Generating Relations from XML Documents*

Sara Cohen, Yaron Kanza, and Yehoshua Sagiv

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

Abstract. This paper discusses several mechanisms for creating relations out of XML documents. A *relation generator* consists of two parts: (1) a tuple of path expressions *and* (2) an index indicating which path expressions may not be assigned the null value. Evaluating a relation generator involves finding tuples of nodes that satisfy the path expressions and are related to one another in a meaningful fashion. Different semantics for evaluation are given that take into account the possible presence of incomplete information. The complexity of generating relations from documents is analyzed and evaluation algorithms are described.

1 Introduction

Increasingly large amounts of data are accessible to the 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.

We present several mechanisms for creating relations out of documents. The relations created can be used in many different ways. One use is to integrate our mechanisms into SQL in order to allow simultaneous querying of relations and XML. Another scenario where our mechanisms are useful is to create a universal relation interface to a set of documents, thereby enabling a simple and powerful search of the documents. It has been noted that the universal relation [8,11, 12] is a first step towards facilitating natural-language querying of relational databases. We believe that this also holds for XML documents.

Given a tuple of path expressions, we aim to find tuples of nodes from a given document that (1) match the path expressions *and* (2) are *meaningfully related*. We first try to decide when a pair of nodes are meaningfully related. In principle, any pair of nodes are related by virtue of being in the same document. However, as humans we can often determine that nodes are or are not meaningfully related by simply looking at a document. Several questions arise in this context:

- How can we automate the decision of whether a pair of nodes are related in a meaningful fashion? This becomes especially difficult when one considers the fact that documents may have varied structure.

* This work was supported by The Israel Science Foundation (Grant No. 96/01-1)

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 285–299, 2003.

– How can we deal with incompleteness in documents? If a document may be incomplete, then we may have to discover whether a particular node is meaningfully related to a node that does not even appear in the document.

Our mechanism for deciding whether a pair of nodes is meaningfully related attempts to capture our human intuition of relationships in a document.

Once we have determined which pairs of nodes are meaningfully related, we propose several different mechanisms that allow us to determine when larger tuples of nodes are meaningfully related. The more precise the mechanism, the less coverage it tends to have. Therefore, we present several mechanisms in order to allow the user to choose an appropriate one for a given domain. We also discuss the complexity of finding meaningfully related tuples, which varies depending on the mechanism used.

Section 2 presents some definitions and Section 3 presents our relation generating mechanisms. Some scenarios where our mechanisms are useful are described in Section 4. In Sections 5 we discuss the complexity of creating relations out of XML documents and present evaluation algorithms. Section 6 concludes.

2 Framework

Relations. A *tuple* has the form $t = (c_1 \colon a_1, \ldots, c_k \colon a_k)$ where c_i and a_i are a *column name* and a *value*, respectively. Tuples can have columns with null values, denoted $\perp$. We also allow tuples to have multiple columns with the same name. We call $(c_1, \ldots, c_k)$ the *signature* of t. A *relation* $\mathcal{R}$ is a bag of tuples with the same signature, also called the signature of $\mathcal{R}$.

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 *label*(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 are three examples of such trees. In the sequel we will refer to these trees as $\mathcal{T}_{itm}$, $\mathcal{T}_{bk}$ and $\mathcal{T}_{athr}$. The nodes are numbered to allow easy reference.

Let T be a tree and let n_1 and n_2 be nodes in T. Let n be the *lowest common ancestor* of n_1 and n_2, and T_n be the subtree of T rooted at n. We denote by $T_{|n_1,n_2}$ the tree obtained by pruning from T_n all nodes other than n_1 and n_2 that are not ancestors of either n_1 or n_2. We call this tree the *relationship tree* of n_1 and n_2. For example in $\mathcal{T}_{bk}$, the lowest common ancestor of 8 and 13 is 7. The relationship tree of 8 and 13 is comprised of the nodes 7, 8, 10 and 13.

Graphs and Interconnection Graphs. In order to create a relation out of a tree, we first must be able to decide which pairs of nodes are related in a meaningful fashion in a given tree. The relationship tree comes to our aid for this purpose.

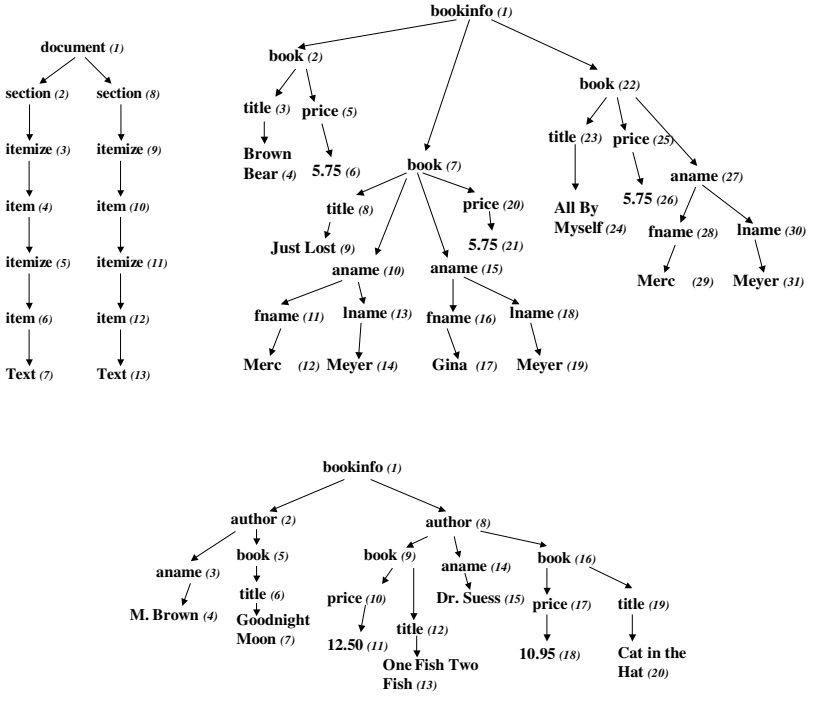

Fig. 1. Document $\mathcal{T}_{itm}$: itemize hierarchy (top left), $\mathcal{T}_{bk}$: bibliography grouped by book (top right), and $\mathcal{T}_{athr}$: bibliography grouped by author (bottom)

We start by giving an intuitive understanding of relationships in a document tree. Intuitively, a node in a tree represents an entity in the world. Two different nodes with the same label correspond to different entities of the same type. If n_a is an ancestor of n, then we may understand that n belongs to the entity that n_a represents. Now, suppose that nodes n and n' have distinct ancestors n_a and n'_a, respectively, such that n_a and n'_a have the same label. Suppose also that n'_a is not an ancestor of n and n_a is not an ancestor of n'. We may conclude that n and n' are not meaningfully related since they belong to different entities of the same type. Note that n_a and n'_a must be in the relationship tree of n and n'. Otherwise, they would be ancestors of both n and n' and would not imply that the nodes n and n' are not related.

We demonstrate and extend this intuition with a few examples. A formal definition of when nodes are related will be given later. Consider nodes 3 and 5 in $\mathcal{T}_{bk}$ (Figure 1). Their relationship tree does not contain two nodes with the same label. Therefore, nodes 3 and 5 are related. However, nodes 3 and 20 are not related since their relationship tree contains different nodes with the label book. This reflects the intuition that 3 is the title of the book with price node

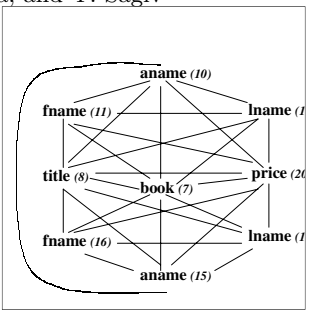

Fig. 2. The interconnection graph $\mathcal{IG}(\mathcal{T}_{bk}, \{7, 8, 10, 11, 13, 15, 16, 18, 20\})$

5 and not the title of the book with price node 20. Now, consider nodes 11 and 15 in $\mathcal{T}_{bk}$. Node 11 belongs to the aname node numbered 10. However, 15 is a different aname node. We may conclude that the fname in node 11 belongs to node 10 and is not related to node 15. Finally, consider nodes 10 and 15. These nodes share the same label. However, all their ancestors are the same, and thus, they belong to the same entities. Therefore, we may conclude that nodes 10 and 15 are meaningfully related. In fact, nodes 10 and 15 represent different author names, but they are related by virtue of belonging to the same book.

We formalize this idea. Let n and n' be nodes in T. We say that n and n' are *interconnected* if one of the following conditions holds:

- the relationship tree of n and n', i.e., $T_{|n,n'}$, does not contain two distinct nodes with the same label *or*
- the relationship tree of n and n', contains exactly one pair of distinct nodes with the same label and this pair is comprised of n and n'.

Given a tree T, we define the *interconnection graph* of T, denoted $\mathcal{IG}(T)$. This undirected graph has the same set of nodes as those in T. There is an edge between nodes n and n' if n and n' are interconnected. Given a tree T and a set of nodes $n_1, \ldots, n_k$, we denote the induced subgraph of $\mathcal{IG}(T)$, that contains the nodes $n_1, \ldots, n_k$, by $\mathcal{IG}(T, \{n_1, \ldots, n_k\})$. To illustrate these definitions, the interconnection graph $\mathcal{IG}(\mathcal{T}_{bk}, \{7, 8, 10, 11, 13, 15, 16, 18, 20\})$ is presented in Figure 2. Self loops have been omitted in the illustration.

We will be interested in interconnection graphs that have certain properties. A graph is *complete* if it contains an edge between every two nodes. A graph is *connected* if there is a path between every two nodes. Finally, a graph is a *star* if there is some node n that is connected by an edge to every other node in the graph. Note that every complete graph is also a star graph and every star graph is also connected.

3 Creating Relations from Trees

An *atomic path expression*, denoted α, is either a nonempty disjunction $(l_1 \mid \cdots \mid l_k)$ of labels from $\mathcal{L}$ or the *wildcard* symbol $*$. A *path expression*, denoted

ω, is either an atomic path expression, or of the form ω'/α or $\omega'/\!/\alpha$ where ω' is a path expression and α is an atomic path expression. We define recursively when a node n in a tree T *matches* a path expression ω, denoted $n \models \omega$:

- $\omega = (l_1 \mid \cdots \mid l_k)$, the root of T is n and $label(n) = l_i$ for some $i \leq k$;
- $\omega = *$ and the root of T is n;
- $\omega = \omega'/\alpha$, the parent of n matches ω' and $n \models \alpha$ in the subtree rooted at n;
- $\omega = \omega'/\!/\alpha$, there is an ancestor of n that matches ω' and $n \models \alpha$ in the subtree rooted at n.

For example, the path expression bookinfo/$*$ matches any child of a root that has the label bookinfo. The path expression $*/\!/$book/(aname | price) matches any aname or price node in a tree that is a child of a book node, regardless of the root's label. Finally, $*/\!/*$ matches any node in any tree. Note that given a node n and a path expression ω, it is possible to determine in polynomial time if $n \models \omega$.

The language of path expressions can be extended without affecting complexity results in this paper, as long as the extensions allow polynomial verification.

Let T be a tree and $(\omega_1, \ldots, \omega_m)$ be a tuple of path expressions. We are interested in finding tuples of nodes $(n_1, \ldots, n_m)$ from T that satisfy:

1. For all $i \leq m$, the node n_i matches the path expression ω_i *and*
2. The nodes $n_1, \ldots, n_m$ are *meaningfully related*.

Requirement 1 is easily verified. On the other hand, Requirement 2 is rather difficult to determine. In accordance with the intuition presented in Section 2, we use the interconnection graph of a tree as an aid in deciding whether a set of nodes are meaningfully related. The interconnection graph only contains information about pairs of nodes that are related. We present three different semantics that enable us to decide whether larger tuples of nodes are related. Later we will compare the semantics in terms of complexity and expressive power.

- **Completely-Interconnected:** We say that a set of nodes N are *completely-interconnected* in a tree T, denoted $\approx_c\{N\}$, if the interconnection graph of T, projected on N, i.e., $\mathcal{IG}(T, N)$, is a complete graph. Intuitively this states that a set of nodes is meaningfully related if every pair of nodes in the set is meaningfully related.
- **Reachably-Interconnected:** We say that a set of nodes N are *reachably-interconnected* in a tree T, denoted $\approx_r\{N\}$, if $\mathcal{IG}(T, N)$ is a connected graph. The intuition behind this notion is that meaningful relationships are transitive, i.e., if n_1 and n_2 are meaningfully related and so are n_2 and n_3, then n_1 and n_3 must also be meaningfully related.
- **Star-Interconnected:** We say that a set of nodes N are *star-interconnected* in a tree T, denoted $\approx_s\{N\}$, if $\mathcal{IG}(T, N)$ is a star graph. Intuitively, a set of nodes are meaningfully related if all the nodes in the set are meaningfully related to the same node.

Let $\Omega = (\omega_1, \ldots, \omega_m)$ be an m-tuple of path expressions and T be a tree with a set of nodes N. We say that a function μ: $\{1, \ldots, m\} \rightarrow N \cup \{\perp\}$ is a

matching of Ω to T if for all $i \leq m$, either $\mu(i) \models \omega_i$ or $\mu(i) = \perp$. We denote the set of nodes in the image of a matching μ by $Image(\mu)$. We use $Mat_T^c(\Omega)$ to denote the set of matchings μ, such that the nodes in $Image(\mu)$ are *completely-interconnected*, i.e., $\approx_c\{Image(\mu)\}$. Similarly, $Mat_T^r(\Omega)$ is the set of matchings μ, such that $\approx_r\{Image(\mu)\}$, and $Mat_T^s(\Omega)$ is the set of matchings μ, such that $\approx_s\{Image(\mu)\}$. It is not difficult to see that for all trees T and tuples of path expressions Ω,

$$Mat_T^c(\Omega) \subseteq Mat_T^s(\Omega) \subseteq Mat_T^r(\Omega).$$

Our matchings can have null values. However, we are interested in matchings that have maximal information. A matching μ' *subsumes* μ, denoted $\mu \sqsubseteq \mu'$, if μ' gives the same value for every index that is assigned a non-null value by μ, i.e., for all $i \leq m$, either $\mu'(i) = \mu(i)$ or $\mu(i) = \perp$. Intuitively, if μ' subsumes μ, then μ' contains more information than μ. An interconnection assignment μ is *maximal* if for every matching μ', we have that $\mu \sqsubseteq \mu'$ implies that $\mu = \mu'$.

Even a maximal matching can map some of the path expressions in Ω to null values. At times we may be interested in deriving only matchings that give non-null values to specific path expressions in Ω. We call the pair $\Delta = (\Omega, k)$, where Ω is an m-tuple of path expressions and $k \leq m$, a *relation generator*. For $\square \in \{c, r, s\}$, we use $MMat_T^{\square}(\Omega, k)$ to denote the set of maximal elements in $Mat_T^{\square}(\Omega)$ that do not map any of the *first k path expressions* to the null value.

A relation generator Δ can be used in order to create a relation out of parts of an XML document. Given $\Delta = (\Omega, k)$ and a tree T, we compute $MMat_T^{\square}(\Omega, k)$ with $\square$ chosen as desired. (See Examples 3.1, 3.2 and 3.3 below and complexity analysis in Section 5 for a comparison of the semantics.) We can create a relation with signature Ω out of $MMat_T^{\square}(\Omega, k)$, in the obvious way. Formally, for each matching μ in the set, our relation contains a tuple t_μ with value $\mu(i)$ in column ω_i. Depending on the end purpose of the relation, we may sometimes want to apply the *val* function to the nodes in the relation in order to derive tuples of strings instead of tuples of node ids.

Example 3.1. The relation generator $((*/\!/\mathsf{aname}, */\!/\mathsf{title}, */\!/\mathsf{price}), 1)$ finds triples of author names, titles and prices, all belonging to the same book. Only triples where the author name is non-null will be returned. For all three semantics, the relations created for $\mathcal{T}_{bk}$ ($\mathcal{T}_{athr}$) are the same. The tables for $\mathcal{T}_{bk}$ and $\mathcal{T}_{athr}$ are depicted in Figure 3(a) and 3(b). In parenthesis we note the value of the tuple after the *val* function is applied.

Note that there is no tuple corresponding to the title Brown Bear (node 4 in $\mathcal{T}_{bk}$) since it has no interconnected aname node. Observe also that the correct triples were found for both documents, even though they have different hierarchies.

Example 3.2. Using the tuple of path expressions $\Omega = (*/\!/\mathsf{title}, */\!/\mathsf{aname/fname}, */\!/\mathsf{aname/fname})$ we can create a relation generator that finds titles with pairs of first names of authors that wrote them. The set $MMat_T^c(\Omega, 3)$ will only contain

*//aname	*//title	*//price
10 (Mercy Meyer)	8 (Just Lost)	20 (5.75)
15 (Gina Meyer)	8 (Just Lost)	20 (5.75)
27 (Mercy Meyer)	23 (All By Myself)	25 (5.75)

(a) Evaluated over $\mathcal{T}_{bk}$

*//aname	*//title	*//price
3 (M. Brown)	6 (Goodnight Moon)	$\perp$
14 (Dr. Suess)	12 (One Fish Two Fish)	10 (12.50)
14 (Dr. Suess)	19 (Cat in the Hat)	17 (10.95)

(b) Evaluated over $\mathcal{T}_{athr}$

Fig. 3. Tables for $((*/\!/\mathsf{aname}, */\!/\mathsf{title}, */\!/\mathsf{price}), 1)$

matchings μ for which $\mu(2) = \mu(3)$, To see this, observe that in the interconnection graph in Figure 2, there are no two different fname nodes that are interconnected. This reflects the intuition that two fname nodes are not related since they belong to different authors. However, the sets $MMat^r_T(\Omega, 3)$ and $MMat^s_T(\Omega, 3)$ will contain matchings for the title Just Lost with the two different first names Mercy and Gina. Intuitively, this can be understood since the pair of names is related by virtue of both belonging to authors of the same title.

Consider now the tuple of path expressions $\Omega' = (*/\!/\mathsf{title}, */\!/\mathsf{aname/fname}, */\!/\mathsf{aname/lname})$. The set $MMat^c_T(\Omega', 3)$ will only contain matchings corresponding to first and last names of the same author. The sets $MMat^r_T(\Omega', 3)$ and $MMat^s_T(\Omega', 3)$ will contain in addition matchings with first names and last names of different authors of the same book. As before, such nodes are related since they belong to authors who wrote the same book. However, the set $MMat^c_T(\Omega', 3)$ may be perceived as a more precise answer. Therefore, we may conclude that choosing among the different semantics involves a trade-off between precision and recall (coverage).

Example 3.3. Tree $\mathcal{T}_{itm}$ in Figure 1 depicts two list hierarchies, as in a LaTeX document. Consider the tuple $\Omega = (*/\!/\mathsf{itemize}, */\!/\mathsf{item}, */\!/\mathsf{itemize}, */\!/\mathsf{item})$. The sets $MMat^c_T(\Omega, 4)$ and $MMat^s_T(\Omega, 4)$ do not contain matchings in which all nodes are different. The set $MMat^r_T(\Omega, 4)$ contains matchings corresponding to the quadruples (3, 4, 5, 6) and (9, 10, 11, 12). It does not, however, contain any matching with nodes from both list hierarchies.

4 Example Uses for Relation Generating Mechanism

Our mechanism for creating relations out of trees can be utilized in many different ways. We present several examples to illustrate possible uses.

4.1 Querying Trees and Relations Using SQL

Data is generally stored in relations. However, XML has become the standard for data exchange. Therefore, it is common to have both relations and XML as data sources. Posing queries against both types of data sources simultaneously is difficult. Our mechanism for translating trees to relations suggests one possible

solution. We extend the FROM clause of SQL to allow on-the-fly creation of relations from trees. Specifically, we use the predicates Complete, Reachable and Star to create tuples of nodes that match the given path expressions and are completely-, reachably- or star-interconnected, respectively.

For example, suppose that we wish to query the document $\mathcal{T}_{bk}$ and a table UserRatings(title, user, rating) stored in a relational database. The following query finds titles of books with a rating of at least 8 and author 'Smith'.

```
SELECT  Book.title
FROM    Complete(T_bk, ('*//title','*//aname'),0) as Book(title, author),
        UserRatings
WHERE   Book.title = UserRatings.title and
        Book.author like '%Smith%' and rating ≥ 8
```

The relation Book is created from the set $MMat^c_{\mathcal{T}_{bk}}((*/\!/title, */\!/aname), 0)$ by creating a tuple t_μ out of every matching μ in the computed set, in the obvious way. Note how we queried both XML and relations seamlessly.

4.2 An XML Search Engine

Currently, search engines cannot be used to query XML. More and more XML pages are finding their way onto the Web. Thus, it is becoming increasingly important to be able to query both the data and the meta-data content of the XML pages on the Web. Using the mechanism that we describe in this paper, a simple search language can be defined. A search query could have the form

$$path\_expression_1 : search\_phrase_1 \quad \cdots \quad path\_expression_n : search\_phrase_n$$

where $search\_phrase_i$ is a word or a quoted phrase. In addition, we allow the plus symbol to preface a path expression. Such path expressions must be matched to non-null values. Path expressions without a plus could be matched to null value.

As an example, the following query searches for books with a title containing the word XML and either the author Smith or no specified author.

$$+ */\!/\mathsf{title} : \mathsf{XML} \qquad */\!/\mathsf{aname} : \mathsf{Smith}$$

This query could be evaluated under any one of the three semantics (complete-interconnection, reachable-interconnection or star-interconnection). Basically, we would compute $MMat^c_T((*/\!/\mathsf{title}, */\!/\mathsf{aname}), 1)$, (or $MMat^r_T(\dots)$ or $MMat^s_T(\dots)$), on the documents T available. Then, only the created tuples that satisfy the search conditions, i.e., that contain the specified phrases, would be returned.

5 Complexity of Creating Relations from Trees

In this section, we discuss the complexity of computing the sets $MMat^\square_T(\Omega, k)$ for some $\square \in \{c, r, s\}$, relation generator (Ω, k) and tree T. The complexity of evaluation is likely to be one of the factors that influence the decision of which

semantics to employ for a specific purpose. We will use the measure of *input-output complexity*, an extension of *combined complexity*, when analyzing the complexity of generating relations under the different semantics. In combined complexity both the document and the query are part of the input. In input-output complexity, we analyze the complexity of a problem as a function of the input (i.e., query and document) and the output. The choice of this complexity measure is justified both because it is of greater theoretical interest and because queries and query results may be large. In general, we will be interested in the following questions.

- **Non-emptiness:** Is $MMat_T^{\square}(\Omega, k)$ non-empty?
- **Evaluation:** How can we compute $MMat_T^{\square}(\Omega, k)$ efficiently?

For $\square = s$, i.e., when star-interconnected nodes are desired, it is not difficult to see that a relation generator can always be computed in polynomial time.

Theorem 5.1 (Evaluation). *Let (Ω, k) be a relation generator and T be a tree. Then $MMat_T^s(\Omega, k)$ can be computed in polynomial time under input-output complexity.*

We present complexity results for the problems of non-emptiness and of evaluation for $\square = c$ and $\square = r$ below.

5.1 Complete-Interconnections

We solve the problems above for $\square = c$, i.e., when only completely-interconnected nodes are desired. We first show that given a relation generator (Ω, k) and a tree T, determining whether $MMat_T^c(\Omega, k)$ is non-empty is an NP-complete problem.

Theorem 5.2 (Non-emptiness). *Let T be a tree and let (Ω, k) be a relation generator. Determining whether $MMat_T^c(\Omega, k) \neq \emptyset$ is NP-complete.*

Proof. (Sketch) The proof is by a reduction from 3-SAT, and is omitted because of space limitations. □

Since determining non-emptiness is NP-complete, we do not consider the general problem of finding all query results. There are, however, several important cases in which query evaluation is polynomial under input-output complexity. The first such case is when every path expression can be assigned the value null, i.e., for relation generators of the form $(\Omega, 0)$.

Theorem 5.3 (Evaluation (Case 1)). *Let Ω be a tuple of path expressions and T be a tree. The set $MMat_T^c(\Omega, 0)$ can be computed in polynomial time under input-output complexity.*

Proof. (Sketch) Basically, we build up matchings in $MMat_T^c(\Omega, 0)$ gradually. We start with the matching that maps all path expressions to the null value and then try to extend this matching in all possible ways. This must be done in a

CompleteInterconnections$((\omega_1, \ldots, \omega_m), T)$

1. $\mathcal{M} := \{\emptyset\}$
2. **for** $i := 1$ to m **do**
3. $\quad \mathcal{M}' := \emptyset$
4. $\quad$ **for each** $n \in T$ such that $n \models \omega_i$ **do**
5. $\quad\quad$ **for each** $\mu \in \mathcal{M}$ **do**
6. $\quad\quad\quad \mu' := \{(i, n)\} \cup \{\, (j, n') \in \mu \mid n'$ and n are interconnected $\}$
7. $\quad\quad\quad \mathcal{M}' := \mathcal{M}' \cup \{\mu'\}$
8. $\quad \mathcal{M} := \mathcal{M} \cup \mathcal{M}'$
9. $\quad$ Remove strictly subsumed matchings from $\mathcal{M}$
10. **return** $\mathcal{M}$

Fig. 4. Polynomial algorithm to compute $MMat^c_T(\Omega, 0)$ (Theorem 5.3)

careful fashion to make sure that we do not create many matchings that will subsequently be removed, because they are not maximal.

An algorithm for creating $MMat^c_T(\Omega, 0)$ is presented in Figure 4. In this procedure, we represent matchings as sets of pairs of indices and nodes. For each matching we only represent explicitly pairs for which the index is not mapped to the null value.

We collect matchings in the set $\mathcal{M}$. In Line 1, the matching that maps all path expressions to null is added. Then, we loop over the path expressions (Line 2) and collect in $\mathcal{M}'$ matchings that map the i-th path expression to a non-null value. This is done by looping over all nodes n that satisfy the i-th path expression (Line 4) and over all matchings μ created thus far (Line 5). We try to extend μ with n; however, nodes that are not interconnected with n must be left out in order to derive a set of completely-interconnected nodes (Line 6). Subsumed assignments are removed in Line 9.

A formal proof of correctness is omitted due to lack of space. □

Even when we do not allow some of the path expressions to be mapped to the null value, it may still be possible to evaluate a relation generator in polynomial time under input-output complexity. We first present some necessary definitions and then describe a case in which polynomial evaluation is possible.

We say that a tree T is *recursive* if T contains nodes n, n' such that n' is a strict descendent of n and $label(n) = label(n')$. Otherwise, we say that T is *non-recursive*.

Given a tree T and a path expression ω, we use $MatchingNodes(\omega, T)$ to denote the set of nodes in T that match ω and $MatchingLabels(\omega, T)$ to denote the set of all labels of nodes in $MatchingNodes(\omega, T)$. We denote by $LabelsAbove(\omega, T)$ all the labels l, such that there is a node labeled l in T with a descendent $n \in MatchingNodes(\omega, T)$. Note that by definition, if $n \models \omega$, then

$label(n) \in LabelsAbove(\omega, T)$. We associate each non-recursive tree T and path expression ω with a relation $\mathcal{R}_{\omega,T}$. This relation has a column for each label in $LabelsAbove(\omega, T)$. For each node $n \in MatchingNodes(\omega, T)$, the relation $\mathcal{R}_{\omega,T}$ has a tuple t_n. The value of t_n in column l is the node id of the ancestor of n with label l, if such an ancestor exists. Otherwise, the value in column l is null. Since T is non-recursive, there is at most one ancestor with label l for any given node n. Thus, the tuples are well-defined. For each tuple t_n created from node n in $\mathcal{R}_{\omega,T}$, the *originating* column of t_n, denoted $Originating(t_n)$, is the column in which n appears in tuple t_n, i.e., the column $label(n)$.

The notion of creating a hypergraph out of a set of relations has been studied for query optimization [12]. We review this idea briefly here. Relations $\mathcal{R}_1, \ldots, \mathcal{R}_m$ give rise to a hypergraph $\mathcal{H}(\mathcal{R}_1, \ldots, \mathcal{R}_m)$ in the following fashion:

- $\mathcal{H}(\mathcal{R}_1, \ldots, \mathcal{R}_m)$ has a node for each column in $\mathcal{R}_1, \ldots, \mathcal{R}_m$;
- for each relation $\mathcal{R}_i$, there is a hyperedge in $\mathcal{H}(\mathcal{R}_1, \ldots, \mathcal{R}_m)$ containing the nodes in the signature of $\mathcal{R}_i$, i.e., all the columns appearing in $\mathcal{R}_i$.

A hyperedge e is an *ear* if (1) e is the only hyperedge in the hypergraph *or* (2) there is a hyperedge e' such that all nodes in $e \setminus e'$ are only in edge e. We call the removal of e from the hypergraph *ear removal*. The *GYO-reduction* of a hypergraph is the result of applying ear removals (combined with the removal of nodes that do not belong to any hyperedge) until there remains no ear in the hypergraph. It has been proven that the GYO-reduction of a hypergraph is unique, i.e., is not dependent on the order in which ears are removed. A hypergraph is *acyclic* if its GYO-reduction is an the empty hypergraph (see also [5,15]). It has been proven [14,1,13] that if $\mathcal{H}(R_1, \ldots, R_m)$ is acyclic, then the natural join of $R_1, \ldots, R_m$ can be computed in polynomial time under input-output complexity.

The natural join of two relations requires equality on shared columns. By definition, the null value is never equal to any other value. Therefore, if a tuple has the null value in a shared column, this tuple will be lost in the result of the join. We define the pseudo natural join that differs from the natural join on exactly this issue, i.e., on how null values are dealt with. When performing a pseudo natural join, null values are always equal to any other value. The new value for the shared column, however, will be the non-null value, if such a value exists. For example, the pseudo natural join of the relations $\{(a\colon 1, b\colon \bot, c\colon 3), (a\colon 2, b\colon 3, c\colon 3)\}$ and $\{(a\colon \bot, b\colon 2, d\colon 4), (a\colon 1, b\colon \bot, d\colon 5)\}$ is the relation $\{(a\colon 1, b\colon 2, c\colon 3, d\colon 4), (a\colon 1, b\colon \bot, c\colon 3, d\colon 5)\}$. The pseudo natural join of relations with an acyclic hypergraph can be computed in polynomial time, in a fashion similar to computing the natural join. Note that the pseudo natural join, unlike the outer join, does not keep tuples that do not match other tuples in the way specified above.

Theorem 5.4 (Evaluation (Case 2)). *Let $((\omega_1, \ldots, \omega_m), m)$ be a relation generator and T be a tree. Then $MMat^c_T((\omega_1, \ldots, \omega_m), m)$ can be computed in polynomial time under input-output complexity if the following conditions hold:*

1. *T is non-recursive,*
2. *$\mathcal{H}(\mathcal{R}_{\omega_1,T}, \ldots, \mathcal{R}_{\omega_m,T})$ is acyclic,* and
3. *for all $i, j \leq m$, the sets MatchingLabels(ω_i, T) and MatchingLabels(ω_j, T) are disjoint if $i \neq j$.*

Proof. (Sketch) Let T be a non-recursive tree. We are searching for sets of nodes that are completely-interconnected. Suppose that n and n' are interconnected and that n has an ancestor n_l with label l. Then, in the tree T the node n' must either not have any ancestor with label l, or have n_l as its ancestor. If the tree is non-recursive, then this condition is sufficient and necessary for two nodes to be interconnected. Formally, for a non-recursive tree T, nodes n and n' are interconnected if and only if the following conditions hold.

- For every ancestor n_l of n, either n_l is an ancestor of n' or n' has no ancestor with label *label*(n_l).
- For every ancestor n'_l of n', either n'_l is an ancestor of n or n has no ancestor with label *label*(n'_l).

Now, suppose that ω is a path expression for which $n \models \omega$. Similarly, suppose that $n' \models \omega'$. It is not difficult to see that the tuples t_n in $\mathcal{R}_{\omega,T}$ and $t_{n'}$ in $\mathcal{R}_{\omega',T}$ will create a tuple in the pseudo natural join of $\mathcal{R}_{\omega,T}$ and $\mathcal{R}_{\omega',T}$ if and only if n and n' are interconnected.

In order to compute the set $MMat^c_T((\omega_1, \ldots, \omega_m), m)$, we first compute the pseudo natural join $\mathcal{R}$ of $\mathcal{R}_{\omega_1,T}, \ldots, \mathcal{R}_{\omega_m,T}$. This can be done in polynomial time, since $\mathcal{H}(\mathcal{R}_{\omega_1,T}, \ldots, \mathcal{R}_{\omega_m,T})$ is acyclic. Now, consider a tuple t in $\mathcal{R}$. Let $t_1, \ldots, t_m$ be the tuples in $\mathcal{R}_1, \ldots, \mathcal{R}_m$ that joined together to create t. Then, the projection of t on the columns *Originating*$(t_1), \ldots,$ *Originating*(t_m) is a tuple containing completely-interconnected nodes. Since the sets *MatchingLabels*(ω_i, T) and *MatchingLabels*(ω_j, T) are disjoint for all i and j, this tuple contains exactly m different nodes. In this fashion, we can create all the matchings out of the tuples in the pseudo natural join. □

Building on Theorems 5.3 and 5.4, we can prove the following theorem.

Theorem 5.5 (Evaluation (General)). *Let $((\omega_1, \ldots, \omega_m), k)$ be a relation generator and T be a tree. Then $MMat^c_T((\omega_1, \ldots, \omega_m), k)$ can be computed in polynomial time under input-output complexity if the following conditions hold:*

1. *either $k = 0$ or T is non-recursive,*
2. *$\mathcal{H}(R_{\omega_1,T}, \ldots, R_{\omega_k,T})$ is acyclic,* and
3. *for all $i, j \leq k$, either*
   - *MatchingLabels(ω_i, T) and MatchingLabels(ω_j, T) are disjoint,* or
   - *MatchingNodes(ω_i, T) = MatchingNodes(ω_j, T).*

5.2 Reachable-Interconnections

We solve the problems of non-emptiness and evaluation for $\square = r$, i.e., when only reachably-interconnected nodes are desired. We first show that given a relation

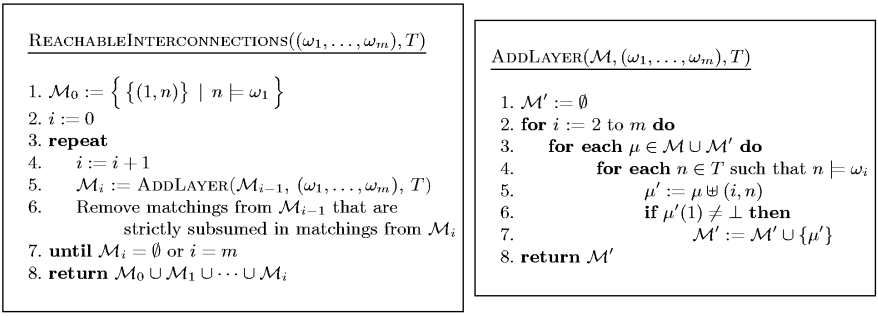

Fig. 5. Polynomial algorithm to compute $MMat^r_T(\Omega, 1)$ (Theorem 5.7)

generator Ω, an integer k and a tree T, determining whether $MMat^r_T(\Omega, k)$ is non-empty is an NP-complete problem. This is rather surprising since finding reachably-interconnected nodes only requires finding connected subgraphs in an interconnection graph.

Theorem 5.6 (Non-emptiness). *Let T be a tree and let (Ω, k) be a relation generator. Determining whether $MMat^r_T(\Omega, k) \neq \emptyset$ is NP-complete.*

Proof. (Sketch) The proof is by a reduction from 3-SAT and is omitted because of space limitations. □

As in Section 5.1, we present an important case in which query evaluation is polynomial. If all path expressions can be assigned the null value, then the evaluation can be performed in polynomial time.

Theorem 5.7 (Evaluation). *Let Ω be a tuple of path expressions and T be a tree. Then $MMat^r_T(\Omega, 0)$ can be computed in polynomial time under input-output complexity.*

Proof. (Sketch) Conceptually, creating matchings in $MMat^r_T(\Omega, 0)$ is more difficult than creating matchings in $MMat^c_T(\Omega, 0)$. The difficulty stems from the fact that nodes may be together in the image of a matching even if they are not directly interconnected, but rather connected by a path of nodes also in the image of the matching.

As in the proof of Theorem 5.3, we represent matchings as sets of pairs of indexes and nodes. We introduce some notation used in the algorithm. Given a matching μ and a set of nodes N contained in the image of μ, we define $\mathit{ConnectedSubMatching}(\mu, N)$ as the set of pairs (i, n) in μ, such that n is connected to all nodes in N in the graph $\mathcal{IG}(T, \mathit{Image}(\mu))$. Clearly, if $\approx_r\{\mathit{Image}(\mu)\}$, then $\mathit{ConnectedSubMatching}(\mu, N) = \mu$.

The operator $\updownarrow$ replaces the value for an index in a matching. The operator $\uplus$ performs an exchange and then removes non-connected nodes. Formally,

$$\mu \updownarrow (i, n) \stackrel{\text{def}}{=} \{ (j, n') \in \mu \mid j \neq i \} \cup \{ (i, n) \}$$
$$\mu \uplus (i, n) \stackrel{\text{def}}{=} \mathit{ConnectedSubMatching}(\mu \updownarrow (i, n), \{n\}).$$

In Figure 5 we present a polynomial algorithm for finding all matchings in $MMat_T^r(\Omega, 0)$ that do not assign ω_1 the null value, i.e., the set $MMat_T^r(\Omega, 1)$. A complete proof of correctness is not given due to space limitations. In Line 1 of ReachableInterconnections, we create a matching for each node that matches ω_1. While we succeed in extending a matching (Lines 3 and 7), we call the procedure AddLayer. There we loop over all the rest of the path expressions (Line 2). For each path expression, each matching μ created thus far, and each node n matching the current path expression, we try to extend the μ with n (Line 5). Only if the matching created still gives a non-null value to ω_1, do we add this matching to the set of matchings created (Line 6). Since it is possible to find all matchings that do not assign ω_1 a null value in polynomial time, we can repeat this process for each of the path expressions ω_i and thus, derive a polynomial algorithm for computing $MMat_T^r(\Omega, m)$. □

6 Conclusion

Relation generators, used for producing relations from XML documents were defined. Our relation generators allow naive users to retrieve interesting and naturally related portions of a document. Thus, they can be used for integrating relations and XML and as a foundation for XML search engines.

We presented the notion of an interconnection graph which describes connections between pairs of nodes. Several different semantics for finding larger tuples of related nodes from the interconnection graph 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. Note that almost all our complexity results still hold even if the interconnection graph is created differently. For example, it is likely that the interconnection graph would be defined differently in the presence of either a schema or IDs and IDREFs. Once the interconnection graph was defined in this context, our mechanisms for creating tuples of meaningfully related nodes could be used.

This work extends [3]. Generating relations from semistructured data has been considered in the context of wrapper generation and inferring schemas from documents [4,6,9]. In [10], relations are created from semistructured data by schema generation. However, their approach is different as they attempt to "reverse-engineer" a website, while we simply look for semantic relationships between entities. Hence, the work in [10] is most applicable when the data has been constructed in a systematic manner, whereas our approach can be used even when the data does not conform to any schema. Interestingly, for many important special cases our complete answers coincide with their relations that are created using compact skeletons. In [2,7] a query language that uses a flexible semantics which can deal with variations in the data structure was presented. However, their approach was more restricted and their focus was on query equivalence.

One important open problem is how to define indices over an XML document that will allow relation generators to be quickly evaluated. Since some of the tuples produced by a relation generator may be more relevant than others, a

ranking system for such tuples should be defined. We intend to implement the mechanisms described here and to perform extensive experimentation in order to discover which semantics perform best and return the best results in practice.

References

[1] U. Chakravarthy and J. Minker. Multiple query processing in deductive databases using query graphs. In *Proceedings of International Conference on Very Large Data Bases*, pages 384–391. Morgan Kaufmann, 1986.

[2] S. Cohen, Y. Kanza, and Y. Sagiv. SQL4X: A flexible query language for XML and relational databases. In *Proc. of the 8th International Workshop on Database and Programming Languages (DBPL)*, pages 263–280, Marino, (Rome, Italy), September 2001. Springer-Verlag.

[3] S. Cohen, Y. Kanza, and Y. Sagiv. Select project queries over xml documents. In *Proc. 5th Workshop on Next Generation Information Technologies and Systems*, pages 2–13, Caesarea (Israel), June 2002. Springer-Verlag.

[4] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A system for extracting document type descriptors from xml documents. In *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, pages 165–176, Dallas (Texas, USA), May 2000. ACM Press.

[5] M. Graham. On the universal relation. Technical report, University of Toronto, Toronto (Canada), 1979.

[6] A. Gupta, V. Harinarayan, and A. Rajaraman. Virtual database technology. In *Proc. 14th International Conference on Data Engineering*, pages 297–301, Orlando (Florida, USA), Feb. 1998. IEEE Computer Society.

[7] Y. Kanza and S. Sagiv. Flexible queries over semistructured data. In *Proc. 20th Symposium on Principles of Database Systems*, pages 40–51, Santa Barbara (California, USA), May 2001. ACM Press.

[8] 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.

[9] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. 1998 ACM SIGMOD International Conference on Management of Data*, pages 295–306, Seattle (Washington, USA), June 1998. ACM Press.

[10] A. Rajaraman and J. D. Ullmann. Querying websites using compact skeletons. In *Proc. 20th Symposium on Principles of Database Systems*, pages 16–27, Santa Barbara (California, USA), May 2001. ACM Press.

[11] 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.

[12] J. D. Ullman. *Principles of Database and Kowledge Base Systems*, volume II. Computer Science Press, 1989.

[13] E. Wong and K. Youssefi. Decomposition-a strategy for query processing. *ACM Trans. on Database Systems*, 1(3):223–241, 1976.

[14] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of International Conference on Very Large Data Bases*, pages 82–94. Morgan Kaufmann, 1981.

[15] C. Yu and M. Özsoyoglu. An algorithm for tree-query membership of a distributed query. In *Proceedings of IEEE COMPSAC*, pages 306–312, 1979.