplus v}}_{sub}$ is less than that of $min(p' \oplus v)$ since $(min(p' \oplus v))^{n_{p'\oplus v}}_{sub}$ is a subpattern of $min(p' \oplus v)$. Notice that $\|min(p' \oplus v)\| = \|min(p)\|$ due to $p' \oplus v \equiv p$. Finally, we have that $\|min(p' \oplus (v)^{o_v}_{sub})\| = \|min((p' \oplus v)^{n_{p'\oplus v}}_{sub})\| = \|(min(p' \oplus v))^{n_{p'\oplus v}}_{sub}\| \leq \|min(p' \oplus v)\| = \|min(p)\| \leq \|p\|$. Hence, $min(p' \oplus (v)^{o_v}_{sub})$ is a compensation pattern with size not greater than $\|p\|$. This implies the minimal compensation pattern of p using v has size not greater than $\|p\|$. □

Based on the above lemmas, we can obtain the following complexity result:

Theorem 5.7 *Let p and v be two patterns. The problem of whether there exists a compensation pattern p' of p using v such that p' has size less than k is Σ^p_3, where $k \leq \|p\|$.*

Proof. We can guess in polynomial time a pattern p', which doesn't introduce new labels and has size less than k. And, testing $p' \oplus v \equiv p$ is in coNP [23] (also in Σ^p_2). □

5.3 Tractable Results

In this subsection, we show that the problem of finding minimal rewritings is in P for the three subclasses. In subsection 4.3, we have already shown that for the three subclasses of $XP^{\{/,//,*,[]\}}$, if a compensation pattern of Pattern p using Pattern v exists, then there is a node n_p of p such that the subpattern $(p)^{n_p}_{sub}$ of p is also a compensation pattern.

For $XP^{\{/,//,*\}}$, we can easily have that $(p)^{n_p}_{sub}$ is the minimal compensation pattern based on the fact: any pattern in $XP^{\{/,//,*\}}$ is minimized. Any pattern p' s.t. $p' \oplus v \equiv p$ must have the same size as $(p)^{n_p}_{sub}$, because the size of $p' \oplus v$ is equal to that of $(p)^{n_p}_{sub} \oplus v$ according to they are equivalent and minimized.

However, for two other subclasses, $(p)^{n_p}_{sub} \oplus v$ may not be minimized even if both $(p)^{n_p}_{sub}$ and v are minimized. Hence, there maybe exists another pattern, which is also a compensation pattern but with less size than $(p)^{n_p}_{sub}$. An interesting question arises: can the minimal compensation pattern be found among subpatterns of $(p)^{n_p}_{sub}$?

We first discuss a special case of the problem of finding minimal rewritings, which restricts the pattern v's output node to its root, for the whole fragment $XP^{\{/,//,*,[]\}}$. In this special case, our next result shows that p is a compensation pattern if one exists.

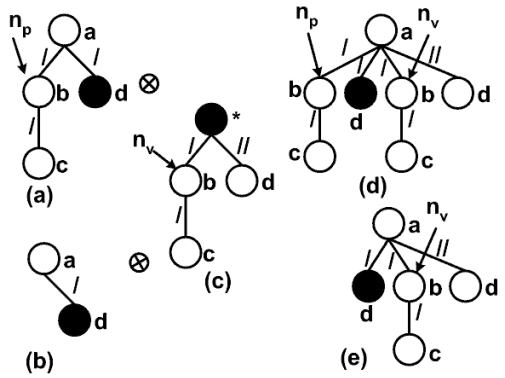

Figure 7: Example 5.9: (a)p; (b)$p - n_p$; (c)v; (d)$p \oplus v$; and (e)$(p - n_p) \oplus v$ or $(p \oplus v) - n_p$.

Lemma 5.8 *Let p be a pattern and v be a pattern whose output node is its root. If a compensation pattern of p using v exists, then p is also a compensation pattern.*

The following example shows that although p may not be a minimal compensation pattern, a minimal compensation pattern can be obtained from p.

Example 5.9 Let $p = a[b/c]/d$ and $v = *[//d][b/c]$, which are shown in Fig. 7 (a) and (c) respectively. p is a compensation pattern and $p \oplus v = a[//d][b/c][b/c]/d$ is shown in (d). We denote two 'b'-labelled nodes in p and v as n_p and n_v respectively. We see that the subpatterns $(p \oplus v)^{n_p}$ and $(p \oplus v)^{n_v}$ of $p \oplus v$ are equivalent. Hence, n_p is redundant for $p \oplus v$. $(p \oplus v) - n_p$ shown in (e), obtained from $p \oplus v$ by pruning the $(p \oplus v)^{n_p}_{sub}$, is equivalent to $p \oplus v$. It implies that $p - n_p$ obtained from p by pruning $(p)^{n_p}_{sub}$ is also a compensation pattern, because the concatenation from $p - n_p$ to v(i.e., $(p - n_p) \oplus v$) can be viewed as a pattern obtained from $p \oplus v$ by pruning n_p(i.e., $(p \oplus v) - n_p$), which is equivalent to $p \oplus v$ and p. We say that n_p is a rewriting-redundant node of p against v. In fact, n_p is the only rewriting-redundant node and $p - n_p$ is a minimal compensation pattern. $p - n_p$ and $(p - n_p) \oplus v$ are shown in (b) and (e) respectively. □

This example gives us motivation to obtain the minimal compensation pattern by pruning all rewriting-redundant nodes of p against v. This motivation leads to our following definition and the most important lemma of this work.

Definition 5.10 *Let p be a pattern and v be a pattern whose output node is its root. C_p and C_v are two node sets including all children of the roots of p and v respectively. $n_p \in C_p$ is called to be a rewriting-redundant node of p against v if there exists a node $n_v \in C_v$ s.t. $(p \oplus v)^{n_p} \equiv (p \oplus v)^{n_v}$. The set R^p_v including*

all rewriting-redundant nodes is called the rewriting-redundant node set of p against v.

Lemma 5.11 *Let p be a **minimized** pattern and v be a pattern whose output node is its root. Let R_v^p be the rewriting-redundant node set of p against v. If there exists a compensation pattern of p using v, then $p - R_v^p$ is a minimal compensation pattern of p using v.*

The proof is based on the following observation.

Claim 5.12 *Let p be a pattern and v be a pattern whose output node is its root. Pattern p' is a compensation pattern of p using v. $C_{p'\oplus v}$, $C_{p'}$ and C_v are three node sets including all children of $p' \oplus v$, p' and v's roots respectively. The following results hold:*

- *the root of $p' \oplus v$ has the same label as that of p;*
- *$C_{p'\oplus v} = C_{p'} \cup C_v$;*
- *for a node $n_{p'} \in C_{p'}$, $(p')^{n_{p'}}$ in $P(p')$ is isomorphic to $(p' \oplus v)^{n_{p'}}$ in $P(p' \oplus v)$.*

PROOF. (Lemma 5.11): We show that $p - R_v^p$ is also a compensation pattern. From Lemma 5.8, p is a compensation pattern of p using v exists. The concatenation of $p - R_v^p$ to v, i.e., $(p - R_v^p) \oplus v$, can be viewed as the pattern obtained from $p \oplus v$ by pruning all nodes in R_v^p, i.e., $(p \oplus v) - R_v^p$. According to our rewriting-redundant node set definition, all nodes in R_v^p are also redundant for $p \oplus v$, i.e., $(p \oplus v) - R_v^p \equiv p \oplus v$. Hence, $p - R_v^p$ is a compensation pattern. For simplicity, we denote $p - R_v^p$ as p' in the rest of this proof.

Now, we prove that p' is a minimal compensation pattern by showing that any other compensation pattern p'' must have size at least as p'. We show first that for each pattern in $P(p')$, there exists a pattern in $P(p'')$ equivalent to it.

Since $p' \oplus v$ and $p'' \oplus v$ are equivalent to p which is minimized, we have the fact that for each pattern in $P(p)$, both of $P(p' \oplus v)$ and $P(p'' \oplus v)$ have a pattern equivalent to it according to Lemma 5.2.

Notice that $C_{p'} = C_p - R_v^p$ and p' is obtained from p by pruning nodes in R_v^p. Then, for a node $n_{p'} \in C_{p'}$, $n_{p'} \in C_p$. We have that $(p')^{n_{p'}} (\in P(p'))$ and $(p)^{n_{p'}}$ are the same pattern. Because $p'' \oplus v \equiv p$, there exists a node $n_{p''\oplus v}$ in $C_{p''\oplus v}$ such that $(p''\oplus v)^{n_{p''\oplus v}}$ is equivalent to $(p)^{n_{p'}}$ (i.e., $(p')^{n_{p'}}$) based on the fact. We show next that $(p'' \oplus v)^{n_{p''\oplus v}}$ must also be in $P(p'')$.

$C_{p''\oplus v} = C_{p''} \cup C_v$. This node $n_{p''\oplus v}$ is included in either $C_{p''}$ or C_v. Actually, $n_{p''\oplus v}$ must be included in $C_{p''}$. If $n_{p''\oplus v} \in C_v$, we can have that $n_{p'}$ is a rewriting-redundant node of p against v. $n_{p'}$ must be included in R_v^p and should not be included in $C_{p'}$. This causes a contradiction. We show $n_{p'}$ is a rewriting-redundant node of p against v as follows: $(p''\oplus v)^{n_{p''\oplus v}}$ is isomorphic to $(p \oplus v)^{n_{p''\oplus v}}$, because $(p'' \oplus v)^{n_{p''\oplus v}}$ and $(p\oplus v)^{n_{p''\oplus v}}$ are isomorphic to $(v)^{n_{p''\oplus v}}$. Note that, in some cases, the root of v may have a different label from roots of $(p''\oplus v)^{n_{p''\oplus v}}$ and $(p\oplus v)^{n_{p''\oplus v}}$. But, roots of $(p''\oplus v)^{n_{p''\oplus v}}$ and $(p\oplus v)^{n_{p''\oplus v}}$ have the same label as the root of p according to Claim 5.12. Hence, the fact that $(p'' \oplus v)^{n_{p''\oplus v}}$ is isomorphic to $(p \oplus v)^{n_{p''\oplus v}}$ still holds. Then, $(p\oplus v)^{n_{p''\oplus v}}$ is equivalent to $(p)^{n_{p'}}$. Since p is also compensation pattern of p using v, $(p)^{n_{p'}}$ is equivalent to $(p\oplus v)^{n_{p'}}$ according to Claim 5.12. Thus, $(p \oplus v)^{n_{p''\oplus v}}$ is equivalent to $(p \oplus v)^{n_{p'}}$. It means that $n_{p'}$ is a rewriting-redundant node.

Since $n_{p''\oplus v}$ is included in $C_{p''}$, $(p'' \oplus v)^{n_{p''\oplus v}}$ is isomorphic to $(p'')^{n_{p''\oplus v}}$ according to Claim 5.12. Hence, we have that $(p'')^{n_{p''\oplus v}} \in P(p'')$ is equivalent to $(p)^{n_{p'}}$, i.e., $(p')^{n_{p'}} (\in P(p'))$. This proves that for each pattern in $P(p')$, there exists a pattern in $P(p'')$ equivalent to it.

Finally, we show that $\|p'\| \leq \|p''\|$. From the above discussion, each pattern in $P(p')$ is equivalent to a corresponding pattern $P(p'')$. Since p' is minimized, each pattern in $P(p')$ is also minimized and has no greater size than the corresponding pattern in $P(p'')$. In addition, no two patterns in $P(p')$ will be equivalent to one pattern in $P(p'')$. Thus, it's obvious that p' has less size than p''. It comes to our conclusion. □

So far, we only show how to obtain the minimal compensation pattern in case that v's output node is its root for the whole fragment $XP^{\{/,//,*,[]\}}$ in the above lemma. Unfortunately, this lemma can't be extended to the general case that v's root and output node are not the same one, for $XP^{\{/,//,*,[]\}}$.

However, for two subclasses $XP^{\{/,//,[]\}}$ and $XP^{\{/,*,[]\}}$, we can reduce the problem of finding minimal rewritings for any two patterns to that for one pattern and another pattern whose root is its output node, which is shown in the following result.

Lemma 5.13 *Let v be a pattern and o_v be its output node. Let p be a pattern and n_p in p be the corresponding node with same position to o_v. Assume that a compensation pattern of p using v exists. Then, Pattern p' is a minimal compensation pattern of p using v iff p' is a minimal compensation pattern of $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$, whose output node is its root.*

PROOF. We only need to show that p' is a compensation pattern of p using v iff p' is a compensation pattern of $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$. Since a compensation pattern of p using v exists, the subpattern $(p)_{sub}^{n_p}$ of p is a compensation pattern according to Lemma 4.8.

($\Rightarrow$)p' is a compensation pattern of p using v, so $p' \oplus v \equiv p$. We can have two homomorphisms between $p' \oplus v$ and p in both ways. Let $n_{p'\oplus v}$ be the merged node of $p' \oplus v$. $n_{p'\oplus v}$ and n_p have the same position in the selection paths of $p' \oplus v$ and p. Then, we can have mappings by these two homomorphisms between $n_{p'\oplus v}$ and n_p in both ways. Hence, we have homomorphisms in both ways between the subpattern $(p' \oplus v)_{sub}^{n_{p'\oplus v}}$ of $p' \oplus v$ and the subpattern $(p)_{sub}^{n_p}$, i.e., $(p' \oplus v)_{sub}^{n_{p'\oplus v}} \equiv (p)_{sub}^{n_p}$. According to Claim 5.6,

$(p' \oplus v)_{sub}^{n_{p'} \oplus v}$ is isomorphic to $p' \oplus (v)_{sub}^{o_v}$. It means that p' is a compensation pattern of $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$.

$(\Leftarrow) p' \oplus v$ is equivalent to $p' \oplus ((v)_{sub}^{o_v} \oplus v)$ according to Claim 5.5, and is equivalent to $(p' \oplus (v)_{sub}^{o_v}) \oplus v$ for $\oplus$ is associative, and is equivalent to $(p)_{sub}^{n_p} \oplus v$ because p' is a compensation pattern of $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$, and is equivalent to p for $(p)_{sub}^{n_p}$ is a compensation pattern of p using v. Hence, p' is a compensation pattern of p using v. □

By combining the above lemmas, we have the following complexity result of finding minimal rewritings problem:

Theorem 5.14 *For the three subclasses of $XP^{\{/,//,*,[]\}}$, the problem of finding minimal rewritings is in P.*

Proof. We use notations of Lemma 5.13 here. Finding the minimal rewriting of p using v is equivalent to finding the minimal rewriting $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$. Because $(v)_{sub}^{o_v}$'s output node is its root, we only need to compute the rewriting-redundant set of $(p)_{sub}^{n_p}$ against $(v)_{sub}^{o_v}$ according to Lemma 5.11. Obviously, computing rewriting-redundant node set can be polynomially done since checking the equivalence of two patterns is in P for the subclasses of $XP^{\{/,//,*,[]\}}$. □

5.4 Algorithm for Finding Minimal Rewritings

In the case of three subclasses, for two patterns p and v, Lemma 4.8 suggests a way to decide the existence of a compensation pattern of p using v by testing the only one compensation pattern candidate, which is a subpattern of p. Lemma 5.11 and 5.13 directly propose an algorithm to find the minimal compensation pattern by pruning rewriting-redundant nodes. The following algorithm just follows the ideas in those lemmas. It's sound and complete, and also runs in polynomial time for the three subclasses. This algorithm will run in exponential time for $XP^{\{/,//,*,[]\}}$ since checking equivalence of two patterns and minimizing patterns are coNP-hard. However, our last result shows that this algorithm is still sound for $XP^{\{/,//,*,[]\}}$.

Theorem 5.15 *The finding minimal rewritings algorithm is sound for $XP^{\{/,//,*,[]\}}$.*

Proof. We use notations of Algorithm 1 here. The algorithm checks whether $(p)_{sub}^{n_p}$ is a compensation pattern of p using v. If it is, the algorithm returns the minimal compensation pattern of $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$. Notice that when $(p)_{sub}^{n_p}$ is a compensation pattern of p using v, then the fact holds for $XP^{\{/,//,*,[]\}}$ that the minimal compensation pattern of $(p)_{sub}^{n_p}$ using $(v)_{sub}^{o_v}$ is also the minimal compensation pattern of p using v. This follows from the proof of the necessary condition of Lemma 5.13. Hence, this algorithm is sound for $XP^{\{/,//,*,[]\}}$. □

Algorithm 1: Finding minimal rewritings

Input: p and v(are two patterns)

Output: a minimal compensation pattern of p using v if exists; otherwise *null*.

```
1: Minimizing p and v;
2: Let o_v be the output node of v;
3: Let n^p be a node in p has the same position to o_v;
4: if (p)_sub^{n_p} ⊕ v ≢ p then
5:    return null;
6: end if
7: R_v^p ← Φ;
8: p' ← (p)_sub^{n_p};
9: v' ← (v)_sub^{o_v};
10: for Each n_{p'} ∈ C_{p'} do
11:    for Each n_{v'} ∈ C_{v'} do
12:       if (p' ⊕ v')^{n_{p'}} ≡ (p' ⊕ v')^{n_{v'}} then
13:          R_{v'}^{p'} = R_{v'}^{p'} ∪ {n_{p'}};
14:       end if
15:    end for
16: end for
17: return p' − R_{v'}^{p'};
```

However, this algorithm isn't complete for $XP^{\{/,//,*,[]\}}$, because the fact, that $(p)_{sub}^{n_p}$ is not a compensation pattern of p using v, doesn't imply no compensation pattern of p using v exists.

6 Related Work

The problem of rewriting queries using views has been studied in depth in the relational model [20, 21]. Recently, this problem has also been exploited in the semi-structural data model [27] with regular path queries [7, 19].

Most recently, the problem of rewriting queries using materialized XML views has attracted moderate attention. In [9], Chen et al consider in client side using cached results of previous XQuery queries to answer new queries. In [32], Yang et al consider mining frequent tree patterns to materialize and use them to answer new queries. In [3], Balmin et al consider in server side using materialized XPath views, which can include XML fragments, data values, full paths, or node references, to speedup processing of XPath queries. All above works use heuristics to decide the existence of rewritings, and [3] uses another heuristic to minimize compensation queries. No theoretic analysis on this problem has been addressed in all of them. But, a lot of theoretical works have been done for containment and minimization problems of XPath queries, which lead to our theoretical research on the problem of rewriting queries using materialized XPath views.

The most similar theoretical work to ours is [14]. Authors consider the XQuery reformulation problem for XML publishing scenario. They reduce the XML data model into a relational data model under constraints such that the XQuery reformulation problem

can be reduced to the rewriting problem of conjunctive queries under constraints. They show that an extended Chase and BackChase (C&B) algorithm is complete for the reformulation problem of a restricted class of XQueries called Behaved XQueries. The techniques we use and the conclusions we have in this paper are totally different to theirs.

In addition, other works about XPath queries are expressive powers [4, 18], satisfiability [22], the time complexity of query evaluation [16, 17], and the containment in presence of disjunction, DTDs, existential variables and SXICs (Simple XPath Integrity Constraints) [2, 26, 13, 31].

7 Conclusion

In this paper, we have discussed two problems: the rewriting existence problem and finding minimal rewritings problem for a fragment of XPath: $XP^{\{/,//,*,[]\}}$ and its three subclasses.

We have shown that the rewriting existence problem is in coNP-hard and the problem of finding minimal rewritings is in Σ_3^p for $XP^{\{/,//,*,[]\}}$, but both problems are in P for the three subclasses of $XP^{\{/,//,*,[]\}}$.

Moreover, in case of the three subclasses, we have shown that a subpattern of Pattern p is sufficient as the only one compensation pattern candidate for testing whether a rewriting of p using Pattern v exists. We have also shown that if the subpattern is a compensation pattern, then the minimal compensation pattern can be obtained from it by pruning all rewriting-redundant nodes. Based on these results, we have designed an algorithm for finding minimal rewritings, which is only sound for $XP^{\{/,//,*,[]\}}$. However, this algorithm is complete and also runs in polynomial time for the three subclasses.

Acknowledgements: This work is supported by NSF grant DBI-0128061.

References

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas and D. Srivastava. *Structural Joins: A Primitive for Efficient XML Query Pattern Matching*, In ICDE 2002: 141-152.

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

[3] A. Balmin, F. Ozcan, K. Beyer, R. Cochrane and H. Pirahesh. *A Framework for Using Materialized XPath Views in XML Query Processing*, In VLDB 2004: 60-71.

[4] M. Benedikt, W. Fan, and G. M. Kuper. *Structural Properties of XPath Fragments*, In ICDT 2003: 79-95.

[5] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie and J. Simeon. *XQuery 1.0: An XML query language*, http://www.w3.org/TR/xquery, November 2003.

[6] N. Bruno, N. Koudas, and D. Srivastava. *Holistic Twig Joins: Optimal XML Pattern Matching*, In SIGMOD 2002: 310-321.

[7] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. *Answering Regular Path Queries Using Views*, In ICDE 2000: 389-398.

[8] A. K. Chandra and P. M. Merlin. *Optimal Implementations of Conjunctive Queries in Relational Data Bases*, In STOC 1977.

[9] Li Chen and Elke A. Rundensteiner. *ACE-XQ: A Cache-aware XQuery Answering System*, In WebDB 2002: 31-36.

[10] Y. Chen, Y. Zheng and Susan B. Davidson. *BLAS: An Efficient XPath Processing System*, In SIGMOD 2004: 47-58.

[11] J. Clark. *XML Path Language (XPath)*, http://www.w3.org/TR/xpath.

[12] J. Clark. *XSL Transformations(XSLT) 1.0*, http://www.w3.org/TR/xslt.

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

[14] A. Deutsch and V. Tannen. *Reformulation of XML Queries and Constraints*, In ICDT 2003: 225-241.

[15] S. Flesca, F. Furfaro and E. Masciari. *On the Minimization of XPath Queries*, In VLDB 2003: 153-164.

[16] G. Gottlob, C. Koch and R. Pichelar. *The Complexity of XPath Query Evaluation*, In PODS 2003: 179-190.

[17] G. Gottlob, C. Koch and R. Pichelar. *XPath Query Evaluation: Improving Time and Space Efficiency*, In ICDE 2003: 379-390.

[18] G. Gottlob, C. Koch and K. U. Schulz. *Conjunctive Queries over Trees*, In PODS 2004: 189-200.

[19] G. Grahne and A. Thomo. *Query Containment and Rewriting Using Views for Regular Path Queries under Constraints*, In PODS 2003: 111-122.

[20] A. Y. Halevy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. *Answering Queries Using Views*, In PODS 1995: 95-104.

[21] A. Y. Halevy. *Answering Queries Using Views: A Survey*, In VLDB Journal 2001: Volume 10, Issue 4, Pages 270 -294.

[22] Laks V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. (Jessica) Zhao. *On Testing Satisfiability of Tree Pattern Queries*, In VLDB 2004: 120-131.

[23] G. Miklau and D. Suciu. *Containment and Equivalence for an XPath Fragment*, In PODS 2002: 65-76.

[24] G. Miklau and D. Suciu. *Containment and Equivalence for an XPath Fragment*, In Journal of the ACM 2004: Vol. 51, No. 1.

[25] T. Milo and D. Suciu. *Index Structures for Path Expressions*, In ICDT 1999: 277-295.

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

[27] Y. Papakonstantinou and V. Vassalos. *Query Rewriting Using Semistructured Views*, In SIGMOD 1999: 455-466.

[28] P. Ramanan. *Efficient Algorithms for Minimizing Tree Pattern Queries*, In SIGMOD 2002: 299-309.

[29] H. Wang, S. Park, W. Fan, and P.S. Yu. *ViST: A Dynamic Index Method for Querying XML Data by Tree Structures*, In SIGMOD 2003: 110-121.

[30] P. T. Wood. *Minimizing Simple XPath Expressions*, In WebDB 2001: 13-18.

[31] P. T. Wood. *Containment for XPath Fragments under DTD Constraints*, In ICDT 2003: 300-314.

[32] L. H. Yang, M. Lee and W. Hsu *Efficient Mining of XML Query Patterns for Caching*, In VLDB 2003: 69-80.