Select-Project Queries over XML Documents*

Sara Cohen, Yaron Kanza, and Yehoshua Sagiv

Dept. of Computer Science,
The Hebrew University,
Jerusalem 91904, Israel
{sarina, yarok, sagiv}@cs.huji.ac.il

Abstract. This paper discusses evaluation of select-project (SP) queries over an XML document. A SP query consists of two parts: (1) a conjunction of conditions on values of labels (called the *selection*) and (2) a series of labels whose values should be outputed (called the *projection*). Query evaluation involves finding tuples of nodes that have the labels mentioned in the query and are related to one another other in a meaningful fashion. Several different semantics for query evaluation are given in this paper. Some of these semantics also take into account the possible presence of incomplete information. The complexity of query evaluation is analyzed and evaluation algorithms are described.

1 Introduction

Increasingly large amounts of data are accessible to the general public in the form of XML documents. It is difficult for the naive user to query XML and thus, potentially useful information may not reach its audience. Search engines are currently the only efficient way to query the Web. These engines do not exploit the structure of documents and hence, are not well suited for querying XML.

As a long-term goal, we would like to allow a natural-language interface for querying XML. It has been noted that the universal relation [9,12,13] is a first step towards facilitating natural-language querying of relational databases. This is because of the inherent simplicity of formulating a query against the universal relation. Such queries usually consist of only selection and projection and are called *select-project* or *SP* queries. Evaluating queries over the universal relation was studied in [11,7].

Many languages, such as XQuery [3] and XML-QL [6] have been proposed for querying XML. However, these languages are not suitable for a naive user. They also require a rather extensive knowledge of document structure in order to formulate a query correctly. The language EquiX [4] has been proposed for querying XML by a naive user. However, EquiX queries can only be formulated against a document with a DTD. A query language for XML must also take into consideration incomplete information. This has been studied in [2,8].

* Supported by Grant 96/01-1 from the Israel Science Foundation

A. Halevy and A. Gal (Eds.): NGITS 2002, LNCS 2382, pp. 2–13, 2002.

In this paper we explore the problem of answering an SP query formulated against an XML document. In order to formulate a query, users only need to know the names of the tags appearing in the document being queried. Queries consist of two parts:

- **Select:** boolean conditions on tags of a document (e.g., title = 'Cat in the Hat');
- **Project:** names of tags whose values should appear in the result (e.g., price).

Answering an SP query requires finding elements in a document that are *related* to one another in a *meaningful fashion*. Intuitively, such sets of elements correspond to rows in a universal relation that could be defined over an XML document. However, there are several questions that arise in this context:

- How can we decide when elements are related in a meaningful fashion? This becomes especially difficult when one considers the fact that documents may have varied structure.
- How can we deal with incompleteness in documents? If a document may be missing information, then we may have to discover whether a particular element is meaningfully related to an element that does not even appear in the document.

This paper deals with these questions.

Section 2 presents some necessary definitions and Section 3 present query semantics. In Sections 4 and 5 we discuss the complexity of answering SP queries over XML documents and present algorithms for query evaluation. Section 6 concludes.

2 Definitions

In this section we present some necessary definitions. We specify our data model and describe the syntax of *select-project* or *SP* queries.

Trees. We assume that there is a set $\mathcal{L}$ of labels and a set $\mathcal{A}$ of constants. An XML document is a tree T in which each *interior node* is associated with a *label* from $\mathcal{L}$ and each *leaf node* is associated with *value* from $\mathcal{A}$. We denote the label of an interior node n by $lbl(n)$ and the value of a leaf node n' by $val(n')$. We extend the *val* function to interior nodes n by defining $val(n)$ to be the concatenation of the values of its leaf descendents. In Figure 1 there is an example of such a tree, describing information about books. The nodes are numbered to allow easy reference.

Let T be a tree and let $n_1, \ldots, n_k$ be nodes in T. We denote by $lca\{n_1, \ldots, n_k\}$ the *lowest common ancestor* of $n_1, \ldots, n_k$. Let T_{lca} be the subtree of T rooted at $lca\{n_1, \ldots, n_k\}$. We denote by $T_{n_1,\ldots,n_k}$ the tree obtained by pruning from T_{lca} all nodes that are not ancestors of any of the nodes $n_1, \ldots, n_k$. We call this tree the *relationship tree* of $n_1, \ldots, n_k$. For example, in Figure 1, $lca\{15, 19, 21\}$ is 13. The relationship tree of 15, 19, 21 contains the nodes 13, 14, 15, 19 and 21.

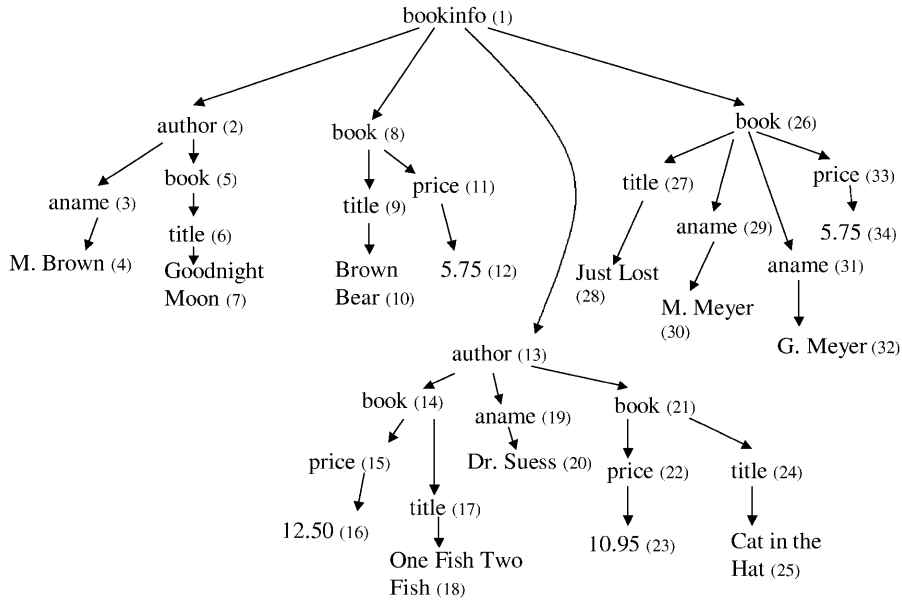

Fig. 1. An XML document describing books for sale

Relations. A *tuple* has the form $t = \{l_1 : a_1, \ldots, l_k : a_k\}$ where l_i and a_i are a *column name* and a *value*, respectively. We will use $l_i(t)$ to denote the value a_i. We call $\{l_1, \ldots, l_k\}$ the *signature* of t. A *relation* R is a set of tuples with the same signature, also called the signature of R.

Let N be a set of nodes in which no two nodes have the same label. Let L be the set of labels of nodes in N. The set N naturally gives rise to a tuple denoted t_N with signature L. Formally, if $n \in N$ and $lbl(n) = l$, then $l(t_N) = n$. Given a set of labels L' that contains L, the set N gives rise to a tuple with signature L', denoted $t_{L',N}$ by padding t_N with null values (denoted $\perp$), as necessary.

Select-Project Queries. A *condition* has the form $l\,\theta\,a$, $a\,\theta\,l$, or $l\,\theta\,l'$ where l, l' are labels, a in a constant, and θ is an operator (e.g., $<$, $=$, $\in$). A *query* has the form

$$q(l_1, \ldots, l_k) \leftarrow c_1 \wedge \cdots \wedge c_n \tag{1}$$

where l_i are labels and c_j are conditions. We do not allow a label to appear more than once among $l_1, \ldots, l_k$. We sometimes denote the above query by $q(l_1, \ldots, l_k)$ or simply by q. We call the conjunction $c_1 \wedge \cdots \wedge c_n$ the *selection* of q and we call the sequence $(l_1, \ldots, l_k)$ the *projection* of q. Note that we allow the selection to be an empty conjunction of conditions. We denote the empty

conjunction by $\top$. We call queries with empty selections *project queries*. The set of labels appearing in either the selection or the projection of q is denoted $lbl(q)$. We will say that q is defined *over* the set $lbl(q)$.

Example 1. We present a few queries and their intuitive meaning.

- Pairs of titles and their respective prices:

$$\mathsf{q}(\mathsf{title}, \mathsf{price}) \leftarrow \top$$

- Titles and prices of books written by Dr. Suess that cost less than $12:

$$\mathsf{q}(\mathsf{title}, \mathsf{price}) \leftarrow (\mathsf{aname} = \text{`Dr.Suess'}) \wedge (\mathsf{price} < 12)$$

- Title, author and price of books written by Meyer:

$$\mathsf{q}(\mathsf{title}, \mathsf{aname}, \mathsf{price}) \leftarrow \text{`Meyer'} \in \mathsf{aname}$$

3 Query Semantics

Consider a query q and a tree T. Suppose that $lbl(q) = \{l_1, \ldots, l_k\}$. Intuitively, we can understand query evaluation as a two-step process. First, compute a relation R which contains tuples of nodes from T with labels $l_1, \ldots, l_k$ that are *related* in a *meaningful fashion*. We call this relation the *relational image* of T with respect to $l_1, \ldots, l_k$ and it is denoted $R(q, T)$. Next, evaluate the selection and projection given in q on $R(q, T)$ to derive the query result.

In order to compute the relational image of a tree with respect to a set of nodes, we must be able to decide which nodes are related in a meaningful fashion in a given tree. We observe that nodes are not meaningfully related if their relationship tree contains two different nodes with the same label. Intuitively, two nodes in a tree that have the same label correspond to different entities in the world. Thus, in Figure 1, nodes 22 and 24 are related. However, nodes 22 and 27 are not since their relationship tree contains the label book twice. This reflects the intuition that 22 is the price of the book with title 24 and not the price of the book with title 27.

We formalize this idea. Let $n_1, \ldots, n_k$ be nodes in T. We say that $n_1, \ldots, n_k$ are *interconnected*, denoted $\approx(n_1, \ldots, n_k)$, if the tree $T_{n_1,\ldots,n_k}$ does not contain any two nodes with the same label. We say that N is *maximally interconnected* with respect to a set of labels L if there is no strict superset N' of N with labels from L that is also interconnected. Now, given a query q over labels L, let $\mathcal{S}$ be the set of all sets of maximally interconnected nodes in T with labels from L. The relational image of T w.r.t. L is defined as follows

$$R(q, T) := \{t_{L,N} \mid N \in \mathcal{S}\}.$$

The relational image of a tree contains nodes that are related to each other. However, some such relationships may be more significant than others. Nodes

are more likely to be meaningfully related if their lowest common ancestor is relatively deep in the tree. If their lowest common ancestor is very high, then it is more likely that their relationship is coincidental. Thus, nodes 19 and 24 are more likely to be related then nodes 19 and 27. Note that in both these cases, the relationship trees do not have any repeated labels.

Let N be a set of interconnected nodes. We say that N' is an *improvement* on N, denoted $N \prec N'$ if

- $N \setminus N' = \{n_1\}$, $N' \setminus N = \{n_2\}$, $lbl(n_1) = lbl(n_2)$, i.e., N' is derived from N by replacing n_1 with n_2;
- For all nodes n in $N \cap N'$, the lowest common ancestor of $\{n_2, n\}$ is a descendent of the lowest common ancestor of $\{n_1, n\}$.

If N is maximal w.r.t. $\prec$, we say that N is $\prec$-*maximal*. We can remove some of the tuples in $R(q,T)$ that may be related in a less significant fashion, using the definition above. Let $\mathcal{S}^{\prec}$ be the set of all sets of maximally interconnected nodes in T that are also $\prec$-maximal. We define the $\prec$-*relational image* of T w.r.t. L as

$$R^{\prec}(q,T) := \{t_{L,N} \mid N \in \mathcal{S}^{\prec}\}.$$

Example 2. Consider the query

$$\mathsf{q}(\mathsf{title}, \mathsf{price}) \leftarrow (\mathsf{aname} = \text{`Dr. Suess'}) \wedge (\mathsf{price} < 12).$$

The $\prec$-relational image of q over the tree presented in Figure 1 is

title	aname	price
Goodnight Moon	M. Brown	$\perp$
Brown Bear	$\perp$	5.75
One Fish Two Fish	Dr. Suess	12.50
Cat in the Hat	Dr. Suess	10.95
Just Lost	M. Meyer	5.75
Just Lost	G. Meyer	5.75

We extend the function *val* to sets of tuples of nodes in the natural fashion. Note that the tuple

$$(\text{`One Fish Two Fish'}, \text{`Dr. Suess'}, 10.95) = val(17, 19, 22)$$

is not in $R^{\prec}(q,T)$, since its relationship tree contains the same label twice. The tuple

$$(\text{`Just Lost'}, \text{`Dr. Suess'}, 5.75) = val(27, 19, 33)$$

is also not in $R^{\prec}(q,T)$ since $(27, 19, 33) \prec (27, 31, 33)$. However, (‘Just Lost’, ‘Dr. Suess’, 5.75) is in $R(q,T)$.

When computing the result of a query, we may be interested in tuples that are related in a less or more significant fashion. Thus, we consider evaluating queries over both the relational image and the $\prec$-relational image of a tree. Note that both $R(q,T)$ and $R^{\prec}(q,T)$ can contain null values. Let $R_s(q,T)$ be the tuples in $R(q,T)$ that do not contain null values. We define $R_s^{\prec}(q,T)$ similarly. We differentiate between *strong query results*, which are derived from tuples which do not contain null values, and *weak query results*, which may be derived from tuples with null values. We extend the function *val* to sets of tuples of nodes in the natural fashion.

The *strong* result of a query $q(l_1,\ldots,l_k) \leftarrow c_1 \wedge \cdots \wedge c_n$ evaluated over a tree T is defined as

$$q(T)_s := \pi_{l_1,\ldots,l_k}(\sigma_{c_1\wedge\cdots\wedge c_n}(val(R_s(q,T)))).$$

We define the $\prec$-*strong* result of evaluating q over T, denoted $q(T)_s^{\prec}$ similarly by substituting $R_s^{\prec}(q,T)$ for $R_s(q,T)$ above. The *weak* result of evaluating q over T is

$$q(T)_w := \pi_{l_1,\ldots,l_k}(\sigma_{c_1\wedge\cdots\wedge c_n}(val(R(q,T))))$$

where selection is applied over tuples with null values using standard three-valued logic. We define the $\prec$-*weak* result, denoted $q(T)_w^{\prec}$, similarly.

Example 3. Recall the query q from Example 2. Both the $\prec$-strong and $\prec$-weak query result over the tree in Figure 1 is {('Cat in the Hat', 10.95)}.

As an additional example, evaluating the query $q(title, aname, price) \leftarrow \top$ will yield the $\prec$-weak query result containing all rows in the relation from Figure 1. The $\prec$-strong query result will contain all rows but the first two.

4 Computing Strong and $\prec$-Strong Query Results

In this section we consider the problem of finding strong query results and $\prec$-strong query results. We show that given a query and a tree, it is NP-complete to determine whether $q(T)_s$ or $q(T)_s^{\prec}$ is non-empty. We then present several different algorithms to compute strong and $\prec$-strong query results.

4.1 Complexity

We show that checking for nodes that are interconnected is NP-complete.

Theorem 1 (NP-Completeness of Interconnectiveness). *Let T be a labeled tree and let $l_1,\ldots,l_k$ be a set of labels. Determining whether there are nodes $n_1,\ldots,n_k$ in T such that $lbl(n_i) = l_i$ and $\approx(n_1,\ldots,n_k)$ is NP-complete.*

Proof. Membership in NP is easy. We show that the problem is NP-hard by a reduction from 3-SAT. Let U be a set of variables and let $F = C_1 \wedge \cdots \wedge C_k$ be conjunction of clauses over U such that each C_i is of size 3. We create a tree T from the formula. To simplify, we do not specify values of leaf nodes, since they may be chosen arbitrarily.

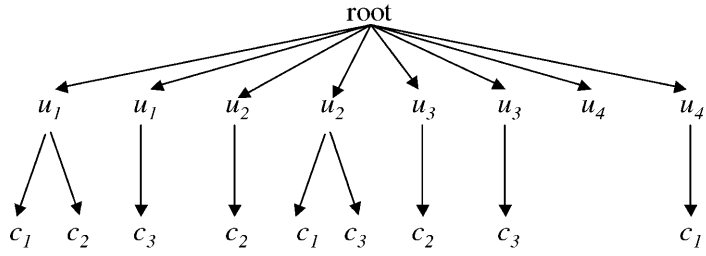

Fig. 2. A tree for an example formula

- The root of T is r and $lbl(r) = \text{root}$.
- For each variable $u \in U$ there are two children of r, namely u_p and u_n. Both nodes have label u.
- For each clause C_i and for each variable u in C_i, if u appears positively in C_i, then node u_p has a child labeled C_i. If u appear negatively in C_i, then node u_n has a child labeled C_i.

As an example, Figure 2 depicts the tree corresponding to the formula $(u_1 \vee \neg u_2 \vee \neg u_4) \wedge (u_1 \vee u_2 \vee u_3) \wedge (\neg u_1 \vee \neg u_2 \vee \neg u_3)$.

Let L be the set of labels $\{C_1, \ldots, C_k\}$. It is not difficult to see that F is satisfiable if and only if there are nodes $n_1, \ldots, n_k$ such that $lbl(n_i) = C_i$ and $T_{n_1,\ldots,n_k}$ does not contain the same label twice. □

Corollary 1 (Strong Query Results). *Let q be a project query and let T be a tree. Checking whether $q(T)_s$ is nonempty is NP-complete.*

In order to show that checking whether $q(T)_s^{\prec}$ is nonempty is also NP-complete we need a lemma.

Lemma 1. *Let N be a set of interconnected nodes. It is possible to check in polynomial time if N is $\prec$-maximal.*

Proof. We can check if N is $\prec$-maximal in the following fashion. For each node $n_1 \in N$ and for each other node n_2 in T with $lbl(n_1) = lbl(n_2)$ check if the set $N \setminus \{n_1\} \cup \{n_2\}$ is an improvement on N. This can be checked in polynomial time. □

Corollary 2 ($\prec$-Strong Query Results). *Let q be a project query and let T be a tree. Checking whether $q(T)_s^{\prec}$ is nonempty is NP-complete.*

4.2 Algorithms

In this section we present several different algorithms for computing strong query results. These algorithms can be used to compute $\prec$-strong results by adding a

phase in the computation where sets of nodes that are not $\prec$-maximal are discarded. Throughout this discussion, we consider a query q with labels $l_1, \ldots, l_k$ and a selection condition $c_1 \wedge \cdots \wedge c_n$. We present a naive method of computation. We then present two improvements on this method.

Naive Method. We divide query evaluation into two phases. In the first phase, given a tree T, find all k-tuples of nodes in T that have labels from $l_1, \ldots, l_k$. For each such tuple, compute the corresponding relationship tree. Discard the tuple if the relationship tree contains a repeated label. In the second phase, compute the selection and projection from q to yield the query result. Note that it possible to push selection when computing strong query results, but not when computing $\prec$-strong query results.

Incremental Method. We attempt to improve on the first phase of query evaluation, by taking advantage of the following property:

> **The A Priori Property:** Suppose that $\approx(n_1, \ldots, n_k)$. For all subsets, $\{n_{i_1}, \ldots, n_{i_j}\}$ of $\{n_1, \ldots, n_k\}$, it holds that $\approx(n_{i_1}, \ldots, n_{i_j})$.

The Incremental Method builds sets of interconnected nodes with labels $l_1, \ldots, l_k$ incrementally. First, the set $\mathcal{S}_1$, defined below, is computed:

$$\mathcal{S}_1 = \{\{n\} \mid lbl(n) = l_1\}.$$

Then, we compute $\mathcal{S}_{i+1}$ from $\mathcal{S}_i$ in the following fashion

$$\mathcal{S}_{i+1} = \{N \cup \{n\} \mid N \in \mathcal{S}_i \text{ and } lbl(n) = l_{i+1} \text{ and } \approx(N \cup \{n\})\}.$$

Note that when extending $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$, we try extending each set in $\mathcal{S}_i$ with each node labeled l_{i+1}. The set $\mathcal{S}_k$ contains exactly the same sets as those created in phase one of the Naive Method. This method tries fewer combinations of nodes than the Naive Method. However, it is guaranteed not to miss interconnected sets because of the a priori property.

Incremental Tree Walk Method. The Incremental Tree Walk Method improves upon the Incremental Method by not trying to extend each set of nodes in $\mathcal{S}_i$ with all nodes having label l_{i+1}. Instead, for each set N in $\mathcal{S}_i$, we find all nodes labeled l_{i+1} that are interconnected with N by *walking* around the tree surrounding N. Given a set N, these nodes are found as follows. Let L be the labels of the nodes in the relationship tree of N. Let n_0 be an arbitrary node in N. The nodes with label l_{i+1} that extend N to an interconnected set are derived by calling $\mathsf{TreeWalk}(n_0, l_{i+1}, N, L)$, presented in Figure 3. Intuitively, $\mathsf{TreeWalk}$ walks around the tree in all possible directions in an attempt to find nodes with label l_{i+1}. When a repeated label is found, the walk is terminated.

TreeWalk(n_0, l_{i+1}, N, L)

if $n_0 \notin N$ and $lbl(n_0) \in L$ **then return** $\emptyset$;
if $lbl(n_0) = l_{i+1}$ **then return** $\{n_0\}$;
$Res := \emptyset$;
for each neighbor n of n_0 **do** /* children and parent of n_0 */
    $Res := Res \cup$ TreeWalk$(n, l_{i+1}, N, L \cup \{lbl(n_0)\})$
return Res

Fig. 3. Incremental step of the Incremental Tree Walk Method

5 Computing Weak Query Results

In this section we consider the problem of finding weak answers to queries. Surprisingly, for project queries, weak answers can be found in polynomial time in the size of the input (i.e., the query and the document tree) and the output. However, the complexity of finding $\prec$-weak answers in still unknown. We present some necessary definitions and then a polynomial algorithm for computing weak answers.

Let N be a set of nodes. If there are no two nodes in N with the same label, we say that N is *proper*. Let N and N' be sets of nodes. We define a special union operator $\Cup$,

$$N \Cup N' := N \cup N' \setminus \{n \in N \mid \exists n' \in N'(lbl(n) = lbl(n'))\}.$$

Note that $\Cup$ is not a symmetric operator and that $N \Cup N'$ is proper if N and N' are proper.

Given a project query q over labels $l_1, \ldots, l_k$, the weak result of applying to a tree T is derived directly from the set $\mathcal{S}$ of sets of maximally interconnected nodes. By calling the procedure ComputeWeakResults$(\{l_1, \ldots, l_k\}, T)$, presented in Figure 4, we can compute the set $\mathcal{S}$.

We prove that the algorithm is correct with a series of lemmas.

Lemma 2. *If* $N \in$ ComputeWeakResults$(\{l_1, \ldots, l_k\}, T)$ *then* N *is interconnected.*

Proof. We show that only interconnected sets of nodes are added to $\mathcal{S}$ in the procedure. The sets added in lines 1 and 6 are clearly interconnected since they contain a single node. The set N_4 is proper since N_3 is proper and $N_1 \cup N_2$ is proper. The set N_5 is proper, since it is a subset of N_4 and it is also connected. Therefore, N_5 is interconnected. Clearly, $n_0 \in N_5$. Therefore, $(N_5 \cap N) \cup \{n_0\} \subseteq N_5$, added in line 15, is interconnected. □

Lemma 3. *Let N be a set of interconnected nodes and let $n_{lca} := lca\{N\}$. Suppose that $n \in N$ and N contains at least two nodes. Then, there is a node $n' \in N$ such that $n \neq n'$ and $n_{lca} = lca\{n, n'\}$.*

Proof. Suppose that $N = \{n_1, \ldots, n_k, n\}$. We define $n^i_{lca} := lca\{n, n_i\}$ for $i \leq k$. Note that since n^i_{lca} is an ancestor of n for all i, all the nodes n^i_{lca} lay on a single path. Let n_j be a node such that n^j_{lca} is an ancestor of n^i_{lca} for all i. Clearly, n^j_{lca} is then the lowest common ancestor of N. □

Lemma 4. *If N_{max} is a maximally interconnected set in T w.r.t. the labels $\{l_1, \ldots, l_k\}$, then $N_{max} \in$ ComputeWeakResults$(\{l_1, \ldots, l_k\}, T)$.*

Proof. We prove this claim by induction on k, the number of labels. For $k = 1$ the claim obviously holds. We assume correctness for $k - 1$ and prove for k.

Suppose that N_{max} is a maximally interconnected w.r.t. labels $\{l_1, \ldots, l_k\}$. If N_{max} does not contain any node with label l_k, then by the induction hypothesis, $N_{max} \in$ ComputeWeakResults$(\{l_1, \ldots, l_{k-1}\}, T)$. Clearly, N_{max} is also in ComputeWeakResults$(\{l_1, \ldots, l_k\}, T)$.

Otherwise, let n_0 be the node in N_{max} with label l_k. Then, $N_{max} \setminus \{n_0\}$ is interconnected. If $N_{max} \setminus \{n_0\}$ is empty, then $N_{max} = \{n_0\}$ is added to $\mathcal{S}'$ in line 6. Otherwise, by the induction hypothesis there is a set N, such that $N_{max} \setminus \{n_0\} \subseteq N$ and $N \in$ ComputeWeakResults$(\{l_1, \ldots, l_{k-1}\}, T)$.

Let n be a node in N_{max} such that $lca\{n_0, n\} = lca\{N_{max}\}$ and $n \neq n_0$. Such a node exists, by Lemma 3. Clearly, $n \in N$. Given the values defined for n_0, N and n, the set $N_1 \cup N_2$ (line 11) is proper since $\{n_0, n\}$ is interconnected. The set N_5 contain nodes from T_N that are interconnected with $\{n_0, n\}$. This hold since N_5 is both proper and connected. It is not difficult to see that $N_{max} = (N_5 \cap N) \cup \{n_0\}$ which is added to $\mathcal{S}'$ as required. □

Theorem 2 (Correctness of ComputeWeakResults**).** *Let N be a set of nodes. Then, $N \in$ ComputeWeakResults$(\{l_1, \ldots, l_k\}, T)$ if and only if N is maximally interconnected w.r.t. $\{l_1, \ldots, l_k\}$. In addition,* ComputeWeakResults *runs in polynomial time relative to the input and output.*

Proof. By Lemma 4, every maximally interconnected set is returned by ComputeWeakResults. By Lemma 2, every set returned by ComputeWeakResults is interconnected. Since line 17 removes sets that are contained in other sets, every set returned must also be maximal.

To show that ComputeWeakResults runs in polynomial time, it is sufficient to show that line 17 does not remove too many sets, since the rest of the procedure is clearly polynomial. Observe that in each iteration of the loop in line 2, the set $\mathcal{S}$ can only grow. Intuitively, this holds since sets created can never be *merged*. Therefore, at most a polynomial number of sets are removed. □

6 Conclusion

The concept of formulating select-project queries against XML documents was introduced. We believe that such queries can be the foundation for user interfaces

ComputeWeakResults($\{l_1, \ldots, l_k\}, T$)

1. $\mathcal{S} := \{\{n\} \mid n \in T \text{ and } lbl(n) = l_1\}$
2. **for** i = 2 to k **do** /* loop over labels */
3. $\mathcal{S}' := \emptyset$
4. **for each** $N \in \mathcal{S}$ **do** /* loop over all sets created so far */
5. **for each** $n_0 \in T$ with $lbl(n_0) = l_i$ **do** /* loop over all possible extensions to N */
6. $\mathcal{S}' := \mathcal{S}' \cup \{\{n_0\}\}$
7. **for each** $n \in N$ **do** /* loop over elements of N */
8. $n_{lca} := lca(n_0, n)$
9. $N_1 := \{$nodes on the path between n_0 and $n_{lca}\}$
10. $N_2 := \{$nodes on the path between n and $n_{lca}\}$
11. **if** $N_1 \cup N_2$ proper **then**
12. $N_3 := \{$nodes in $T_N\}$ /*nodes in relationship tree of N */
13. $N_4 := N_3 \Cup (N_1 \cup N_2)$
14. $N_5 := \{$nodes in N_4 that are connected by a path of nodes in N_4 to $n_0\}$
15. $\mathcal{S}' := \mathcal{S}' \cup \{(N_5 \cap N) \cup \{n_0\}\}$
16. $\mathcal{S} := \mathcal{S} \cup \mathcal{S}'$
17. Remove from $\mathcal{S}$ sets that are strictly contained in a set in $\mathcal{S}$
18. **return** $\mathcal{S}$

Fig. 4. Polynomial algorithm to compute weak results of a query on a tree

that allow easy formulation of queries by a naive user. Select-project queries can also be used to allow naive users to retrieve interesting portions of a document that are naturally related. Several different semantics for query evaluation were described. These semantics take into consideration that documents may not contain complete information, a situation that arises frequently in the context of the Web. The complexity of query evaluation was analyzed and evaluation algorithms were presented.

There are several interesting directions in which this work can be extended. Presently, query results are tuples of data. Allowing query results to be in XML could be useful. Our language can be extended to compute joins by allowing a

label to appear more than once in a query. We believe that it will not be difficult to extend our work to deal with such queries. By transforming XML documents to relations, it is possible to join these documents with other documents and to join them with relations, for example by using full-disjunction [10]. This allows integration of XML documents with relations and can be used as a complimentary method to the integration method suggested in [5]. It is also of interest to allow additional relational operators in a query.

References

1. S. Abiteboul. Querying semi-structured data. In *In Proc. of the 6th International Conference on Database Theory (ICDT)*, volume 1186 of *Lectures Notes in Computer Science*, pages 1–18. Springer-Verlag, January 1997.
2. S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying xml with incomplete information. In *Proc. of the 20th ACM Symp. on Principles of Databae Systems (PODS)*, Santa Barbara (California, USA), May 2001. ACM Press.
3. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML query language, June 2001. Available at `http://www.w3.org/TR/xquery`.
4. S. Cohen, Y. Kanza, Y. A. Kogan, W. Nutt, Y. Sagiv, and A. Serebrenik. Combining the power of searching and querying. In *In Proc. of the 7th International Conference on Cooperative Information Systems (CoopIS)*, volume 1901 of *Lecture Notes in Computer Science*, pages 54–65, Eilat, (Israel), September 2000. Springer.
5. S. Cohen, Y. Kanza, and Y. Sagiv. SQL4X: A flexible query language for XML and relational databases. In *Informal Proc. of the 8th International Workshop on Database and Programming Languages (DBPL)*, Marino, (Rome, Italy), September 2001.
6. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML, 1998. Available at `http://www.w3.org/TR/NOTE-xml-ql`.
7. R. Fagin, O. Mendelzon, and J. D. Ullman. A simplified universal relation assumption and its properties. *ACM Trans. on Database System (TODS)*, 7(3):343–360, 1982.
8. Y. Kanza, W. Nutt, and Y. Sagiv. Queries with incomplete answers over semistructured data. In *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, pages 227–236, Philadelphia, (Pennsylvania), May 1999. ACM Press.
9. D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundation of the universal relation model. *ACM Trans. on Database System (TODS)*, 9(2):283–308, 1984.
10. A. Rajaraman and J. D. Ullman. Integrating information by outerjoins and full disjunctions. In *Proc. of the 5th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Montreal, (Canada), June 1996. ACM Press.
11. Y. Sagiv. Can we use the universal instance assumption without using nulls? In *Proc. of the ACM SIGMOD Symp. on on the Management of Data*, pages 108–120, 1981.
12. J. D. Ullman. The U. R. strikes back. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS)*, pages 10–22, Los Angeles, (California), March 1982. ACM Press.
13. J. D. Ullman. *Principles of Database and Kowledge Base Systems*, volume II. Computer Science Press, 1989.