Finding Irredundant Contained Rewritings of Tree Pattern Queries Using Views

Junhu Wang[1], Kewen Wang[1], and Jiuyong Li[2]

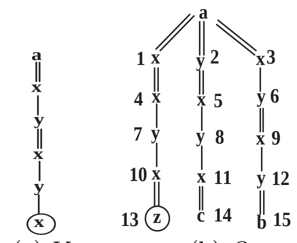
[1] Griffith University, Australia
{J.Wang, k.Wang}@griffith.edu.au
[2] University of South Australia, Adelaide, Australia
Jiuyong.Li@unisa.edu.au

Abstract. Contained rewriting and maximal contained rewriting of tree pattern queries using views have been studied recently for the class of tree patterns involving /, //, and []. Given query Q and view V, it has been shown that a contained rewriting of Q using V can be obtained by finding a *useful embedding* of Q in V. However, for the same Q and V, there may be many useful embeddings and thus many contained rewritings. Some of the useful embeddings may be redundant in that the rewritings obtained from them are contained in those obtained from other useful embeddings. Redundant useful embeddings are useless and they waste computational resource. Thus it becomes important to identify and remove them. In this paper, we show that the criteria for identifying redundant useful embeddings given in a previous work are neither sufficient nor necessary. We then present some useful observations on the containment relationship of rewritings, and based on which, a heuristic algorithm for removing redundant useful embeddings. We demonstrate the efficiency of our algorithm using examples.

1 Introduction

Query rewriting using views has many important applications such as in data integration and query caching [1]. A *view* is a pre-defined query which may or may not be materialized. A *contained rewriting (CR)*, or simply *rewriting*, of query Q using view V is a new query Q' such that, when evaluated over the answers to the view, Q' produces part or all of the correct answers to the original query Q, for any instance of the database.

The problem of finding contained rewritings has been well studied in relational databases [4, 5, 1]. Recently, contained rewritings of tree pattern queries using views were studied in [2] for the class $P^{\{/,//,[]\}}$. A tree pattern in $P^{\{/,//,[]\}}$ represents an `XPath` expression. It is a tree with every node labeled with an XML tag, and every edge labeled with either / or // (see Section 2 for more details). It was shown in [2] that, given query Q and view V in $P^{\{/,//,[]\}}$, a contained rewriting of Q using V can be found by finding a *useful embedding (UE)* of Q in V. However, there can be many UEs, and some of them may be redundant in that the rewritings obtained from them are contained in those obtained from



(a) V (b) Q

13 //	14 //	15 //
13 //	14 //	12 /
13 //	14 //	9 //
13 //	5 //	15 //
⋮	⋮	⋮
1 //	2 //	3 //

(c) UE matrix

Fig. 1. (a) and (b): Example tree patterns; (c): part of a UE matrix

other UEs. For example, for the view V and query Q shown in Fig.1 (a) and (b), there are 80 UEs, but 79 of them are redundant. Redundant UEs are useless and they waste computational resource because all results produced by them can be produced by other rewritings. Therefore it is important for the redundant UEs to be efficiently identified and removed. As observed in [2], identifying and removing all redundant UEs is a challenging problem.

In this paper, we first show that the criterion given in [2] for identifying irredundant UEs is wrong, and in fact, it is neither sufficient nor necessary. This is done in Section 3.1. We then present some useful observations on the relationships between UEs (Section 3.2). Based on these observations, we give a heuristic algorithm for computing all irredundant UEs (in Section 4). We demonstrate the efficiency of our algorithm by examples.

We will introduce some background knowledge and notations in the next section, before proceeding to present our major results in Sections 3 and 4.

2 Preliminaries

XTree and tree patterns Let Σ be an infinite set of tags. An XML *tree (*`XTree`*)* is a tree with every node labeled with a tag in Σ. A *tree pattern (*`TP`*)* in $P^{\{/,//,[]\}}$ is a tree with a unique *distinguished node*, with every node labeled with a tag in Σ, and every edge labeled with either / or //. The path from the root to the distinguished node is called the *distinguished path*. Fig.2 shows some example `TP`s, where single and double lines are used to represent /-edges and //-edges respectively, and a circle is used to indicate the distinguished node. A `TP` corresponds to an `XPath` expression. The `TP`s in Fig.2 (a) and (b) correspond to the `XPath` expressions $a[c]//b[//d]$ and $a[c]//b[x]/y$ respectively.

Let P be a `TP`. We will use `DN(P)`, and `DP(P)` to denote the distinguished node and the distinguished path of P respectively. For any tree T, we will use $N(T)$, $E(T)$ and $rt(T)$ to denote the node set, the edge set, and the root of T respectively. We will also use $label(v)$ to denote the label of node v, and call a node labeled a an *a-node*. In addition, if (u, v) is a /-edge (resp. //-edge), we say v is a /-child (resp. //-child) of u.

A *matching* of `TP` P in an `XTree` t is a mapping δ from $N(P)$ to $N(t)$ that is (1) *root-preserving*: $\delta(rt(P)) = rt(t)$, (2) *label-preserving*: $\forall v \in N(P)$, $label(v) =$

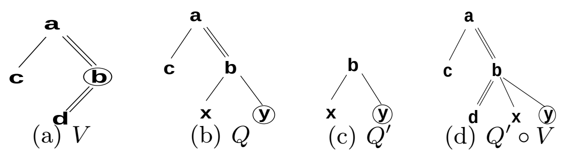


Fig. 2. View (a), Query (b), CP (c), and CR (d)

label$(\delta(v))$, and (3) *structure-preserving*: for every edge (x, y) in P, if it is a /-edge, then $\delta(y)$ is a child of $\delta(x)$; if it is a //-edge, then $\delta(y)$ is a descendant of $\delta(x)$. Each matching δ produces a subtree of t rooted at $\delta(\texttt{DN(P)})$, denoted $t^{\delta(\texttt{DN(P)})}$, which is known as an *answer* to the TP. We use $P(t)$ to denote the set of all answers of P over t. For a set T of XTrees, we define $P(T) = \cup_{t \in T} P(t)$.

Containment and containment mapping Let P and Q be TPs. P is said to be *contained* in Q, denoted $P \subseteq Q$, if for every XTree t, $P(t) \subseteq Q(t)$. For P and Q in $P^{\{/,//,[]\}}$, $P \subseteq Q$ iff there is a containment mapping from Q to P [3]. Recall: a *containment mapping (*CM*)* from Q to P is a mapping δ from $N(Q)$ to $N(P)$ that is label-preserving, root-preserving as discussed above, structure-preserving (which now means for any /-edge (x,y) in Q, $(\delta(x), \delta(y))$ is a /-edge in P, and for any //-edge (x, y), there is a path from $\delta(x)$ to $\delta(y)$ in P) and is output-preserving, which means $\delta(\texttt{DN(Q)}) = \texttt{DN(P)}$.

Rewriting TP using views A *view* is a pre-defined TP. Let V be a view and Q be a TP. In this paper we implicitly assume that $rt(Q)$ and $rt(V)$ have the same label. A *compensation pattern* (CP) of Q using V is a TP Q' such that (1) for any XTree t, $Q'(V(t)) \subseteq Q(t)$, and (2) there exists an XTree t such that $Q'(V(t))$ is non-empty. Let Q' be a CP of Q using V. It is clear that *label*$(rt(Q'))$ = *label*$(\texttt{DN(V)})$. We use $Q' \circ V$ to represent the TP obtained by merging $rt(Q')$ and DN(V). The distinguished node of Q' becomes that of $Q' \circ V$. We call $Q' \circ V$ a *contained rewriting* of Q using V. Fig.2 shows a view V, a TP Q, a CP Q' of Q using V, and the corresponding CR $Q' \circ V$. Note that condition (1) in the definition of contained rewriting is equivalent to $Q' \circ V \subseteq Q$. The *maximal contained rewriting (MCR)* of Q using V is defined to be the union of all CRs of Q using V [2].

Given TP Q and view V in $P^{\{/,//,[]\}}$, [2] shows that the existence of a CR of Q using V can be characterized by the existence of a *useful embedding (UE)* of Q in V. In brief, a useful embedding of Q in V is a partial mapping f from $N(Q)$ to $N(V)$ that is root-preserving, label-preserving, structure-preserving as defined in a containment mapping (except that they are required only for the nodes of Q on which the function f is defined), and satisfies the following conditions. (1) if $f(x)$ is defined, then f is defined on *parent*(x) (the parent of x). (2) for every node x on the distinguished path DP(Q), if $f(x)$ is defined, then $f(x) \in \texttt{DP(V)}$, and if $f(\texttt{DN(Q)})$ is defined, then $f(\texttt{DN(Q)}) = \texttt{DN(V)}$; (3) for every path $p \in Q$, *either* p is fully embedded, i.e., f is defined on every node in p, *or* if x is the last node in p such that $f(x)$ is defined and $f(x) \in \texttt{DP(P)}$ (x is called the *anchor node*), and y the node immediately following x (y is call the *successor* of x), then either $f(x) = \texttt{DN(V)}$, or the edge (x, y) is a //-edge.

Given a UE f from Q to V, a *clip-away tree (*CAT*)* CAT_f can be constructed as follows. The root of CAT_f is labeled $label(\texttt{DN(V)})$. For each path $p \in Q$ that is not fully embedded in V, find the anchor node x and its successor y, then connect the root of Q^y (Q^y is subtree of Q rooted at node y) to $rt(\mathsf{CAT}_f)$ with the same type of edge as (x, y). The distinguished node of CAT_f is the node corresponding to $\texttt{DN(Q)}$. The MCR of Q using V is the union of all CRs found this way.

3 Redundant UEs re-examined

We fix query Q and view V. For UE f, we define $\mathsf{CR}_f = \mathsf{CAT}_f \circ V$, namely CR_f is the CR obtained from f. Let f and g be UEs. We say f is contained in g if $\mathsf{CR}_f \subseteq \mathsf{CR}_g$.

Definition 1. *A UE f and the rewriting CR_f are said to be* redundant *if CR_f is contained in the union of the CRs generated by other UEs. Otherwise they are said to be* irredundant*.*

Using the independence of containing patterns property proved in [6], we know that f is redundant if and only if there is another UE g such that $\mathsf{CR}_f \subseteq \mathsf{CR}_g$. Thus the above definition of redundancy coincides with that in [2].

Intuitively redundant UEs and their corresponding CRs may be ignored because all answers produced by them can be produced by CRs obtained from other UEs.

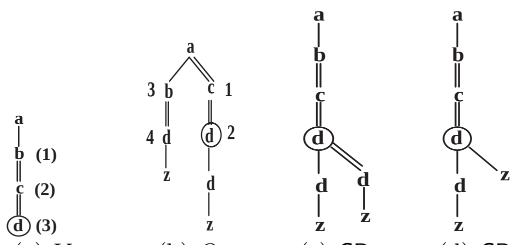
3.1 Re-examining the conditions given in [2]

For $\texttt{TP}$s in $P^{\{/,//,[]\}}$, Lemma 1 of [2] gave a condition for irredundant UEs, and it claims the condition is sufficient and necessary. For easy reference we copy the lemma below.

A node $v \in Q$ is said to be *special* wrt useful embedding f, if $f(v) = \texttt{DN(V)}$, and v has a /-child u which is undefined by f. Two nodes are said to be *incomparable* if neither is the ancestor of the other. Given useful embeddings f and g, if g is defined on every node on which f is defined, then g is called an *extension* of f. A /-path is a path consisting of /-edges only.

Lemma 1 of [2]: Let $Q, V \in P^{\{/,//,[]\}}$. Suppose f is a useful embedding and CAT_f is the CAT induced by f. Then CR_f is irredundant iff, for every node $x \in Q$ on which f is undefined, one of the following conditions holds:

1. there is no extension g of f such that $g(x)$ is defined;
2. $\exists z \in Q$: Q has a /-path from x to z or $x = z$ (note: $x = z$ is the case when the /-path is of length 0, as allowed in [2]), and z is special wrt every extension g of f for which $g(x)$ is defined;
3. x is the distinguished node of Q, and every extension g of f for which $g(x)$ is defined, maps x to $\texttt{DN(V)}$, and there is a node $y \in Q$, incomparable with x, such that $g(y)$ is undefined.



(a) V_1 (b) Q_1 (c) CR_f (d) CR_g

Fig. 3. Counter example to Lemma 1 of [2]

The following examples show that the above lemma is wrong, and in fact, the condition is neither sufficient nor necessary for f to be irredundant. Since the root of the query is always mapped to the root of the view, we omit such explicit description in the examples.

Example 1. Consider the view V_1 and query Q_1 in Fig.3. Let f be the UE which only maps the nodes 1, 2, 3 to (2), (3), and (1) respectively. Let g extend f by mapping the node 4 to the node (3). CR_f and CR_g are shown in Fig.3 (c), (d) respectively. Clearly, $\mathsf{CR}_g \subseteq \mathsf{CR}_f$ (and $\mathsf{CR}_f \nsubseteq \mathsf{CR}_g$). Therefore g is redundant. But for every node in Q_1 that is not embedded by g, there is no extension of g which embeds that node. In other words, for every node on which g is undefined, condition 1 of the lemma is true, but g is not irredundant.

The example above also demonstrates that (1) $\mathsf{CR}_g \subseteq \mathsf{CR}_f$ does not imply that f must be an extension of g, and (2) g is redundant does not imply there is an extension h of g such that $\mathsf{CR}_g \subseteq \mathsf{CR}_h$.

Example 2. Consider V_2 and Q_2 in Fig.4. Let f map the nodes 1, 2, 3 and 5 to the nodes (2), (3), (4) and (1) respectively. Let g extend f by mapping the nodes 6 and 7 to the nodes (3) and (4) respectively as well. g is the only extension of f, and g does not embed the two u-nodes, and the node (6) and (7) are special wrt g. In other words, for every node on which f is not defined, either condition 1 or condition 2 is true, but f is redundant because $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ (CR_f and CR_g are as shown in Fig.4).

Example 3. Consider V_1 and Q_3 in Fig.4. Let f' map the nodes 1, 3, 4 to the nodes (2), (1), (3) respectively. Let g' extend f' by mapping node 2 to node (3) as well. g' is the only extension of f', and every node that is not embedded by f' is either not embedded by g' (the z-nodes), or it (node 2) is the distinguished node of Q_3 and it is mapped to $\mathtt{DN(V_1)}$ by g', and there is a node (the left z-node) incomparable to node 2, which is not defined by g'. In other words, for every node on which f' is not defined, either condition 1 or condition 3 is true. But $\mathsf{CR}_{f'}$ is redundant because $\mathsf{CR}_{f'} \subseteq \mathsf{CR}_{g'}$.

The above examples show that the condition in Lemma 1 of [2] is insufficient. The next example will show it is not necessary either.

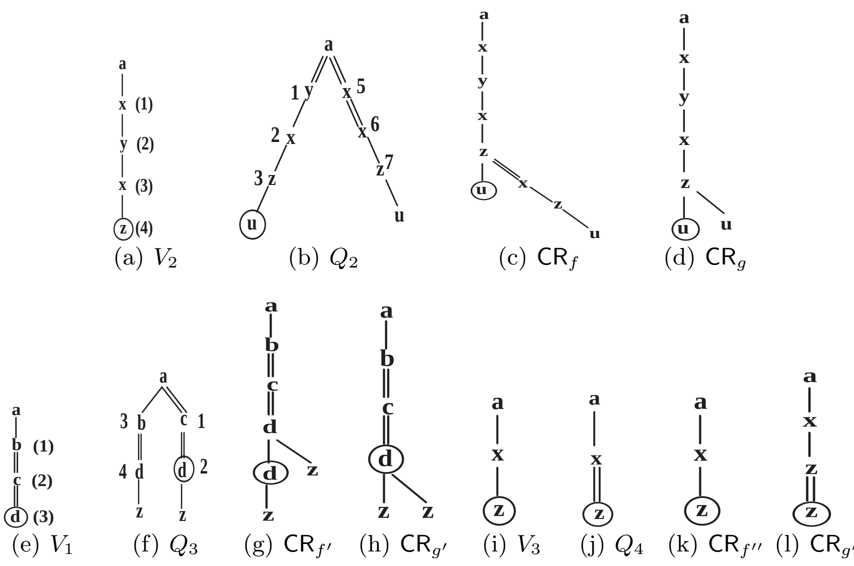


Fig. 4. Counter examples to Lemma 1 of [2]

Example 4. Consider V_3 and Q_4 in Fig.4. Let f'' map the x-node (in Q_4) to the x-node in V_3. The z-node is the only node not embedded by f'', and it is the distinguished node of Q. The only extension g'' of f'' maps the z-node in Q_4 to the z-node in V_3 (the distinguished node of V). Clearly f'' is irredundant ($\mathsf{CR}_{f''}$ and $\mathsf{CR}_{g''}$ are as show in the figure), but there is no y, incomparable with z, such that $g''(y)$ is undefined, z is not special with respect to g'', and z is embedded by g''. That is, for the z-node, none of the conditions in the lemma holds.

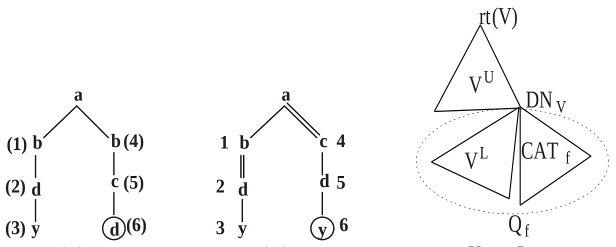
3.2 Sufficient conditions for redundant UEs and other observations

Finding a simple, exact condition for irredundant UEs turns out to be quite challenging. From practice point of view, it is more important to identify UEs that are redundant, because such UE can be ignored when computing the MCR. In this section we provide some sufficient conditions and some related observations. They will be used in our heuristic algorithm (to be presented in the next section) for finding irredundant UEs. By definition, a UE f is redundant if there is another UE g such that $\mathsf{CR}_f \subseteq \mathsf{CR}_g$. Therefore, we will first look at conditions under which $\mathsf{CR}_f \subseteq \mathsf{CR}_g$.

In the following, if UE f is defined on node $x \in N(Q)$, we say f *embeds* x. If f embeds all nodes on path $p \in Q$, we say f fully embeds p.

Lemma 1. *Let f and g be two useful embeddings. If f and g embed the same set of nodes in Q, then $\mathsf{CAT}_f = \mathsf{CAT}_g$ and $\mathsf{CR}_f = \mathsf{CR}_g$. However, $\mathsf{CAT}_f = \mathsf{CAT}_g$ does not imply f and g embed the same set of nodes.*

Proof. By definition, if f and g embed the same set of nodes in Q, then $\mathsf{CAT}_f = \mathsf{CAT}_g$ and $\mathsf{CR}_f = \mathsf{CR}_g$. The example below shows that $\mathsf{CAT}_f = \mathsf{CAT}_g$ does not



(a) V_5 (b) Q_5 (c) V^U, V^L and Q_f

Fig. 5. (a) and (b) Example view and query used in proof of Lemma 1. (c) V^U, V^L and Q_f

imply f and g embed the same set of nodes. For V_5 and Q_5 in Fig.5 (a) and (b), let f and g be as follows:

$f : 1 \rightarrow (1), 2 \rightarrow (2), 3 \rightarrow (3), 4 \rightarrow (5), 5 \rightarrow (6).$

$g : 1 \rightarrow (4), 2 \rightarrow (6), 4 \rightarrow (5), 5 \rightarrow (6).$

Then $\mathsf{CAT}_f = \mathsf{CAT}_g$ although g and f do not embed the same set of nodes.

Lemma 2. *Let Q be the query, and V be the view in $P^{\{/,//,[]\}}$. $\mathsf{CAT}_f \circ V \subseteq \mathsf{CAT}_g \circ V$ if, but not only if, $\mathsf{CAT}_f \subseteq \mathsf{CAT}_g$.*

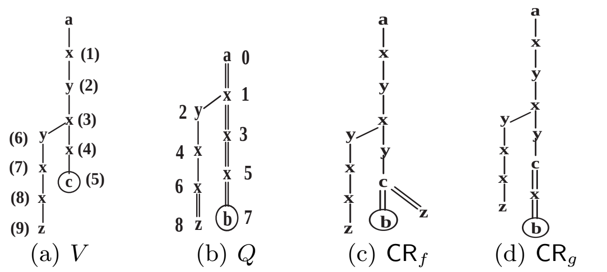
The proof of 'if' is straightforward. As an example to show $\mathsf{CAT}_f \circ V \subseteq \mathsf{CAT}_g \circ V$ does not imply $\mathsf{CAT}_f \subseteq \mathsf{CAT}_g$, consider $V = a//x$, and $Q = a//x/y$. Let f map $rt(V)$ to $rt(Q)$, let g extend f by mapping the x-node in Q to the x-node in V. Then $\mathsf{CAT}_f = x//x/y$ and $\mathsf{CAT}_g = x/y$. Thus $\mathsf{CR}_f \circ V = a//x//x/y$ and $\mathsf{CAT}_g \circ V = a//x/y$. Clearly, $\mathsf{CAT}_f \not\subseteq \mathsf{CAT}_g$, but $\mathsf{CAT}_f \circ V \subseteq \mathsf{CAT}_g \circ V$.

The theorem below provides a sufficient condition for $\mathsf{CR}_f \subseteq \mathsf{CR}_g$, which is the main idea behind our algorithm in the next section. In the theorem, $x(p, f)$ denotes the anchor node of path p with respect to UE f.

Theorem 1. *Let f and g be two useful embeddings. $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ if the following conditions are satisfied.*

1. *Every path in Q that are fully embedded by f is also fully embedded by g;*
2. *For every path p in Q that is not fully embedded by g either $x(p, f) = x(p, g)$, or $x(p, f)$ is an ancestor of $x(p, g)$, and $x(p, g)$ and its successor are connected by a $//$-edge.*
3. *Either both $g(\mathtt{DN(Q)})$ and $f(\mathtt{DN(Q)})$ are defined, or both are undefined.*

Proof. By definition, in Q only paths that are not fully embedded by g will contribute to CAT_g. For each path p that is not fully embedded by g, it is not fully embedded by f either (condition (1)). Furthermore, by condition (2), either (A) $x(p, f) = x(p, g)$, or (B) $x(p, f)$ is an ancestor of $x(p, g)$, and $x(p, g)$ and its successor are connected by a $//$-edge. In case (A), the subtree contributed by p to CAT_g is identical to that contributed by p to CAT_f; in case (B), the subtree contributed by p to CAT_g is a subtree of the subtree contributed by p to CAT_f.


(a) V (b) Q (c) CR_f (d) CR_g

7 //	8 //
5 //	8.5 //
5 //	8 //
3 //	8.5 //
3 //	8 //
1 //	8.5 //
1 //	8 //

(e) UE matrix

Fig. 6. V, Q and two CATs in Example 5

Since $x(p, g)$ and its successor are connected by a //-edge, the edge between $rt(\mathsf{CAT}_g)$ and the successor of $x(p, g)$ is also a //-edge. Therefore, there is a homomorphism from CAT_g to CAT_f. Furthermore, condition (3) ensures that either both $g(\texttt{DN(Q)})$ and $f(\texttt{DN(Q)})$ are defined, or both are undefined. In the former case, $g(\texttt{DN(Q)})$ and $f(\texttt{DN(Q)})$ are both $\texttt{DN(V)}$; in the later case, $g(\texttt{DN(Q)})$ can be mapped to $f(\texttt{DN(Q)})$ by the homomorphism. Thus in both cases, there is a containment mapping from CAT_g to CAT_f. That is, $\mathsf{CAT}_f \subseteq \mathsf{CAT}_g$. Hence $\mathsf{CR}_f \subseteq \mathsf{CR}_g$.

The above theorem provides a sufficient condition for f to be redundant: f is redundant if there is another UE g that satisfies the conditions (1) to (3) in the theorem. Note that the conditions (1) and (2) are equivalent to say g is an extension of f, and for every path on which g embeds more nodes than f, the anchor node (wrt g) is connected to its successor via a //-edge. Note also that the condition (3) in Theorem 1 is necessary. As an example, consider $V = a/x$ and $Q = a//x$. Let g map x in Q to x in V, but f do not. Then g is an extension of f, and CAT_g has no /-child. But $\mathsf{CR}_f \not\subseteq \mathsf{CR}_g$.

Using the above theorem, we can verify that, for the view V and query Q in Fig.1, the UE that embeds all nodes except 13, 14, and 15 contains all other UEs.

One might wonder whether it is possible for there to be two UEs f and g such that f embeds more nodes on path p_1 than g, but less nodes on path p_2 than g, $rt(\mathsf{CAT}_f)$ and $rt(\mathsf{CAT}_g)$ have no /-child, and both f and g are irredundant. The next example gives a positive answer to this question.

Example 5. Consider the view V and query Q in Fig.6. Let f and g be as follows:

f: $1 \to (1)$, $3 \to (3), (5) \to (4)$, $2 \to (2)$, $4 \to (3)$, $6 \to (4)$

g: $1 \to (3)$, $3 \to (4), 2 \to (6)$, $4 \to (7)$, $6 \to (8)$, $8 \to 9$

Compared with g, f embeds more nodes on the distinguished path, but less nodes on the non-distinguished path. CR_f and CR_g are as shown in Fig.6 (c) and (d). Both f and g are irredundant. Note there is no UE that embeds both node 5 and node 8.

In what follows, we will use P^y to denote the subtree of P rooted at node y, and use V^L as a shorthand for $V^{\texttt{DN(V)}}$. We will also use Q_f to denote the pattern obtained by merging the roots of V^L and CAT_f (see Fig.5 (c)). For a UE f and a path p (in Q) that is not fully embedded by f, we will use $y(p, f)$ to denote the successor of $x(p, f)$(recall: $x(p, f)$ is the anchor node of p with respect to f). As a shorthand we will use x_f to denote the last node on $\mathtt{DP(Q)}$ that is embedded by f.

Proposition 1. *Let f and g be useful embeddings.*

(1) If x_f is a descendant of x_g, then $\mathsf{CR}_f \nsubseteq \mathsf{CR}_g$.
(2) If $x_f = x_g$, then $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ iff $Q_f \subseteq \mathsf{CAT}_g$,
(3) If x_f is an ancestor of x_g, and there are no //-edges on $\mathtt{DP(V)}$, then $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ iff $Q_f \subseteq \mathsf{CAT}_g$.

Proof. We use the fact that $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ iff there is a containment mapping from CR_g to CR_f.

(1) If x_f is an descendant of x_g then $|\mathtt{DP}(\mathsf{CR}_\mathtt{g})| > |\mathtt{DP}(\mathsf{CR}_\mathtt{f})|$, and thus cannot be a containment mapping from CR_g to CR_f.
(2) If $x_f = x_g$, there is an obvious one-to-one correspondence between the nodes on $\mathtt{DP}(\mathsf{CR}_\mathtt{f})$ and those on $\mathtt{DP}(\mathsf{CR}_\mathtt{g})$, and $\mathtt{DN(V)}$ corresponds to $\mathtt{DN(V)}$ on the two paths. Therefore, $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ iff $Q_f \subseteq \mathsf{CAT}_g$.
(3) If x_f is an ancestor of x_g, and there are no //-edges on $\mathtt{DP(V)}$, then $\mathtt{DN(V)}$ on $\mathtt{DP}(\mathsf{CR}_\mathtt{g})$ corresponds to $\mathtt{DN(V)}$ on $\mathtt{DP}(\mathsf{CR}_\mathtt{f})$. Hence $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ iff $Q_f \subseteq \mathsf{CAT}_g$.

Proposition 1 implies that (1) UEs that embed more nodes on $\mathtt{DP(Q)}$ can not be contained in those that embed less nodes on $\mathtt{DP(Q)}$. (2) if f and g embed the same nodes on $\mathtt{DP(Q)}$, or they satisfy condition (3), then testing for $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ can be done by testing $Q_f \subseteq \mathsf{CAT}_g$, which is easier because Q_f and CAT_g are usually smaller than CR_f and CR_g.

The next proposition can be used to determine non-containment of UEs quickly.

Proposition 2. *If $g(\mathtt{DN(Q)})$ is defined, but $f(\mathtt{DN(Q)})$ is not, then $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ implies there is a homomorphism from V_L to Q^L (Q^L is the subtree of Q rooted at $\mathtt{DN(Q)}$), and there is a homomorphism from CAT_g to Q^L, and there is at least one //-edge on $\mathtt{DP(V)}$.*

The above proposition is true because $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ implies there is a containment mapping from CR_g to CR_f, which maps $g(\mathtt{DN(Q)})$ to $f(\mathtt{DN(Q)})$. Thus there must be a //-edge on $\mathtt{DP(V)}$. If we restrict the containment mapping to nodes in Q_g, it becomes a homomorphism from Q_g to Q^L. Thus there is a homomorphism from V_L to Q^L, and from CAT_g to Q^L.

4 A heuristic algorithm for finding irredundant UEs

We are interested in a set of UEs that has the following property: (1) the union of CRs generated by these UEs is equivalent to the MCR; and (2) no UE in the set is contained in any other. We call such a set of UEs a *minimal cover set*.

A naive approach to finding a minimal cover set is to, first, find all of the UEs using the algorithm in [2], and then, for each UE f, test whether it is contained in another g using the method of containment mapping, and if yes, discard it. This method is very inefficient in general. For example, for the query Q and view V in Fig.1, there are 80 UEs, but only one of them is irredundant. If we compare every pair of UEs randomly we would have to make at least 79 containment tests (e.g., in the lucky case we have chosen to compare the irredundant one with every other)[3]. Recall that finding the existence of a containment mapping from tree pattern P_1 to P_2 takes $O(|P_1||P_2|)$ where $|P_1|$ represents the number of edges in P_1 [3].

We now present a heuristic algorithm for finding a minimal cover set of UEs. Our algorithm uses Theorem 1 to filter out some redundant UEs first (see Algorithm 1). Then it compares the UEs that embed the same nodes on $\texttt{DP(Q)}$ to see whether further redundancy can be removed. This can be done by comparing their CATs (Proposition 1 (2)). Finally it deals with the case where f embeds less nodes than g on $\texttt{DP(Q)}$. In such case we know $\mathsf{CR}_g \not\subseteq \mathsf{CR}_f$ (Proposition 1 (1)). So we only need to test whether $\mathsf{CR}_f \subseteq \mathsf{CR}_g$. We do this using Proposition 1 (3) and Proposition 2 first, before resorting to the method of containment mapping. The algorithm is summarized into the following steps.

1 Filter redundant UEs using Algorithm 1.
2 If there are two or more UEs left after the above step, divide them into sets $S_1, \ldots, S_N$, such that each UE in S_i embeds k_i nodes on $\texttt{DP(Q)}$, and $k_1 > k_2 > \ldots > k_N$. For each pair of UEs f and g within the same group, test whether $Q_f \subseteq \mathsf{CAT}_g$. If yes, remove f.
3 For $i = 1$ to $N-1$, and $j = i+1$ to N, if there are $g \in S_i$ and $f \in S_j$, then test whether $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ (if yes, remove f) as follows. (1) If the are no //-edges on $\texttt{DP(V)}$, simply test $Q_f \not\subseteq \mathsf{CAT}_g$; (2) if $g(\texttt{DN(Q)})$ is defined, and (there is no //-edge on $\texttt{DP(Q)}$ *or* there is no homomorphism from CAT_g to Q^L, then $\mathsf{CR}_f \not\subseteq \mathsf{CR}_g$. In other cases test whether $\mathsf{CR}_f \subseteq \mathsf{CR}_g$ using containment mapping.

Steps 2 and 3 are easy to understand, they are applications of Propositions 1 and 2. We now describe Step 1 in more detail. Given query Q, we give each node v a unique identifying integer N_v such that for any two nodes u and v in any path $p \in Q$, u is a descendant of v iff $N_u > N_v$. This number is assigned using *breadth-first traversal*, starting from $rt(Q)$ and the number 0, with increment 1. Fig.6 (b) shows such a Q. Note this process takes time linear in the size of Q.

Suppose $p_1 = \texttt{DP(Q)}$ and $p_2, \ldots, p_k$ are all other paths in Q. These paths can be identified by finding the leaf nodes. Paths corresponding to leaf nodes which are descendants of $\texttt{DN(Q)}$ can be ignored. Furthermore, paths that have a common ancestor which can never be embedded into V by any UE (such nodes are easily found after generating all UEs using the algorithm in [2]) can be treated as a single path.

Suppose we have n UEs $f_1, \ldots, f_n$. We represent each UE f_i by a vector $\langle a_{i,1}, a_{i,2}, \ldots, a_{i,k} \rangle$, where $a_{i,j} = N_{y(p_j, f_i)}$ if p_j is not fully embedded by f_i, and

[3] We may have to make many more containment tests.

$y_i(f) = M + 0.5$ otherwise, where M is the maximal value among the identifiers of all possible successor nodes across all UEs [4]. For example, for the query Q and view V in Fig.6, $M = 8.5$. All UEs together form a $n \times k$ matrix, denoted $A = (a_{i,j})$. We mark each value in the matrix with either / or //, representing the fact the corresponding node is connected to its parent with a /-edge or a //-edge. The value $M + 0.5$ is marked with //.

In Algorithm 1, we first sort the rows in descending order of the sum of values in the row, and then in descending order of the number of values marked //. The aim of this is to arrange those UEs which are more likely to contain other UEs at the top, so as to remove redundant UEs early, thus reducing the number of subsequent comparisons. Intuitively, the more //-children the CAT has, and the more nodes an UE embeds, the more likely it contains more other UEs. Sorting using the sum of values in a row will also help to arrange UEs in such a way that if $i \leq j$, then row j cannot be an extension of row i, because for any UE f and g, g is an extension of f implies the sum of successor nodes of g is greater than the sum of successor nodes of f. This is partly due to our numbering scheme in which descendants have a larger number than ancestors on each path. This will also explain why we have chosen the breadth-first (rather than depth-first) traversal when numbering the nodes in Q: we want to treat the paths in Q as evenly as possible, but slightly favor paths that have more nodes embedded by some UE. The reason we have chosen $M + 0.5$ rather than ∞ for fully embedded paths can also be explained here: if two or more UEs both fully embed some paths of Q, we are still able to arrange them by the number of nodes they embed on other paths. We chose $M + 0.5$ rather than $M + 1$ in order to distinguish this special value from the *real* successor nodes (which are all integers) and avoid the confusion in the case where there is a node in Q which is numbered $M + 1$. Fig.6 (e) shows the sorted matrix representing all UEs for the V and Q shown in Fig.6 (a),(b). The algorithm then proceeds to check the rows one by one from top to bottom, to see if any other row is contained in the current row (thus can be removed) using Theorem 1.

Example 6. Consider the query Q and the view V in Fig.1. Using the algorithm in [2] we can find 80 UEs. The possible set of successor nodes for the three paths in Q are {13, 7, 4, 1}, {14, 8, 5, 2}, and {15, 12, 9, 6, 3} respectively (Thus $M = 15.5$). For all of these UEs we will have an UE matrix A which contains 80 rows and three columns. Part of the matrix is shown in Fig.1 (c). The first row of the sorted matrix contains the vector (13, 14, 15), where each element in the vector is associated with //. Using Algorithm 1, we will compare this row with all other rows. Since the values in this row are greater than or equal to the corresponding values in other rows, and each value in this row is marked //, and $a_{1,1} = 13 \neq M$, we can remove all other rows. We are left with only one row, which represents a single UE. The minimal cover set contains only this UE.

[4] It is obvious that the CR generated by a UE is determined by the anchor nodes (or their successors) of those paths that are not fully embedded. Also by Lemma 1, UEs that have embed the same set of nodes into V will generate equivalent CRs, so they can be regarded as the same UE.

Algorithm 1 Filtering matrix $A = (a_{i,j})$ using Theorem 1

```
1: sort the rows in $A$ in descending order of the sum of all values in the row, and then
   in descending order of the number of values marked //
2: for (for $i = 1$ to $n$) do
3:   if (row $i$ is not removed) then
4:     for $j = i + 1$ to $n$ and $j \neq i$ do
5:       if row $j$ is not removed then
6:         if for all $s = 1$ to $k$, either $(a_{i,s} = a_{j,s})$ or $(a_{i,s} > a_{j,s}$ and $a_{i,s}$ is marked
           //) then
7:           if $\neg(s = 1 \wedge a_{i,1} = M \wedge a_{j,1} \neq M)$ then
8:             remove row $j$
```

Example 7. Consider the view V and Q in Fig.6. There are 7 UEs which make up the matrix shown in Fig.6 (e). All values in the matrix are marked //. Using Algorithm 1, we can remove the 3rd, 5th, and the last rows when they are compared with the first row; we can then remove the 4th and 6th rows by comparing them with the 2nd row. The remaining two rows turned out to be not contained in each other. So they form the minimal cover set.

5 Conclusion

We showed that the conditions for irredundant UE given in [2] is incorrect, and it is neither sufficient nor necessary. We then provided some sufficient conditions and separate necessary conditions for UE containment. Using these results, we designed a heuristic algorithm for removing redundant UEs from a given set. We demonstrated, using examples, that our algorithms can be significantly faster than the brute-force method of testing UE containment using containment mappings pair by pair.

References

1. A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
2. L. V. S. Lakshmanan, H. Wang, and Z. J. Zhao. Answering tree pattern queries using views. In *VLDB*, 2006.
3. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1), 2004.
4. A. Nash, L. Segoufin, and V. Vianu. Determinacy and rewriting of conjunctive queries using views: A progress report. In *ICDT*, pages 59–73, 2007.
5. J. Wang, R. W. Topor, and M. J. Maher. Rewriting union queries using views. *Constraints*, 10(3):219–251, 2005.
6. J. Wang, J. X. Yu, and C. Liu. Independence of containing patterns property and its application in tree pattern query rewriting using views. To appear in WWWJ.