A New Algorithm for Minimizing Tree Pattern Queries

Yangjun Chen
University of Winnipeg, Canada

INTRODUCTION

XML employs a tree-structured model for representing data. Queries in XML query languages, for example, XPath (World Wide Web Consortium, 1999), XQuery (World Wide Web Consortium, 2001), XML-QL (Deutch, Fernandex, Florescu, Levy, & Suciu, 1999), and Quilt (Chamberlin, Clark, Florescu, & Stefanescu 1999; Chamberlin, Robie, & Florescu, 2000), typically specify patterns of selection predicates on multiple elements that have some specified tree structured relationships. For instance, the following XPath expression:

$a[b[c \text{ and } //d]]/b[c \text{ and } e//d]$

asks for any node of type b that is a child of some node of type a. In addition, the b-node is the parent of some c-node and some e-node, as well as an ancestor of some d-node. In general, such an expression can be represented by a tree structure as shown in Figure 1(a).

In such a tree pattern, the nodes are types from $\sum \cup \{*\}$ (* is a wildcard, matching any node type), and edges are *parent-child* or *ancestor-descendant* relationships. Among all the nodes of a query Q, one is designated as the output node, denoted by *output*(Q), corresponding to the output of the query.

In the following discussion, we use $\tau(v)$ to denote the type of node v. A parent-child edge is referred to as a c-edge and a c-edge from node v to node u is denoted by $v \rightarrow u$ in the text. Also, u is called a *c-child* of v. An ancestor-descendant edge is referred to as a d-edge and a d-edge is denoted by $v \Rightarrow u$ in the text. u is called a *d-child* of v. The output node is indicated by "-" in the figures.

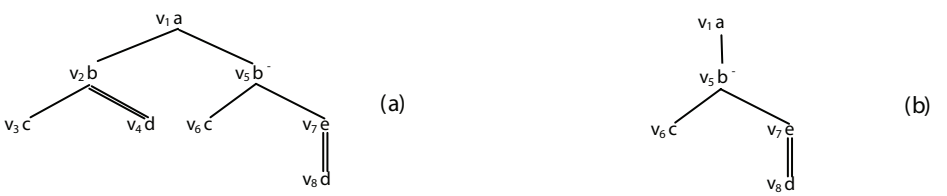
In any DAG (*directed acyclic graph*), a node u is said to be a descendant of a node v if there exists a path (sequence of edges) from v to u. In the case of a TPQ, this path could consist of any sequence of c-edges and/or d-edges.

In terms of Ramanen (1999), an embedding of a tree pattern query (TPQ) Q into an XML document T is a mapping f: $Q \rightarrow T$, from the nodes of Q to the nodes of T, which satisfies the following conditions:

i. Preserve node type: For each $v \in Q$, v and $f(v)$ are of the same type.
ii. Preserve c/d-child relationships: If $v \rightarrow u$ in Q, then $f(u)$ is a child of $f(v)$ in T; if $v \Rightarrow u$ in Q, then $f(u)$ is a descendant of $f(v)$ in T.

Any document T, in which Q can be embedded, is said to contain Q and considered to be an answer. An embedding of Q in T with $root(Q) = root(T)$ is called a *root preserving embedding*. According to the above definition, more than one node (of the same type) in Q could be mapped to the same node in T. In general, the efficiency of finding the result of a query on a given input database depends on the size of the query. Therefore, it is important to minimize the query before attempting to compute the result of the query. In this article, we propose a new algorithm for this task, which

Figure 1. A query tree



needs $O(|Q|^2)$ time and $O(|Q|\cdot leaf_Q)$ space, where $leaf_Q$ represents the number of the leaf nodes of Q.

BACKGROUND

In this section, we define the minimization of TPQs and review the related work.

Minimization of TPQs

As an example of TPQ minimization, consider the query shown in Figure 1(a) once again. In this TPQ, the subtree rooted at v_2 is made redundant by the subtree rooted at v_5. Therefore, the TPQ is equivalent to the one shown in Figure 1(b), which is also minimal in the absence of integrity constraints (ICs). In Figure 1(a), if v_2 is the output node (instead of v_5), the TPQ is considered to be minimal in the absence of ICs since if we reduce this TPQ as above, we will not have an output node in the reduced version.

In addition, as pointed out by Amer-Yahia, Cho, Laksmanan, and Srivastava (2001), some ICs may exist in an input database, which may make some branches of a TPQ Q redundant. Therefore, a further reduction is possible. In Amer-Yahia et al. (2001), three kinds of ICs are considered:

1. Required child: Every database node of type τ has a child of type τ', denoted by $\tau \rightarrow \tau$'.
2. Required descendant: Every database node of type τ has a descendant of type τ', denoted by $\tau \Rightarrow \tau$'.
3. Suptype: Every database node of τ is also of type τ', denoted by $\tau \leq \tau$'. For example, we always have "graduate student" $\leq$ "student." Trivially, $\tau \leq \tau$ for all types τ.

To see how to reduce a TPQ by using ICs, let us have a look at Figure 1(b). If $b \rightarrow c$ is known to hold in the input database, the tree shown in this figure can be reduced to the tree shown in Figure 2(a). If $e \Rightarrow d$ holds besides $b \rightarrow c$, it can further be reduced to the tree shown in Figure 2(b).

In some cases, in the presence of subtype constraints, a query tree may also be reduced. For example, in the presence of $b' \leq b$ and $c' \leq c$, the tree shown in Figure 2(c) can be reduced to the tree shown in Figure 2(d).

Related Work

The query reduction shown above is closely related to the conjunctive query minimization: a problem that is in general NP-complete for classical relational databases. In the case of TPQs, the problem is treatable in polynomial time in some cases. In Amer-Yahia et al. (2001), the authors pointed out that the tree pattern query is essentially a special kind of conjunctive queries on a tree-structured domain, and presented an $O(n^4)$ algorithm for minimizing TPQs in the absence of integrity constrains. In Florescu, Levy, and Suciu (1998), they showed that containment of conjunctive queries with regular path expressions over semistructured data is decidable (Garey & Johnson, 1979). Techniques like predicate elimination and join minimization are used. However, such kinds of optimization are based on algebraic rewritings, which often generate exponential search spaces and results in problems that cannot be solved in polynomial time.

The minimization of XPath queries in a tree structure database is a harder problem than TPQs. It was first studied in Flesca, Furfaro, and Masciari (2003). In that paper, Flesca et al. address the problem of minimizing XPath queries for limited fragments of XPath, containing only the child, the descendent, the branch, and the wildcard operators.* In their work, they proved

Figure 2. Query tree reduction