Answering Tree Pattern Queries Using Views: a Revisit

Junhu Wang
Griffith University, Gold Coast Campus, Australia
J.Wang@griffith.edu.au

Jiang Li
Griffith University, Gold Coast Campus, Australia
Jiang.Li@griffithuni.edu.au

Jeffrey Xu Yu
Chinese University of Hong Kong, China
yu@se.cuhk.edu.hk

ABSTRACT

We revisit the problem of answering tree pattern queries using views. We first show that, for queries and views that do not have nodes labeled with the wildcard *, there is an alternative to the approach of query rewriting which does not require us to find any rewritings explicitly yet which produces the same answers as the maximal contained rewriting. Then, using the new approach, we give a simple criterion and a corresponding algorithm for identifying redundant view answers, which are view answers that can be ignored when evaluating the maximal contained rewriting. Finally, for queries and views that do have nodes labeled *, we provide a method to find the maximal contained rewriting and show how to answer the query using views without explicitly finding the rewritings.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query processing

General Terms

Theory, Algorithms

Keywords

XML, XPath, tree pattern, view, containment, rewriting

1. INTRODUCTION

Answering queries using views has been studied extensively for relational databases and found applications in fields such as query optimization, data integration, and query answering when the original data is inaccessible, e.g., in a network environment. Motivated by similar potential applications, the problem of answering tree pattern queries using views has recently attracted attention from many researchers.

A main approach for answering tree pattern queries using views exploits the technique of query rewriting, which transforms a query Q into a set of new queries, and then evaluates these new queries over the materialized answers to the view [15, 9, 8, 4, 6]. In the literature, there are two types of rewritings, equivalent rewritings and contained rewritings. An equivalent rewriting will produce all answers to the original query, and a contained rewriting may produce only part of the answers. This work is about/closely related to contained rewritings. Previously [8] gave a method to find the maximal contained rewriting (MCR) (i.e., the union of all contained rewritings) when both the view and query are in $P^{\{/,//,[]\}}$ (i.e., they do not have nodes labeled with the wildcard *), and [11] gave a method for finding the maximal contained rewriting for views in $P^{\{/,//,*,[]\}}$ and queries that have no *-nodes connected to a //-edge and no leaf node u such that u and the parent of u are both labeled *. To actually find the answers to a new query Q, we need to evaluate the rewritings over the materialized view answers. A question arises as to whether it is possible to answer a new query without finding the rewritings. Furthermore, all previous works require us to materialize all answers to the view and evaluate the MCR over all such answers. Since an answer to a tree pattern is a subtree of the original data tree, and some answers may be subtrees of other answers, it is likely that we do not need to evaluate the MCR over *all* of the view answers. In other words some view answers may be *redundant* in that any answers that can be found by evaluating the MCR over them can also be found by evaluating the MCR over other view answers. In our experiments we found that on average 36.91% of view answers for the dataset XMark are redundant, 73.67% percent of view answers for the BIOML dataset are redundant, and up to 69.48% of view answers can be redundant for the dataset Treebank (See Section 4.3). Identifying view answers which do not contribute to the answering of new queries will help us minimize view maintenance and speed-up query evaluation. In this paper, we study these issues and make the following contributions:

1. For query Q and view V in $P^{\{/,//,[]\}}$ we show there is an alternative approach for answering Q using V which does not require us to find the MCR of Q using V explicitly, yet which produces the same set of answers to Q as can be found by the MCR.
2. For Q and V in $P^{\{/,//,[]\}}$, we provide a simple criteria for identifying *redundant* view answers which are view answers that can be ignored when evaluating the MCR. We also provide two procedures to find the redundant view answers, one is fast but incomplete, the other is less efficient but complete. We conducted experiments about redundant view answers which in-

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

dicate that for some real XML datasets and randomly generated views, there can be a large percentage of redundant view answers, and most of the redundant view answers can be removed using the fast procedure.

3. We show how the above results can be extended to cases where V, Q are in $P^{\{/,//,*,[]\}}$, but Q does not have /-edges, or Q does not have *-nodes connected to //-edges or leaf *-nodes.

4. For the more general case where both Q and V can be any pattern in $P^{\{/,//,*,[]\}}$, we present a method to find the MCR of Q using V, and show how to answer Q using answers to V without finding explicit rewritings.

The rest of paper is organized as follows. Section 2 gives the preliminaries. Section 3 presents the new approach for answering tree patterns queries using views when the view and query are both in $P^{\{/,//,[]\}}$. Based on the new approach, Section 4 provides the condition and an algorithm for finding redundant view answers. In Section 5 we study queries and views in $P^{\{/,//,*,[]\}}$. We first give a method for finding the MCR, then show how to answer Q using V without rewriting, and then discuss removing redundant view answers with respect to other classes of patterns than $P^{\{/,//,[]\}}$. Section 6 compares our work with some closely related work. We conclude the paper in Section 7.

2. PRELIMINARIES

2.1 XML data tree and tree patterns

Let Σ be an infinite set of tags which does not contain the symbol *. An XML *data tree*, or simply a *data tree*, is a tree in which every node is labeled with a tag in Σ. A *tree pattern (TP)*, or simply a *pattern*, in $P^{\{/,//,*,[]\}}$ is a tree in which every node is labeled with a symbol in $\Sigma \cup \{*\}$, every edge is labeled with either / or //, and there is a unique *selection node*. The path from the root to the selection node is called the *selection path*. Figure 1 shows some example TPs, where single and double lines are used to represent /-edges and //-edges respectively, and a circle is used to indicate the selection node. A TP corresponds to an XPath expression. The TPs in Figure 1 (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 $\mathtt{sn}(\mathtt{P})$, and $\mathtt{sp}(\mathtt{P})$ to denote the selection node and the selection 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 x an *x-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 a data tree t is a mapping δ from $N(P)$ to $N(t)$ that is (1) *root-preserving*: $\delta(rt(P)) = rt(t)$, (2) *label-preserving*: $\forall v \in N(P)$, $label(v) = label(\delta(v))$ or $label(v) = *$, and (3) *structure-preserving*: for every /-edge (x, y) in P, $\delta(y)$ is a child of $\delta(x)$; for //-edge (x, y), $\delta(y)$ is a descendant of $\delta(x)$. Each matching δ produces a subtree of t rooted at $\delta(\mathtt{sn}(\mathtt{P}))$, 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 S of data trees, we define $P(S) = \cup_{t \in S} P(t)$.

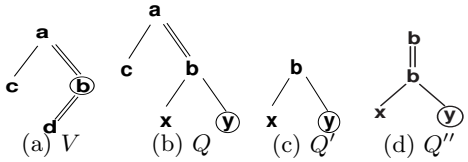


Figure 1: View V, Query Q, and two CRs Q' and Q''

Let $P_1, \ldots, P_n$ be TPs. We use $P_1 \cup \cdots \cup P_n$ and $P_1 \cap \cdots \cap P_n$ to represent the *union* and *intersection* of $P_1, \ldots, P_n$ respectively. The meaning of union and intersection is as usual: for any set of data trees S,
$(P_1 \cup \cdots \cup P_n)(S) = P_1(S) \cup \cdots \cup P_n(S)$, and
$(P_1 \cap \cdots \cap P_n)(S) = P_1(S) \cap \cdots \cap P_n(S)$.

2.2 Tree pattern 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 data tree t, $P(t) \subseteq Q(t)$. 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 $\delta(\mathtt{sn}(\mathtt{Q})) = \mathtt{sn}(\mathtt{P})$.

Several subsets of $P^{\{/,//,*,[]\}}$ are of special interest to us: $P^{\{/,//,[]\}}$ is the set of TPs that do not have *-nodes; $\widehat{P}^{\{/,//,[\,],*\}}$ is the set of TPs in which all *-nodes are incident only to /-edges and are non-leaf; , $P^{\{//,[\,],*\}}$ is the set of TPs that do not have /-edges, and $P^{\{/,[\,],*\}}$ is the set of TPs that do not have //-edges. Let P, Q be TPs in $P^{\{/,//,*,[]\}}$. It is well known that, in general, $P \subseteq Q$ if but not only if there is a CM from Q to P. However, if $P \in P^{\{/,[\,],*\}}$ or $Q \in \widehat{P}^{\{/,//,[\,],*\}}$, then $P \subseteq Q$ iff there is a CM from Q to P [10, 14]. Note that $P^{\{/,//,[]\}}$ is a subset of $\widehat{P}^{\{/,//,[\,],*\}}$.

2.3 Contained rewritings of tree pattern queries using views

A *view* is an existing TP. Let V be a view and Q be a TP. In this paper we will implicitly assume that $label(rt(Q)) = label(rt(V)) \neq *$. A *contained rewriting (CR)* of Q using V is a TP Q' such that (1) for any data tree t, $Q'(V(t)) \subseteq Q(t)$; (2) there exists a data tree t such that $Q'(V(t)) \neq \emptyset$; Figure 1 shows a view V, a TP Q, and two CRs, Q' and Q'', of Q using V. The *maximal contained rewriting (MCR)* of Q using V, denoted $\mathsf{MCR}(Q, V)$, is defined to be the union of all CRs of Q using V.

Let Q' be a CR of Q using V. It is clear that $label(rt(Q')) = label(\mathtt{sn}(\mathtt{V}))$ or at least one of $label(rt(Q'))$ and $label(\mathtt{sn}(\mathtt{V}))$ will be *. We use $Q' \circ V$ to represent the TP obtained by merging $rt(Q')$ and $\mathtt{sn}(\mathtt{V})$. The merged node will be labeled with $label(rt(Q'))$ if $label(rt(Q')) \neq *$, otherwise it will be labeled with $label(\mathtt{sn}(\mathtt{V}))$. The selection node of $Q' \circ V$ is that of Q'. Note that condition (1) in the definition of CR is equivalent to $Q' \circ V \subseteq Q$.

Given TP Q and view V in $P^{\{/,//,[]\}}$, [8] shows that the existence of a CR of Q using V can be characterized by the

existence of a *useful embedding* from Q to V. In brief, a *useful embedding* from Q to 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 that 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 selection path $\mathtt{sp(Q)}$, if $f(x)$ is defined, then $f(x) \in \mathtt{sp(V)}$, and if $f(\mathtt{sn(Q)})$ is defined, then $f(\mathtt{sn(Q)}) = \mathtt{sn(V)}$. (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 $f(x) \in \mathtt{sp(P)}$, and y the child of x on p (call y the *anchor node*), then either $f(x) = \mathtt{sn(V)}$, or the edge (x, y) is a //-edge.

Given a useful embedding f from Q to V, a *Clip-Away Tree (CAT)*, denoted CAT_f, can be constructed as follows. (i) Construct the root of CAT_f and label it with $label(\mathtt{sn(V)})$. (ii) For each path $p \in Q$ that is not fully embedded, find the anchor node y and attach the subtree of Q rooted at y (denoted Q_y) under $rt(\mathsf{CAT}_f)$ by connecting $rt(Q_y)$ and $rt(\mathsf{CAT}_f)$ with the same type of edge as that between y and its parent. The selection node of CAT_f is the node corresponding to $\mathtt{sn(Q)}$. For example, in Figure 1 the query Q has two useful embeddings to V, both of them map the a-node and c-node in Q to the a-node and c-node in V, and one also maps the b-node in Q to the b-node in V, but the other does not. The corresponding CATs are shown in Figure 1 (c) and (d) respectively. Each CAT is a CR of Q using V, and every CR of Q using V is contained in one of these CATs. Thus if $h_1, \ldots, h_k$ are all the useful embeddings from Q to V, then $\mathsf{MCR}(Q, V) = \bigcup_{i \in [1,k]} \mathsf{CAT}_{h_i}$.

3. EVALUATING TPS OVER ANNOTATED VIEW ANSWERS - AN ALTERNATIVE TO QUERY REWRITING

Given view V, new query Q, the rewriting approach finds a set of new queries (i.e., the rewritings) and evaluates the rewritings over the materialized view answers in order to find answers to Q. In this section, we present an alternative approach for answering Q using only the answers to V. The new approach does not require us to find any explicit rewritings, instead, it evaluates Q over a set of annotated view answers. We will show that the new approach produces the same set of answers as evaluating the MCR over the original view answers in $V(t)$. We focus on queries and views in $P^{\{/,//,[]\}}$ in this section and will deal with queries and views in $P^{\{/,//,*,[]\}}$ in Section 5.

Definition 3.1: (**annotated view answer**) Let $V \in P^{\{/,//,[]\}}$ be a view, t be a data tree, and t_i be a view answer in $V(t)$. We construct a new tree from V and t_i, denoted $t_i \cdot V$, by merging the root of t_i with the selection node of V. The merged node will be labeled with $label(rt(t_i))$. We call this tree an *annotated answer* of V over t or an *annotated view answer* when V and t are clear from context. □

Figure 2 (a) shows a data tree t. There are two answers, t_1 and t_2, to the view V shown in Figure 1 (a), as indicated by the surrounding boxes. The annotated view answers are shown in Figure 2 (b) and (c).

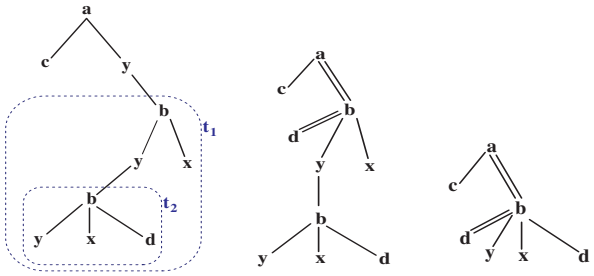


Figure 2: Data tree t and the annotated view answers for the view V in Figure 1

An annotated view answer will be treated like a data tree and used to find answers to a query over the original data tree t, hence it has no selection node. The evaluation of a query over an annotated view answer is similar to that over a normal data tree. Formally, for any query Q, we evaluate Q over $t_i \cdot V$ by finding *matchings* of Q in $t_i \cdot V$. A matching of Q in $t_i \cdot V$ is a mapping from $N(Q)$ to $N(t_i \cdot V)$ which is root-preserving, label-preserving, structure-preserving and which *maps the selection node of Q into a node in t_i*. If h is such a matching, then the subtree of t_i rooted at $h(\mathtt{sn(Q)})$ is called an answer of Q over $t_i \cdot V$. We use $Q(t_i \cdot V)$ to denote the set of all answers of Q over $t_i \cdot V$.

The annotated view answers have the following properties.

Lemma 3.1: *Let $V \in P^{\{/,//,[]\}}$ be a view, t be a data tree, and $t_i \in V(t)$. Then $t_i \in V(t_i \cdot V)$.* □

Proposition 3.1: Let $V \in P^{\{/,//,[]\}}$ be a view, t be a data tree, and $V(t) = \{t_1, \ldots, t_n\}$. Let $Q \in P^{\{/,//,[]\}}$ be a query. If Q has no CR using V, then $Q(t_i \cdot V) = \emptyset$ for all $i \in [1, n]$. □

The proof is straightforward because any matching of Q in $t_i \cdot V$ implies a useful embedding from Q to V: if we restrict the matching to those nodes of Q whose image is in the V-part of $t_i \cdot V$, the matching becomes a useful embedding from Q to V.

Theorem 3.1: Let $V \in P^{\{/,//,[]\}}$ be a view, t be a data tree, and $V(t) = \{t_1, \ldots, t_n\}$. Let $Q \in P^{\{/,//,[]\}}$ be a query, $h_1, \ldots, h_k$ $(k \geq 1)$ be all of the useful embeddings from Q to V. Then

(1) $Q(t_i \cdot V) = \mathsf{CAT}_{h_1}(t_i) \cup \cdots \cup \mathsf{CAT}_{h_k}(t_i), \forall i \in [1, n]$.

(2) $\mathsf{CAT}_{h_1}(V(t)) \cup \cdots \cup \mathsf{CAT}_{h_k}(V(t)) = Q(t_1 \cdot V) \cup \cdots \cup Q(t_n \cdot V)$.

□

PROOF. Denote CAT_{h_j} by Q_j for $j = 1, \ldots, k$. (1) We show $Q_1(t_i) \cup \cdots \cup Q_k(t_i) \subseteq Q(t_i \cdot V)$ and $Q(t_i \cdot V) \subseteq Q_1(t_i) \cup$

$\cdots \cup Q_k(t_i)$. First, for every matching h of Q_j in t_i, we can construct a matching h' of Q in $t_i \cdot V$ as follows: for every node $u \in Q$, if u is mapped into the V-part of $t_i \cdot V$ by h_j, then let $h'(u) = h_j(u)$. If u is not mapped into V by h_j, then it will be a node in CAT_{h_j} $(= Q_j)$, and we let $h'(u) = h(u)$. Clearly h' is a matching of Q in $t_i \cdot V$, and $h(\mathtt{sn}(\mathtt{Q_j}))$ and $h'(\mathtt{sn}(\mathtt{Q}))$ are the same node in t_i. Therefore, $Q_j(t_i) \subseteq Q(t_i \cdot V)$. Thus $Q_1(t_i) \cup \cdots \cup Q_k(t_i) \subseteq Q(t_i \cdot V)$.

Conversely, for every matching δ of Q in $t_i \cdot V$, there is useful embedding h_j from Q to V, which is obtained by restricting δ to those nodes of Q that are mapped into the V-part of $t_i \cdot V$ by δ. The corresponding CAT CAT_{h_j} $(= Q_j)$ has a matching in t_i, which is obtained by restricting δ to those nodes of Q that are mapped into the t_i-part of $t_i \cdot V$. This matching maps $\mathtt{sn}(\mathtt{Q_j})$ to the same node as $\delta(\mathtt{sn}(\mathtt{Q}))$. Therefore, $Q(t_i \cdot V) \subseteq Q_1(t_i) \cup \cdots \cup Q_k(t_i)$.

(2)

$$
\begin{aligned}
& Q_1(V(t)) \cup \cdots \cup Q_k(V(t)) \\
= \; & (Q_1(t_1) \cup \cdots \cup Q_1(t_n)) \cup \cdots \cup (Q_k(t_1) \cup \cdots \cup Q_k(t_n)) \\
= \; & (Q_1(t_1) \cup \cdots \cup Q_k(t_1)) \cup \cdots \cup (Q_1(t_n) \cup \cdots \cup Q_k(t_n)) \\
= \; & Q(t_1 \cdot V) \cup \cdots \cup Q(t_n \cdot V). \quad /* \, by \ (1)
\end{aligned}
$$

□

Proposition 3.1 and Theorem 3.1 imply that, if Q and V are both in $P^{\{/,//,[]\}}$, then evaluating Q directly over the annotated view answers (hereafter referred to as the *annotated view answer* approach) will produce the same answers that can be found by evaluating $\mathsf{MCR}(Q, V)$ over the original view answers, thus proving our claim at the beginning of this section.

Variation of annotated view answer Let us use V^U to denote the pattern obtained from view V by removing all nodes under $\mathtt{sn}(\mathtt{V})$, and V^L to denote the pattern consisting of $\mathtt{sn}(\mathtt{V})$ and all nodes and edges under $\mathtt{sn}(\mathtt{V})$. It is not difficult to see that Proposition 3.1 and Theorem 3.1 still hold if we replace $t_i \cdot V$ with $t_i \cdot V^U$ (for $i = 1, \ldots, n$). This is because t_i is a tree in $V(t)$, thus there is a matching of V^L in t_i.

Discussion on evaluation and performance Assuming naive algorithms for query evaluation for both the rewriting approach and the annotated view answer approach, there are extreme cases where one approach is faster than the other. For instance, when the number of non-redundant CATs is small and the number of view answers is large, the rewriting approach appears faster. In particular, if there are no useful embeddings from Q to V, then the rewriting approach knows immediately that no answers to Q can be obtained using the view answers (although it has the cost to find out that there are no useful embeddings), but the annotated view answer approach still has to evaluate Q over each annotated view answer. On the other hand, in cases where the number of non-redundant CRs is large and the number of view answers is small, the annotated view answer approach appears faster. We use the following example to illustrate this point.

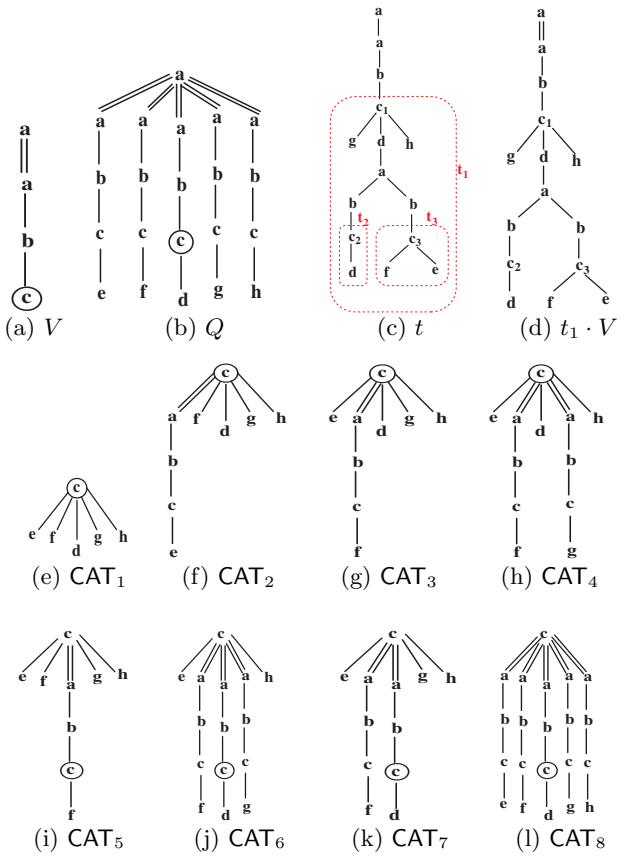


Figure 3: V, Q, t and $t_1 \cdot V$ for Example 3.1

Example 3.1: Consider the query Q and view V in Figure 3. There are a total of 32 $(=2^8)$ non-redundant CATs of Q using V. Some example CATs are listed in Figure 3 (e) to (l). Consider the data tree t shown in Figure 3 (c). It is easy to verify that $V(t)$ has three view answers t_1, t_2 and t_3, rooted at nodes c_1, c_2 and c_3 respectively (we use subscripts to indicate nodes with the same label), and t_2 and t_3 are redundant (see next section for redundant view answers). If we evaluate Q directly over the annotated view answer $t_1 \cdot V$ (Figure 3 (d)), we will quickly obtain two answers, one rooted at c_1 and the other rooted at c_2, and the evaluation will take less time than evaluating all 32 CATs over t_1. □

In the above example, the better efficiency of the new approach comes from the fact that when evaluating Q directly over $t_1 \cdot V$ we only need to verify whether $\mathtt{sn}(\mathtt{Q})$ can be matched to each of the c-nodes by some matching, and in doing so, we can avoid many repeated computations that are done when we evaluate all CATs over t_1. Take c_3 for example, if we evaluate Q over $t_1 \cdot V$, we find that if we map $\mathtt{sn}(\mathtt{Q})$ to c_3, then the d-node under $\mathtt{sn}(\mathtt{Q})$ cannot be mapped to anywhere. Thus we know immediately that c_3 is not in $Q(t_1 \cdot V)$. On the other hand, if we evaluate all of the CATs over t_1, we in effect have to repeat the same comparison 32 times. Similarly with c_2, if we evaluate Q over $t_1 \cdot V$, once we find a matching that maps $\mathtt{sn}(\mathtt{Q})$ to c_2, we do not need to consider other matchings that also map $\mathtt{sn}(\mathtt{Q})$ to c_2. However, if we evaluate all of the CATs over t_1, we in effect have

to find all matchings that produce c_2.

A more careful comparison of performance requires us to design better algorithms for both approaches and there are obvious ways to do so. For instance, for the rewriting approach, for each view answer t_i we can first find a set of *potential nodes* which are the nodes with the same label as $\mathtt{sn}(\mathtt{Q})$, and filter out those potential nodes u such that the subtree of Q rooted at $\mathtt{sn}(\mathtt{Q})$ has no matching in the subtree rooted at u. Then we check the remaining potential nodes, and if one is found to be an answer to Q using one of the CATs, then there is no need to check it using other CATs. A more detailed discussion on the valuation of MCRs can be found in [16]. For the annotated view answer approach, once we have found a matching δ of Q in $t_1 \cdot V$, we can extract the part δ' of δ which maps the nodes in Q to the V-part, and reuse it to find other matchings or when we evaluate Q over $t_2 \cdot V$. Since every matching of Q in an annotated view answer implies a useful embedding, algorithms can be designed for the annotated view answer approach that takes, at most, the same amount of time as for finding the CATs and evaluating the CATs over the view answers in $V(t)$.

One application of the new approach is that it enables us to find a simple condition to identify redundant view answers, as will be discussed in the next section.

4. REDUNDANT VIEW ANSWERS

The previous section shows that to answer a new query Q using answers to V, we can either evaluate the rewritings over the original view answers in $V(t)$, or evaluate Q over the annotated view answers. In practice, $V(t)$ may be a large set and many view answers may be subtrees of other view answers. Thus a natural question to ask is whether we need to evaluate the rewritings over all of the view answers. We will address this problem in this section.

Definition 4.1: (**View answer subsumption and redundancy**) Let V be a view, and $\mathcal{P}$ be a class of TPs. Suppose $t_1, \ldots, t_m \in V(t)$ are view answers. If for every TP $Q \in \mathcal{P}$, $\mathsf{MCR}(Q,V)(t_m) \subseteq \mathsf{MCR}(Q,V)(t_1) \cup \cdots \cup \mathsf{MCR}(Q,V)(t_{m-1})$, then we say t_m is *subsumed* by $t_1, \ldots, t_{m-1}$ wrt $\mathcal{P}$. A view answer is said to be *redundant* wrt $\mathcal{P}$ if it is subsumed by some other view answers wrt $\mathcal{P}$. □

Intuitively, all answers that can be obtained by evaluating $\mathsf{MCR}(Q,V)$ (for all $Q \in \mathcal{P}$) over the redundant view answers can be obtained by evaluating $\mathsf{MCR}(Q,V)$ over some other view answers.

4.1 Conditions for subsumption

Next we investigate conditions under which t_m is subsumed by $t_1, \cdots, t_{m-1}$. We limit our discussion to TPs in $P^{\{/,//,[]\}}$, i.e, $V \in P^{\{/,//,[]\}}$ and $\mathcal{P} = P^{\{/,//,[]\}}$. In this case, due to results in Section 3, t_m is subsumed by $t_1, \cdots, t_{m-1}$ if and only if $\forall Q \in P^{\{/,//,[]\}}, Q(t_m \cdot V) \subseteq Q(t_1 \cdot V) \cup \cdots \cup Q(t_{m-1} \cdot V)$.

We start with the case where $m = 2$. Obviously, if t_2 is not a subtree of t_1, then it cannot be subsumed by t_1. Therefore, when looking for subsumed view answers we only need to examine view answers which are subtrees of other view answers. In what follows, we will use $t_2 \prec t_1$ to denote t_2 is a subtree of t_1. The following example shows that, in general, $t_2 \prec t_1$ does not imply t_2 is subsumed by t_1.

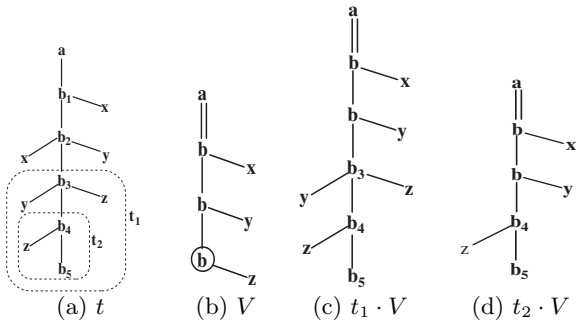


Figure 4: $V(t_2 \cdot V) \nsubseteq V(t_1 \cdot V)$ **although** $t_2 \prec t_1$

Example 4.1: Consider the data tree t and view V shown in Figure 4 (a) and (b) respectively. For convenience we use subscripts to indicate different nodes labeled with the same label. $V(t)$ contains two subtrees of t, one rooted at b_3 (denoted t_1) and the other rooted at b_4 (denoted t_2). t_2 is a subtree of t_1. Let Q be the same as V, then $Q(t_2 \cdot V)$ contains the subtree of t rooted at b_4, but $Q(t_1 \cdot V)$ does not. □

We now prove the following theorem, which provides a necessary and sufficient condition for t_2 to be subsumed by t_1.

Theorem 4.1: Let $t_1, t_2 \in V(t)$ and t_2 be a subtree of t_1. The following three conditions are equivalent:

(1) $V(t_2 \cdot V) \subseteq V(t_1 \cdot V)$

(2) $t_2 \in V(t_1 \cdot V)$

(3) $\forall Q, Q(t_2 \cdot V) \subseteq Q(t_1 \cdot V)$

□

PROOF. (1) → (2): By Lemma 3.1 $t_2 \in V(t_2 \cdot V)$. Thus if $V(t_2 \cdot V) \subseteq V(t_1 \cdot V)$, then $t_2 \in V(t_1 \cdot V)$.

(2) → (3): Suppose $t_2 \in V(t_1 \cdot V)$. That is, there is a matching h of V in $t_1 \cdot V$ such that $h(\mathtt{sn}(\mathtt{V})) = \mathtt{rt}(\mathtt{t_2})$. Let us extend h to a mapping from $N(t_2 \cdot V)$ to $N(t_1 \cdot V)$ by mapping every node in the t_2-part of $t_2 \cdot V$ to the corresponding node in the t_2-part of $t_1 \cdot V$. Now for any query Q, suppose f is a matching of Q in $t_2 \cdot V$, then hf is a matching of Q in $t_1 \cdot V$. Furthermore, $f(\mathtt{sn}(\mathtt{Q}))$ and $hf(\mathtt{sn}(\mathtt{Q}))$ correspond to the same node in t_2. Therefore, $Q(t_2 \cdot V) \subseteq Q(t_1 \cdot V)$.

(3) → (1): if $\forall Q, Q(t_2 \cdot V) \subseteq Q(t_1 \cdot V)$, then we can take V as Q and obtain $V(t_2 \cdot V) \subseteq V(t_1 \cdot V)$.

□

The above theorem indicates that to test whether one view answer t_2 is subsumed by another view answer t_1, we only need to test whether t_2 is in $V(t_1 \cdot V)$. For example, for the

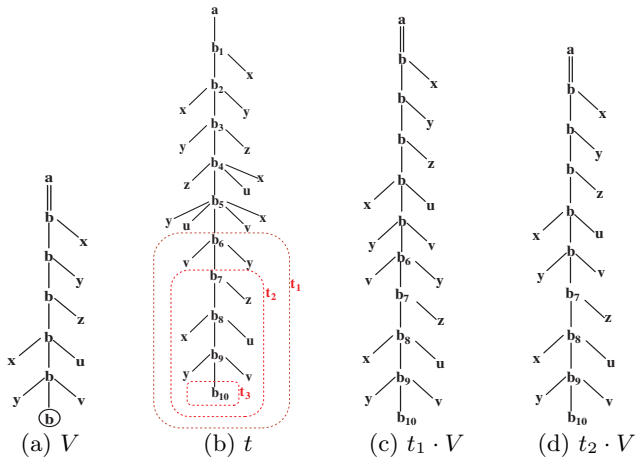


Figure 5: $t_1, t_2, t_3 \in V(t)$, $t_3 \prec t_2 \prec t_1$, **and** $t_3 \in V(t_2 \cdot V)$, **but** $t_3 \notin V(t_1 \cdot V)$

view V and tree t in Figure 3, it is easy to verify that both t_2 and t_3 are in $V(t_1 \cdot V)$, hence they are subsumed by t_1.

Using the above theorem, we can show that t_m is subsumed by $t_1, \ldots, t_{m-1}$ if and only if it is subsumed by one of the them.

Corollary 4.1: Suppose $t_1, \ldots, t_m \in V(t)$, $t_m \prec t_i$ for $i = 1, \ldots m-1$. If $\forall Q, Q(t_m \cdot V) \subseteq Q(t_1 \cdot V) \cup \cdots \cup Q(t_{m-1} \cdot V)$, then there exists $i \in [1, m-1]$ such that $\forall Q, Q(t_m \cdot V) \subseteq Q(t_i \cdot V)$. □

Proof. Suppose $\forall Q, Q(t_m \cdot V) \subseteq Q(t_1 \cdot V) \cup \cdots \cup Q(t_{m-1} \cdot V)$, then $V(t_m \cdot V) \subseteq V(t_1 \cdot V) \cup \cdots \cup V(t_{m-1} \cdot V)$, hence $t_m \in V(t_1 \cdot V) \cup \cdots \cup V(t_{m-1} \cdot V)$. Therefore there exists $i \in [1, m-1]$ such that $t_m \in V(t_i \cdot V)$. By Theorem 4.1, $\forall Q, Q(t_m \cdot V) \subseteq Q(t_i \cdot V)$. □

Now suppose $t_1, t_2, t_3 \in V(t)$, and $t_3 \prec t_2 \prec t_1$. If $t_3 \in V(t_2 \cdot V)$ and $t_2 \in V(t_1 \cdot V)$, then by Theorem 4.1, $V(t_2 \cdot V) \subseteq V(t_1 \cdot V)$, hence $t_3 \in V(t_1 \cdot V)$. But the following examples demonstrate that $t_3 \in V(t_2 \cdot V)$ does not imply $t_3 \in V(t_1 \cdot V)$, and $t_2 \in V(t_1 \cdot V)$ does not imply $t_3 \in V(t_1 \cdot V)$. In other words, t_3 is subsumed by t_2 or t_2 is subsumed by t_1 does not imply t_3 is subsumed by t_1. This observation is useful when we design the algorithm to find redundant view answers.

Example 4.2: Consider the example t and V in Figure 5. For easy identification of the nodes, we use subscripts to indicate distinct nodes with the same label, i.e., the nodes $b_1, \ldots, b_{10}$ are all labeled with b. It is easy to verify that there are three subtrees of t in $V(t)$, which are the subtrees rooted at b_6, b_7 and b_{10} respectively. We denote these subtrees by t_1, t_2 and t_3. Although $t_3 \prec t_2 \prec t_1$ and $t_3 \in V(t_2 \cdot V)$, t_3 is not in $V(t_1 \cdot V)$. □

Example 4.3: Consider the example t and V in Figure 6. In the figure the nodes $b_1, \ldots, b_{10}$ are all labeled with b. It is easy to verify that there are three subtrees of t in $V(t)$, which

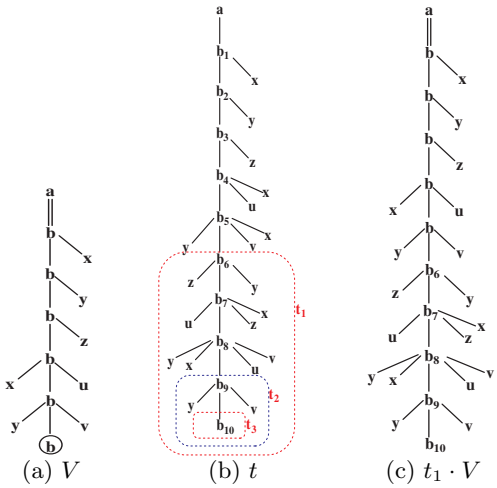


Figure 6: $t_1, t_2, t_3 \in V(t)$, $t_3 \prec t_2 \prec t_1$ **and** $t_2 \in V(t_1 \cdot V)$, **but** $t_3 \notin V(t_1 \cdot V)$

are the subtrees rooted at b_6, b_9 and b_{10} respectively. We denote these subtrees by t_1, t_2 and t_3. Although $t_3 \prec t_2 \prec t_1$, and $t_2 \in V(t_1 \cdot V)$, t_3 is not in $V(t_1 \cdot V)$. □

Using Theorem 4.1 and the above observations, we can design an algorithm for finding all redundant view answers in $V(t)$: to see whether $t_2 \in V(t)$ is redundant, we only need to test whether there exists $t_1 \in V(t)$ such that $t_2 \prec t_1$ and $t_2 \in V(t_1 \cdot V)$, and we may have to test t_2 against all view answers of which t_2 is a subtree. However, testing $t_2 \in V(t_1 \cdot V)$ can sometimes be costly. Next we will provide a more efficient (although incomplete) method for identifying redundant view answers.

Suppose $t_1, t_2 \in V$ and t_2 is a subtree of t_1. The corollary below gives a sufficient condition under which t_2 is subsumed by t_1.

Proposition 4.1: Let t_1, t_2 be in $V(t)$, and t_2 be a subtree of t_1. Let (x, y) be the last //-edge on $\mathtt{sp(V)}$ (counting from $rt(V)$). If the length of the path from y to $\mathtt{sn(V)}$ is less or equal to the length of the path from $rt(t_1)$ to $rt(t_2)$, then $t_2 \in V(t_1 \cdot V)$. □

Proof sketch. Let f be the matching of V in t that generates t_2. Since the length of the path from y to $\mathtt{sn(V)}$ is less or equal to the length of the path from $rt(t_1)$ to $rt(t_2)$, and there are only /-edges on the path from y to $\mathtt{sn(V)}$, all nodes from y to $\mathtt{sn(V)}$ will be mapped into t_1, and the subtree of V rooted at y, V_y, will be mapped into t_1 by f. Therefore, we can construct a mapping h from V to $t_1 \cdot V$ such that for every node u in V_y, $h(u) = f(u)$, and for every other node $u' \in N(V) - N(V_y)$, h maps u' to the corresponding node in the V part of $t_1 \cdot V$. Since (x, y) is a //-edge, h is a matching of V in $t_1 \cdot V$, and $h(\mathtt{sn(V)}) = \mathtt{rt(t_2)}$. Therefore $t_2 \in V(t_1 \cdot V)$. □

Let us call the path from y to $\mathtt{sn(V)}$ in Proposition 4.1 the *last /-segment* of V. Using the proposition and Therorem 4.1,

if the length of the last /-segment of V is not longer than the path from $rt(t_1)$ to $rt(t_2)$, than t_2 is subsumed by t_1. For example, for the view and data tree in Figure 3, the last /-segment (i.e., the path from the second a-node to the selection node) of V is shorter than the path from $rt(t_1)$ to $rt(t_2)$. Therefore t_2 is subsumed by t_1. Similarly t_3 is subsumed by t_1.

A special case of Proposition 4.1 is when the selection node `sn(V)` is connected to its parent with a //-edge. In this case, all view answers in $V(t)$ that are subtrees of other view answers are redundant.

4.2 The algorithm

Based on Theorem 4.1 and Proposition 4.1 we design an algorithm to remove redundant view answers from $V(t)$. Let us use L_V to denote the length of the last /-segment on `sp(V)`, and $d(t_1, t_2)$ to denote the length of the path from $rt(t_1)$ to $rt(t_2)$, if t_2 is a subtree of t_1. The removal of redundant view answers is done by two scans. The first, called `FastScan`, scans the view answers using Proposition 4.1 to remove some redundant view answers in $V(t)$ quickly, and the second, called `CompleteScan`, uses Theorem 4.1 to remove the remaining redundant view answers. For easy exposition, we assume the view answers in $V(t)$ have been arranged into lists where each list consists of the view answers $t_1, \ldots, t_n$ such that $t_n \prec t_{n-1} \prec \cdots \prec t_1$. We associate each t_i with a Boolean variable $t_i.del$ which indicates whether t_i has been deleted. Initially $t_i.del$ =FALSE for all $i \in [1, n]$. The input to each scan is such a list, and the output is the same list with possibly modified $t_i.del$ values. The two scans are listed in Algorithm 1.

Implementation and Complexity: To implement Algorithm 1, we organize the view answers into disjoint subtrees of t, two view answers are in the same subtree if and only if one is a subtree of another or they are both subtrees of another view answer. We use an attribute $u.ans$ for each node u to indicate whether u is the root of some view answer in $V(t)$. To apply `FastScan`, we check the nodes which are roots of some view answers level by level and top-down. If one view answer is found redundant, then all view answers below it will also be redundant. In the worst case, every view answer (except those which are not subtrees of others) will be checked once. Thus `FastScan` is linear in the number of view answers in $V(t)$, that is, it takes $O(|V(t)|)$. Checking whether t_2 is in $V(t_1 \cdot V)$ can be done in $|V||t_1 \cdot V|$ using a variation of Algorithm 3 of [10], and `CompleteScan` needs to make $(m-1)m/2$ such tests. In the worst case the roots of all view answers in $V(t)$ are on the same path and $n = |V(t)|$, and `FastScan` finds no redundant view answers (thus $m = |V(t)|$) and we have to resort to `CompleteScan`. Therefore the worst-case complexity of Algorithm 1 is $O(|V(t)|^2|V|(|t_1| + |V|))$ where t_1 is the largest view answer (i.e., one with the largest number of nodes) in $V(t)$.

4.3 Experiments

The purpose of our experiments is to get some insight on the percentage of view answers which are redundant, and the percentage of redundant view answers that can be found using `FastScan`, as well as the time required to find the redundancies, for some real datasets. We implemented `FastScan` and `CompleteScan` with C++, and all our experiments were carried out on a Dell Latitude D430 laptop with Intel(R) 1.20GHz CPU and 2G RAM running Windows XP.

Algorithm 1 Removing redundant view answers

INPUT: $t_n, \ldots, t_1 \in V(t)$ s. t. $t_n \prec t_{n-1} \prec \cdots \prec t_1$

```
/* FastScan
1: m ← n
2: for i = 2 to n do
3:   if L_V ≤ d(t_1, t_i) then
4:     t_i.del, t_{i+1}.del, ..., t_n.del ← TRUE
5:     m ← i − 1
6:     go to CompleteScan
/* CompleteScan
7: for i = 1 to m − 1 do
8:   if t_i.del =FALSE then
9:     for j = i + 1 to m do
10:      if t_j.del =FALSE and t_j ∈ V(t_i · V) then
11:        t_j.del ← TRUE
```

Datasets and view sets Obviously for there to be redundant view answers in $V(t)$ the data tree t must have recursion (i.e., there must be root-to-leave paths which contain multiple occurrences of the same label), and the view V must have //-edges. The depth of t, the number of occurrences of $label(\texttt{sn(V)})$ in t, and the size of V will all impact on the number of redundancies. In our experiments we used three datasets, i.e., the benchmark dataset XMark [1], a real dataset TreeBank [2], and a dataset generated from the BIOML DTD [3] using IBM XML generator[1]. For XMark, the dataset is around 111MB, contains more than 1.6 million elements and has limited recursion. For TreeBank, the data tree is deep and has a large number of recursive elements. The maximal depth is 36 and there are more than 2.4 million elements. For BIOML, the generated data tree is around 344MB with a depth of 19 and contains more than 2.46 million elements. For TreeBank we randomly generated 40 views of depth between 2 and 10. For XMark we randomly generated 30 views of depth between 2 and 5 involving the recursive elements `parlist` and/or `listitem` on the selection path, and for BIOML we generated 40 views of depth between 3 and 7. A few example views are listed in Table 1 (three views for each dataset Treebank, XMark and BIOML).

Experimental result The result of our experiments is summarized in Table 2. For TreeBank, on average 18.13% of the view answers were found redundant, and 81.3% of the redundancies were found by `FastScan`. For some views the percentage of redundant view answers can be as high as 69.48%. For XMark and BIOML, on average 36.91% and 73.67% respectively, of the view answers were found to be redundant, and all of the redundancies were found by `FastScan`. We also recorded the time taken by the two scans for the three data sets. Table 2 show the total time spent on each set of queries by `FastScan`, `CompleteScan` after `FastScan`, and `CompleteScan` alone. Our experiment shows that using `FastScan` before `CompleteScan` we can find the redundancies in less than 100 milliseconds per view on average, and

[1]http://www.alphaworks.ibm.com/tech/xmlgenerator.

View	No. of view answers	No. of redundancies found by FastScan	Total no. of redundancies
file//s[//pp][//np]//vp	136710	71559	71559
file//s/vp/s[//np]/vp	11688	299	878
file//s[//vp]//np/pp/np/pp	7562	122	828
site//parlist/listitem	60481	22235	22235
site//asia/item[//name][//location]/ description//parlist/listitem	2661	985	985
site//regions/europe/item[name][shipping]//listitem	8035	2968	2968
chromosome//chromosome//[sts_domain]//gene	80262	75838	75838
chromosome//sts_domain//clone/locus/gene/dna/xxx	14042	4790	4790
chromosome//dna/xxx/clone/dna	66771	44927	44927

Table 1: Sample views and their redundancy rate for Treebank, XMark and BIOML

data set	average percentage of redundancies	maximum percentage of redundancies	aver. percentage of redundancies found by FastScan	time (ms) used by FastScan	time used by CompleteScan after FastScan	time used by CompleteScan without FastScan
TreeBank	18.13	69.48	81.3	155	1453	99375
XMark	36.91	38.21	100	15	6	25734
BIOML	73.67	99.8	100	528	137	163053

Table 2: Percentage of redundancies and time used

it only takes a tiny fraction of the time required by CompleteScan alone for all three sets of data and views.

5. QUERIES AND VIEWS IN $P^{\{/,//,*,[]\}}$

In this section, we consider answering TP queries using views when the view V and the query Q are in $P^{\{/,//,*,[]\}}$. We first present a method for finding the MCR, then we present the annotated view answer approach and show it produces the same answers as the MCR. Finally, we discuss extension of the process for removing redundant view answers to some special cases.

5.1 Finding the MCR

We make a slight modification to the definition of contained rewriting: in addition to the conditions (1) and (2) in Section 2.3, we add another condition: (3) If $label(\mathtt{sn}(\mathtt{V})) \neq *$, then $label(rt(Q')) \neq *$. This modification does not change the essence of MCR because, if Q' is a CR of Q using V and $rt(Q')$ is labeled *, then we can change the label of $rt(Q')$ to $label(\mathtt{sn}(\mathtt{V}))$ to obtain another rewriting Q'', and $Q'(V(t)) = Q''(V(t))$. The only purpose of this modification is to make our presentation easier.

We also modify the definition of useful embeddings and CATs in order to deal with the possibility that $label(\mathtt{sn}(\mathtt{V})) = *$. The modification to useful embeddings is to relax the requirement of label-preservation to *weak label-preservation* as defined below.

Definition 5.1: A mapping h from $N(Q)$ to $N(V)$ is *weak label-preserving* if (1) for every $v \in N(Q)$, either $label(v) = *$, or $label(v) = label(h(v))$, or $h(v) = \mathtt{sn}(\mathtt{V})$ and $label(\mathtt{sn}(\mathtt{V})) = *$, (2) all non-* nodes that are mapped to $\mathtt{sn}(\mathtt{V})$ have the identical label. □

Correspondingly, if $label(\mathtt{sn}(\mathtt{V})) = *$, and f is a useful embedding from Q to V which maps some node u in Q to $\mathtt{sn}(\mathtt{V})$, and $label(u) \neq *$, then the root of CAT_f will be labeled with $label(u)$.

Our main results in this subsection are summarized in Theorems 5.1 and 5.2 below.

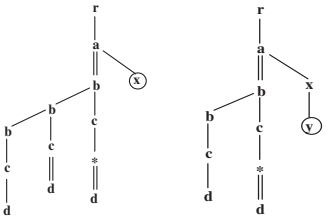
Theorem 5.1: Let V, Q be TPs in $P^{\{/,//,*,[]\}}$. If $Q \in P^{\{//,[\,],*\}} \cup \widehat{P}^{\{/,//,[\,],*\}}$ or $V \in P^{\{/,[\,],*\}}$, then $\mathsf{MCR}(Q,V) = \mathsf{CAT}_{h_1} \cup \cdots \cup \mathsf{CAT}_{h_k}$ where $h_1, \ldots, h_k$ are all of the useful embeddings from Q to V. □

We outline the proof here and leave the formal proof to the full version of this paper[2]. We need to consider two cases:

Case 1: $Q \in P^{\{//,[\,],*\}} \cup \widehat{P}^{\{/,//,[\,],*\}}$**.** We note that if $Q \in \widehat{P}^{\{/,//,[\,],*\}}$ then for any $P \in P^{\{/,//,*,[]\}}$, $P \subseteq Q$ iff there is a CM from Q to P [14]. It can be shown that the same property holds for $Q \in P^{\{//,[\,],*\}}$. Therefore, if $Q \in P^{\{//,[\,],*\}} \cup \widehat{P}^{\{/,//,[\,],*\}}$, and Q' is a CR of Q using V, then there is a CM from Q to $Q' \circ V$. This CM implies a useful embedding from Q to V, and the corresponding CAT contains Q'. Therefore, $\mathsf{MCR}(Q,V) = \mathsf{CAT}_{h_1} \cup \cdots \cup \mathsf{CAT}_{h_k}$.

Case 2: $V \in P^{\{/,[\,],*\}}$**.** Although in this case $V \subseteq Q$ iff there is a CM from Q to V, generally there may not exist a CM from Q to $Q' \circ V$ for a CR Q' of Q using V because Q' may be a pattern in $P^{\{/,//,*,[]\}}$. Thus it is not straightforward to see whether the MCR can be found by finding the useful embeddings and CATs. To prove the theorem for this

[2]http://www.cit.griffith.edu.au/∼jw/edbt11/full.pdf.



(a) V (b) Q

Figure 7: V, Q in Example 5.1

case, we need some terms and notation[3].

Let Q be a tree pattern, and u, v be any two nodes in Q. A **-chain* between u and v, denoted $u \rightsquigarrow v$, is a path from u to v that has only /-edges and only *-nodes between u and v. Note that u and v may be labeled with * or any tag in Σ. The *length* of the *-chain, denoted $|u \rightsquigarrow v|$, is the number of *-nodes between u and v (not including u and v), which is equivalent to the number of edges between u and v minus 1. Note that a /-edge from u to v is a special *-chain of length 0. The length of the longest *-chain in Q is called the **-length* of Q, denoted L^*_Q.

$$L^*_Q = max\{|u \rightsquigarrow v \mid u \rightsquigarrow v \text{ is a *-chain in } Q\}$$

For example, the *-length of the pattern Q in Figure 7 (b) is 0.

In the following, we will use $mod_m(P)$ to denote the set of tree patterns obtained from P by replacing the i-th //-edge (u, v) with a *-chain between u and v of length l_i, where $0 \leq l_i \leq m$. Then we can prove the following lemma.

Lemma 5.1: *Let P, $P_1, \ldots, P_n$ be tree patterns in $P^{\{/,//,*,[]\}}$. Let $w = max\{L^*_{P_1}, \ldots, L^*_{P_n}\}$. If every pattern in $mod_{w+1}(P)$ is contained in $P_1 \cup \cdots \cup P_n$, then $P \subseteq P_1 \cup \cdots \cup P_n$.* □

Using the above lemma, we can prove the next lemma which implies that Theorem 5.1 holds in the case $V \in P^{\{/,[\,],*\}}$.

Lemma 5.2: *If Q is in $P^{\{/,//,*,[]\}}$ and V is in $P^{\{/,[\,],*\}}$, then for every CR Q' of Q using V, there are useful embeddings $h_1, \ldots, h_k$ from Q to V such that $Q' \subseteq \mathsf{CAT}_{h_1} \cup \cdots \cup \mathsf{CAT}_{h_k}$.* □

PROOF SKETCH. Let $w = L^*_Q$, and $mod_{w+1}(Q')$ be the set $\{Q_1, \ldots, Q_n\}$. We will show that for every $Q_i \in mod_{w+1}(Q')$, there is a useful embedding h_j from Q to V such that $Q_i \subseteq \mathsf{CAT}_{h_j}$, thus every Q_i is contained in $\mathsf{CAT}_{h_1} \cup \cdots \cup \mathsf{CAT}_{h_k}$. Clearly the *-length of CAT_{h_i} is no more than w. Hence by Lemma 5.1, $Q' \subseteq \mathsf{CAT}_{h_1} \cup \cdots \cup \mathsf{CAT}_{h_k}$.

The useful embedding h_j can be found as follows. Since $Q' \circ V \subseteq Q$, and $Q_i \subseteq Q'$, we know $Q_i \circ V \subseteq Q$. Since $Q_i \circ V$ does not have //-edges, we know there is a CM δ_i from Q to $Q_i \circ V$. This CM, if restricted to nodes of Q whose image under δ_i is in the V-part of $Q_i \circ V$, is a useful embedding from Q to V. Denote this useful embedding by h_j. It is not hard to verify $Q_i \subseteq \mathsf{CAT}_{h_j}$. □

[3]Some of the terms are borrowed from [10].

This completes the proof outline of Theorem 5.1.

Before describing the next result we need more notations.

Let $x, y \in \Sigma \cup \{*\}$ be two labels. Given TP $Q \in P^{\{/,//,*,[]\}}$, we define $Q(x, y)$ to be the set of *-chains from an x-node or *-node to a y-node or $*$-node:

$$Q(x, y) = \{u \rightsquigarrow v \in Q \mid label(u) \in \{x, *\}, label(v) \in \{y, *\}\}$$

Define the function $L_Q(x, y)$ as follows:

$$L_Q(x, y) = \begin{cases} -1 & \text{if } Q(x, y) = \emptyset; \\ max\{|u \rightsquigarrow v| \mid u \rightsquigarrow v \in Q(x, y)\} & \text{otherwise.} \end{cases}$$

For example, for the TP Q in Figure 7, $L_Q(a, b) = -1$, $L_Q(c, d) = 0$, and $L_Q(*, d) = -1$.

Let V, Q be TPs in $P^{\{/,//,*,[]\}}$. We will use $\mathtt{Mod}_Q(V)$ to denote the set of TPs obtained from V by replacing each //-edge $e = (u, v)$ with a *-chain $u \rightsquigarrow v$ of length l_e, where l_e is an integer in $[0, L_Q(label(u), label(v)) + 1]$.

We can now describe our next main result as follows.

Theorem 5.2: Let V, Q be TPs in $P^{\{/,//,*,[]\}}$. Suppose $\mathtt{Mod}_Q(V) = \{V_1, \ldots, V_n\}$. Then

(1) Q' is a CR of Q using V iff it is a CR of Q using V_i for all $i = 1, \ldots, n$.

(2) $\mathsf{MCR}(Q, V) = \mathsf{MCR}(Q, V_1) \cap \cdots \cap \mathsf{MCR}(Q, V_n)$. □

We discuss how to use the above theorem to find $\mathsf{MCR}(Q, V)$ before outlining its proof.

Theorem 5.2 (2) implies a method for finding $\mathsf{MCR}(Q, V)$ for the general case, which consists of the following steps:

1. Find the set $\mathtt{Mod}_Q(V)$ of patterns (call them *sub-views* of V);
2. Find the MCR of Q using each of the subviews in $\mathtt{Mod}_Q(V)$.
3. Intersect these MCRs to obtain $\mathsf{MCR}(Q, V)$.

Example 5.1: Consider the view V, query Q shown in Figure 7, where V and Q are modified from the examples in [10]. It is easy to verify that x/y is a CR of Q using V, but this rewriting cannot be found by finding all useful embeddings from Q to V. However, the rewriting can be found by first splitting the view into 2 subviews (the //-edge (c, d) may be replaced with a *-chain of length 0 or 1, and the other //-edges can be replaced with a /-edge) with /-edges only, and then finding the CR x/y of Q using each of the subviews, and finally intersecting the two identical CRs. □

Complexity The above method for finding the MCR of Q using V requires us to find the MCRs of Q using each subview in $\mathtt{Mod}_Q(V)$. We do not need to actually do the intersection of the MCRs, i.e., to transform $\mathsf{MCR}(Q, V_1) \cap \cdots \cap$

$\mathsf{MCR}(Q, V_n)$ into a union of tree patterns. Instead, we can take the intersection $\mathsf{MCR}(Q, V_1)(V(t)) \cap \cdots \cap \mathsf{MCR}(Q, V_n)(V(t))$ when evaluating it over $V(t)$. The cost for finding each of $\mathsf{MCR}(Q, V_i)$ has been discussed in [8]. The cost of finding $\mathsf{MCR}(Q, V)$ would be the total cost of finding all of $\mathsf{MCR}(Q, V_1), \ldots, \mathsf{MCR}(Q, V_n)$ (if we ignore the cost to find $\mathtt{Mod}_Q(V)$ which is negligible).

Since the subviews in $\mathtt{Mod}_Q(V)$ have similar structures, it is highly likely that we can design a more efficient algorithm to find all of $\mathsf{MCR}(Q, V_1), \ldots, \mathsf{MCR}(Q, V_n)$ altogether using the ideas of *match sets* [10]. We leave it as future work.

Next we outline the proof of Theorem 5.2. The proof uses Lemma 5.3 below and can be done as follows:

Proof of Theorem 5.2: (2) is a direct corollary of (1), so we focus on the proof of (1).

Let Q' be a CR of Q using V, that is, $Q' \circ V \subseteq Q$. Since $V_i \subseteq V$, we know $Q' \circ V_i \subseteq Q$ (for $i \in [1, n]$). Hence Q' is a CR of Q using V_i.

Conversely, suppose Q' is a CR of Q using V_i (for all $i = 1, \ldots, n$), then $Q' \circ V_i \subseteq Q$. Suppose Q_j is a pattern in $\mathtt{Mod}_Q(Q')$. Then $Q_j \subseteq Q'$, thus $Q_j \circ V_i \subseteq Q$. It is easy to verify that $\mathtt{Mod}_Q(Q' \circ V) = \{Q_j \circ V_i \mid Q_j \in \mathtt{Mod}_Q(Q'), V_i \in \mathtt{Mod}_Q(V)\}$. Therefore, by Lemma 5.3, $Q' \circ V \subseteq Q$. That is, Q' is a CR of Q using V.

Lemma 5.3: *Let P, Q be TPs in $P^{\{/,//,*,[]\}}$. If every pattern in $\mathtt{Mod}_Q(P)$ is contained in Q, then $P \subseteq Q$.* □

To prove Lemma 5.3, we only need to prove Lemma 5.4 below. This is because that the containment of tree patterns in $P^{\{/,//,*,[]\}}$ can be reduced to the containment of *Boolean patterns* [10], which are tree patterns that do not have selection nodes, and which will return either TRUE or FALSE when evaluated over a data tree t, depending on whether there is a matching of the pattern in t. For Boolean patterns P and Q, $P \subseteq Q$ means $\forall t, P(t) \Rightarrow Q(t)$. We use $mod(P)$ to denote the set of all data trees in which P has a matching, that is, $mod(P) = \{t \mid P(t) = \text{TRUE}\}$.

Lemma 5.4: *Let P, Q be Boolean patterns. Let z be a label that is not present in Q. Let $\mathtt{Mod}_Q^z(P)$ be the set of data trees obtained from P by first replacing each //-edge $e = (u, v)$ with a *-chain $u \rightsquigarrow v$ of length l_e where $0 \le l_e \le L_Q(label(u), label(v)) + 1$, and then replacing $*$ with z. If $\mathtt{Mod}_Q^z(P) \subseteq mod(Q)$, then $P \subseteq Q$.* □

To prove Lemma 5.4, we need Lemma 5.5 and Lemma 5.6 below.

Lemma 5.5: *[10] Let e be a matching of P in t. There exists a unique pattern P^0 obtained from P by replacing each //-edge with a *-chain such that, there is a unique matching e' of P^0 in t such that $\forall v \in N(P)$, $e(v) = e'(v)$.* □

Lemma 5.6: *Let t_1 be a data tree and P' be a Boolean pattern such that $P'(t_1) =$FALSE. Let $x_0, x_1, \ldots, x_n, x_{n+1}$ be a path in t_1 such that $n > L_{P'}(label(x_0), label(x_{n+1})) + 1$ and x_{i+1} is the unique child of x_i for $i = 1, \ldots, n-1$, and none of the labels $label(x_1), \ldots, label(x_n)$ has appeared in P'. Let t_2 be the tree obtained from t_1 by deleting the node x_n and transforming all its children into children of x_{n-1}. Then $P'(t_2) =$FALSE.* □

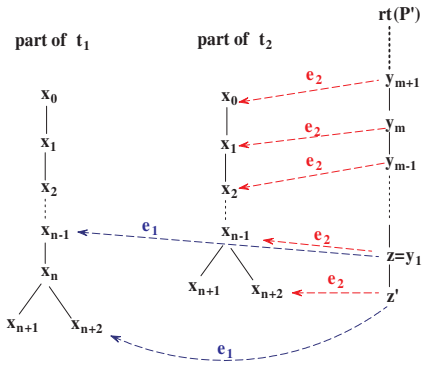


Figure 8: Illustration of the proof of Lemma 5.6

PROOF. The proof is a modification to the proof of Lemma 4 in [10]. Let t_1, P', t_2 be as in the lemma. Assume that $P'(t_2) =$TRUE, and let e_2 be a matching of P' in t_2. We show there is a matching of P' in t_1, contradicting $P'(t_1) =$FALSE. Let $x = label(x_0), y = label(x_{n+1})$. First we consider the special case where $L_{P'}(x, y) = -1$. In this case, there are no /-edges from any x-node to any y-node, or from any *-node to any y-node, or from any x-node to any *-node, or from one *-node to another. Therefore, for any /-edge (u, v) in P', e_2 cannot map u to x_0 and v to x_1. If $n = 1$ $(= L_{P'}(x, y) + 2)$, then in t_2 the path $x_0, x_1, \ldots, x_{n-1}, x_{n+1}$ is just a /-edge (x_0, x_2). Since no /-edge can be mapped to (x_0, x_2), e_2 is also a matching of P' in t_1. Now suppose $n > 1$. Since the labels of $x_1, x_2, \ldots, x_n$ are not present in P', only *-nodes in P' can be mapped to the nodes $x_1, \ldots, x_n$. If no nodes in P' have been mapped into $x_1, \ldots, x_{n-1}$ by e_2, then clearly e_2 is also a matching of P' in t_1; if there are *-nodes $u_1, \ldots, u_k$ that are mapped into $x_1, \ldots, x_{n-1}$ by e_2, then they must be connected with each other via //-edges, and u_1 is connected to its parent via a //-edge, u_k is connected to its children (if any) via //-edges. Therefore, e_2 is also a matching of P' in t_1. This completes the proof for the special case $L_{P'}(x, y) = -1$. Next we consider the case where $L_{P'}(x, y) \ge 0$. Denote C the set $\{x_1, \ldots, x_n\}$, and define the following two sets

$$S = \{(z_0, z_1) \mid (z_0, z_1) \text{ is a //-edge in } P',\ e_2(z_1) \in C\}$$

$$C' = \{z \mid e_2(z) \in C \wedge \exists (z_0, z_1) \in S \text{ such that } z = z_1 \text{ or } z \text{ is a descendant of } z_1\}$$

Define the following mapping e_1 from $N(P')$ to $N(t_1)$:

$$e_1(z) = x_{i+1} \quad \text{if } z \in C' \text{ and } e_2(z) = x_i$$
$$e_1(z) = e_2(z) \quad \text{if } z \notin C'$$

We show e_1 is a matching of P' in t_1. It is easy to see e_1 is root-preserving and label-preserving. Next we show it is

structure-preserving. For any edge (z, z') in P', the number of edges between $e_1(z)$ and $e_1(z')$, denoted $d(e_1(z), e_1(z'))$, is either $d(e_2(z), e_2(z'))$ or $d(e_2(z), e_2(z')) + 1$. This is because $e_1(z')$ is either identical to $e_2(z')$ or one node below $e_2(z')$, while if $e_1(z)$ is one node below $e_2(z)$, then either $e_1(z')$ is one node below $e_2(z')$ as well or $e_2(z) = x_{n-1}$, $e_1(z) = x_n$ and $e_1(z') = e_2(z')$. Therefore, $e_1(z')$ is always a descendant of $e_1(z)$. We now only need to show if (z, z') is a /-edge, then $e_1(z')$ is a child of $e_1(z)$. The only possible case where $e_1(z')$ is not a child of $e_1(z)$ is when $e_1(z) = e_2(z) = x_{n-1}$ and $e_1(z') = e_2(z')$ is a node below x_n (Refer to Figure 8). This happens only if z' is not in C', that is, z' is not below a //-edge whose end node is mapped to C. Consider the path in P' from z to $rt(P')$: $y_1 = z, y_2, \ldots, y_k = rt(P')$. Let m be the largest number such that $e_2(y_m) \in C$, hence $e_2(y_{m+1}) \notin C$ (there exists such m since $e_2(rt(P')) \notin C$). Consider the path $y_m, y_{m-1}, \ldots, y_1$ in P'. All the nodes in it are mapped to C hence they are all labeled *. Furthermore, all edges (y_{i+1}, y_i) (for $i \in [1, m]$) are /-edges, since otherwise $z = y_1$ would be in C', while we already know $z \notin C'$. So the path from y_{m+1} to z' is a *-chain. Hence its length m is less or equal to $L_{P'}(label(y_{m+1}), label(z'))$. On the other hand, since all edges are /-edges, the images of $y_m, \ldots, y_1$ under e_2 must include all nodes $x_1, \ldots, x_{n-1}$, thus $m = n - 1$. This implies $n - 1 \leq L_{P'}(label(y_{m+1}), label(z'))$, which is a contradiction to the assumption. □

Proof of lemma 5.4 We are now ready to prove Lemma 5.4. The proof is similar to that of Proposition 3 of [10]. It is a proof by contradiction. The idea is to show that if $P \not\subseteq Q$, then there is $t_1 \in mod^z_m(P)$ (note: $mod^z_m(P)$ denotes the set of data trees obtained from the patterns in $mod_m(P)$ by replacing the wildcard * with z) for some sufficiently large integer m, such that $Q(t_1) =$FALSE (Step 1). From t_1, we can construct $t' \in \mathtt{Mod}^z_Q(P)$ such that $Q(t') =$FALSE, contradicting the assumption $\mathtt{Mod}^z_Q(P) \subseteq mod(Q)$ (Step 2).

Step 1 of the proof is similar to that of Proposition 3 of [10]. Step 2 can be completed using Lemma 5.6 as follows. If $t_1 \in \mathtt{Mod}^z_Q(P)$ then the proof is complete. Suppose t_1 is not in $\mathtt{Mod}^z_Q(P)$. That is, there is //-edge (u, v) in P such that in t_1 the number of nodes between u and v is larger than $L_Q(x, y) + 1$, where $x = label(u), y = label(v)$. Then we can apply Lemma 5.6 repeatedly to reduce the length of the path until the number of nodes between u and v is less or equal to $L_Q(x, y) + 1$. This process can be repeated for every //-edge in P until we find $t' \in \mathtt{Mod}^z_Q(P)$.

5.2 The alternative approach

We show how to answer Q using V without explicitly finding the MCR in this section.

Let $Q, V \in P^{\{/,//,*,[]\}}$ be TPs. Let t' be *any* data tree such that $label(rt(t')) = label(\mathtt{sn}(\mathtt{V}))$ or $label(\mathtt{sn}(\mathtt{V})) = *$. We use $t' \diamond V$ to denote the tree obtained from t' and V by merging $rt(t')$ and $\mathtt{sn}(\mathtt{V})$, where the merged node is labeled with $label(rt(t'))$. Let $V' \in \mathtt{Mod}_Q(V)$ be a subview, t be a data tree, and t_i be a tree in $V(t)$. We call $t_i \diamond V'$ a *view answer annotated by* V' (note that t_i may be outside $V'(t)$). Similar to Theorem 3.1, we can prove the following lemma.

Lemma 5.7: *Let Q and V be TPs in $P^{\{/,//,*,[]\}}$. Let $h_1, \ldots, h_k$ be all of the useful embeddings from Q to V. Then $Q(t' \diamond V) = \mathsf{CAT}_{h_1}(t') \cup \cdots \cup \mathsf{CAT}_{h_k}(t')$.* □

Using the lemma above, we can prove

Theorem 5.3: For any data tree t, suppose $V(t) = \{t_1, \ldots, t_m\}$, and $\mathtt{Mod}_Q(V) = \{V_1, \ldots, V_n\}$. Then

$$\mathsf{MCR}(Q, V)(V(t)) = \bigcap_{i=1}^{n} (Q(t_1 \diamond V_i) \cup \cdots \cup Q(t_m \diamond V_i))$$

□

PROOF SKETCH. By Theorem 5.2 (2),

$\mathsf{MCR}(Q, V)(V(t)) = \bigcap_{i=1}^{n} \mathsf{MCR}(Q, V_i)(V(t))$

$= \bigcap_{i=1}^{n} (\mathsf{MCR}(Q, V_i)(t_1) \cup \cdots \cup \mathsf{MCR}(Q, V_i)(t_m))$

By Theorem 5.1 and Lemma 5.7, $\mathsf{MCR}(Q, V_i)(t_j) = Q(t_j \diamond V_i)$ for $j \in [1, m], i \in [1, n]$. Therefore, $\mathsf{MCR}(Q, V)(V(t)) = \bigcap_{i=1}^{n} (Q(t_1 \diamond V_i) \cup \cdots \cup Q(t_m \diamond V_i))$. □

The above theorem indicates that we can answer Q without explicitly finding $\mathsf{MCR}(Q, V)$. Instead, we can evaluate Q over $t_j \diamond V_i$ ($i \in [1, n], j \in [1, m]$), and intersect $Q(t_1 \diamond V_i) \cup \cdots \cup Q(t_m \diamond V_i)$ for all subviews V_i in $\mathtt{Mod}_Q(V)$.

5.3 Redundant view answers wrt $\mathcal{P} = P^{\{//,[\,],*\}} \cup \widehat{P}^{\{/,//,[\,],*\}}$

It is easy to verify that all propositions, theorems and corollaries in Sections 3 and 4 still hold if V and Q are changed to any patterns in $P^{\{/,//,*,[]\}}$. Therefore, because of Theorem 5.1, in the special cases $Q \in P^{\{//,[\,],*\}} \cup \widehat{P}^{\{/,//,[\,],*\}}$ or $V \in P^{\{/,[\,],*\}}$, for all data tree t, $\mathsf{MCR}(Q, V)(V(t)) = \bigcup_{t' \in V(t)} Q(t' \cdot V)$. That is, evaluating the query Q directly over the annotated view answers will generate the same answers as evaluating the MCR over the original answers. Furthermore, we can use the same criteria and procedures as described in Section 4 to identify/remove redundant view answers with respect to $\mathcal{P} = P^{\{//,[\,],*\}} \cup \widehat{P}^{\{/,//,[\,],*\}}$, for any view $V \in P^{\{/,//,*,[]\}}$.

6. RELATED WORK

In this section we compare our work with previous ones that are the mostly closely related to ours. As mentioned in Section 1, apparently [8] is the first paper on MCR of tree pattern queries using views, and it proposed the technique of useful embeddings for queries and views in $P^{\{/,//,[]\}}$. The paper also proposed an algorithm to find the MCR under a non-recursive and non-disjunctive DTD which can be represented as an acyclic graph. The basic idea is to reduce the original problem to one without DTD by chasing the tree patterns repeatedly using constraints that can be derived from the DTD. Recently, [11] proposed a method for finding the MCR for queries that have no *-nodes connected to a //-edge and no leaf node u such that u and the parent of u are both labeled *, based on the concepts of *trap embedding* (and its *induced pattern*) and *trap relay*, where a trap embedding is a mapping from a tree pattern to a tree, and a trap relay is a mapping from a pattern to another pattern.

It can be shown that the induced pattern of a trap embedding from Q to a pattern V_i in $mod_{L^*_Q+1}(V)$ is the same as a CAT if the *attach point* (see [11]) of V_i is $\texttt{sn}(\texttt{V}_\texttt{i})$. Thus Theorem 3.3 of [11] can be derived from Theorem 5.2 (1) of this paper (but not the other way around). Previously, [13] studied maximal contained rewriting using multiple views for queries and views in $P^{\{/,//,[]\}}$. More recently, [7] and [16] studied the evaluation of MCRs, and [12] gave an algorithm for identifying redundant contained rewritings, which is orthogonal to Section 4 of this work. It should be noted that the redundant view answers discussed in this paper are with respect to the union of all CATs, and they may not be redundant with respect to equivalent rewritings or with respect to a subset of CATs.

There have been works on *equivalent rewritings (ERs)*, where an equivalent rewriting is a special contained rewriting Q' which satisfies the condition $Q' \circ V = Q$. Among them [15] showed that if V and Q are in $P^{\{/,//,[]\}}$, $P^{\{/,[\,],*\}}$ or they are normalized linear patterns, there is an equivalent rewriting of Q using V iff Q_k is an equivalent rewriting, where k is the position of $\texttt{sn}(\texttt{V})$ on the selection path of V, and Q_k is the subtree of Q rooted at the k-th node on the selection path of Q. [4] extended the above result to $Q, V \in P^{\{/,//,*,[]\}}$ and showed that for many common special cases, there is an equivalent rewriting of Q using V iff either Q_k or $Q_{k//}$ is an equivalent rewriting, where $Q_{k//}$ is the pattern obtained from Q_k by changing all edges connected to the root to //-edges. Thus in those special cases, to find an equivalent rewriting we only need to test whether Q_k or $Q_{k//}$ is an equivalent rewriting. [9] assumed both Q and V are minimized, and reduced tree pattern matching to string matching, allowing a more efficient algorithm for finding equivalent rewritings. They further proposed a way to organize the materialized views in Cache to enable efficient view selection and cache look-up. [6] investigated equivalent rewritings using a single view or multiple views for queries and views in $P^{\{/,//,[]\}}$, where the set of views is represented by grammar-like rules called *query set specifications*. The work can be seen as an extension to an earlier work [5] which investigated the same problem for explicit views.

None of the previous works dealt with the redundant view answers problem or the approach of finding answers to a new TP using annotated view answers. Finally, Section 5 of this work borrows techniques from [10], and some of our results on query containment, namely Lemmas 5.1 and 5.3, are extensions to the results in [10].

7. CONCLUSION

We revisited the problem of answering TP queries using views, and showed that for Q in $P^{\{/,//,[]\}}$ evaluating Q directly over the annotated view answers will find the same answers as evaluating the MCR of Q using V over the original view answers. We also provided an algorithm for identifying redundant view answers in this case. Our preliminary experiments with TreeBank and XMark showed that there can be a large percentage of redundant view answers and the majority of the them can be found using the linear algorithm `FastScan`. We showed that the new approach and the techniques for finding redundant view answers can also be applied if we limit Q to $\widehat{P}^{\{/,//,[\,],*\}}$ or $P^{\{//,[\,],*\}}$. We showed that if V has no //-edges then for any $Q \in P^{\{/,//,*,[]\}}$ the approach of useful embeddings can be used to find $\mathsf{MCR}(Q, V)$. For V and Q which are general TPs in $P^{\{/,//,*,[]\}}$, we presented a method for finding $\mathsf{MCR}(Q, V)$ by dividing V into a finite set of subviews and finding the MCRs of Q using each of the subviews. We discussed how to answer Q using V without explicit rewriting in the general case.

As future work we plan to do experiments to compare the actual performance of the rewriting approach and the annotated view answer approach, design more efficient algorithms for finding the MCR and investigate the redundant view answer problem for the general case where $Q, V \in P^{\{/,//,*,[]\}}$.

Acknowledgement: This work is supported by the Australian Research Council Discovery Grant DP1093404 and by grant of the Research Grants Council of the Hong Kong SAR, China No. 419008. We thank Professor Wenfei Fan for helpful comments on an earlier draft of this paper.

8. REFERENCES

[1] *http://www.xml-benchmark.org/.*
[2] *http://www.cs.washington.edu/research/xmldatasets/.*
[3] *http://xml.coverpages.org/BIOML-XML-DTD.txt.*
[4] F. N. Afrati, R. Chirkova, M. Gergatsoulis, B. Kimelfeld, V. Pavlaki, and Y. Sagiv. On rewriting XPath queries using views. In *EDBT*, 2009.
[5] B. Cautis, A. Deutsch, and N. Onose. XPath Rewriting using multiple views: Achieving completeness and efficiency. In *WebDB*, 2008.
[6] B. Cautis, A. Deutsch, N. Onose, and V. Vassalos. Efficient rewriting of XPath queries using query set specifications. *PVLDB*, 2(1):301–312, 2009.
[7] J. Gao, J. Lu, T. Wang, and D. Yang. Efficient evaluation of query rewriting plan over mterialized XML view. *The Journal of Systems and Software*, 83:1029–1038, 2010.
[8] L. V. S. Lakshmanan, H. Wang, and Z. J. Zhao. Answering tree pattern queries using views. In *VLDB*, 2006.
[9] B. Mandhani and D. Suciu. Query caching and view selection for XML databases. In *VLDB*, 2005.
[10] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1), 2004.
[11] J. Tang and A. W.-C. Fu. Query rewritings using views for XPath queries, framework, and methodologies. *Inf. Syst.*, 35(3):315–334, 2010.
[12] J. Wang, K. Wang, and J. Li. Finding irredundant contained rewritings of tree pattern queries using views. In *APWeb/WAIM*, pages 113–125, 2009.
[13] J. Wang and J. X. Yu. XPath rewriting using multiple views. In *DEXA*, pages 493–507, 2008.
[14] J. Wang, J. X. Yu, and C. Liu. Independence of containing patterns property and its application in tree pattern query rewriting using views. *World Wide Web*, 12(1):87–105, 2009.
[15] W. Xu and Z. M. Özsoyoglu. Rewriting XPath queries using materialized views. In *VLDB*, 2005.
[16] R. Zhou, C. Liu, J. Li, J. Wang, and J. Liu. Evaluating contained rewritings of XPath queries over materialized views. In *DASFAA*, 2011, to appear.