A COST-BASED APPROACH FOR CONVERTING RELATIONAL SCHEMAS TO XML

Ramon Lawrence
Iowa Database and Emerging Applications (IDEA) Lab, University of Iowa
ramon-lawrence@uiowa.edu

ABSTRACT: *Converting relational database schemas to XML schemas is important as applications originally written to manipulate relational data are converted to XML-enabled applications. An important challenge is determining good XML schemas from existing relational schemas, so that applications may be migrated to use XML. In this paper, we propose a general algorithm for converting relational schemas to XML schemas that returns the optimal XML encoding exhibiting no redundancy and respects user constraints on the resulting XML schema. Previous conversion algorithms required the user to specify a complete mapping or did not allow user-specified constraints. The conversion is performed based on a user-supplied cost function. In this paper, the cost measure is based on the space efficiency of the XML encoding as a metric for measuring the XML schema's desirability. The result is a fully automatic system for migrating relational schemas to XML schemas while respecting conversion constraints.*

Keywords: *XML, schema conversion, schema mapping, space efficiency, application migration*

1. INTRODUCTION

As XML becomes the standard for data exchange, existing applications are being migrated from using relational databases to XML. One of the challenges of this migration process is creating a new XML schema from the existing relational schema. Since considerable effort was invested in designing a good relational schema, the goal is to preserve this investment while at the same time exploit the modeling advantages inherent in XML. The major modeling advantage of XML over the relational model is the ability to define nested schemas. This allows data to be displayed more naturally and organized more efficiently. A result is that the number of foreign keys and joins necessary to relate data in XML is reduced.

Although converting a relational schema into an XML schema is trivial if a flat translation is used (no nesting), flat translations do not take advantage of XML's hierarchical nature. By exploiting nesting in an XML document, it is possible to define XML encodings of the relational data that are more natural, easier to read, faster to query, and are more space efficient. In this work, we present an algorithm for translating relational schemas to XML using a specified cost-function. The cost-function used measures space efficiency.

The contributions of this work are:

- A fully automated relational to XML schema conversion algorithm that minimizes a user-specified cost function (in this case related to space efficiency) that does not require the user annotate the relational schema or map to an intermediate model.
- The algorithm incorporates user preferences during the mapping if they are present. Thus, the mapping can be completed with no user involvement, or as much involvement as the user desires. The algorithm then determines the optimal mapping based on the user's constraints on their desired form of the XML schema.
- Empirical results demonstrating how XML nesting improves on flat translation by creating more space efficient schemas without introducing redundancy.

We begin the discussion with a short motivation (Section 2) of the importance of relational to XML translation. Section 3 covers background work and overviews the current translation algorithms. In Section 4, we discuss the advantages that nesting in XML has over flat relations in the relational model, and how we can exploit this hierarchical nature to improve readability and space efficiency. Nesting allows more efficient XML encoding of the same data than a flat translation, and we develop a cost metric to quantify this advantage. An algorithm that uses the cost metric to develop the optimal space efficient mapping from relational to XML is given in Section 5. An advantage of this algorithm is that user input in the form of constraints on the desired form of the XML schema can be quantified as costs, and the algorithm will compute the optimal solution given the constraints. Section 6 provides experimental results that demonstrate the advantages of the algorithm. The paper closes with future work and conclusions.

2. MOTIVATION

Converting relational databases to XML is increasingly important as data stored in relational databases is accessed and exchanged using XML. Research has been performed on the query aspects of how to build XML documents efficiently when queries are generated through an XML view of a relational database [8,14]. Although efficient query generation and execution is a major challenge, another important consideration is the ability to automatically generate entire XML schemas from existing relational schemas.

To rapidly migrate existing relational database applications to XML applications, the relational schema must be mapped to an XML schema. Ideally, this mapping should be automatic and preserve the considerable effort required to develop the relational schema. It is clearly undesirable for the XML schema to be totally redesigned if the application domain is mostly unchanged. Rather, an automatic algorithm that converts the relational schema into a good XML schema is desired. This conversion algorithm should require minimal user input, preserve the normalization present in the relational schema, and take advantage of the nesting in XML schemas to improve readability and storage efficiency. Automatic tools that generate good XML schemas from relational schemas with minimal user input would save considerable time during such software conversions.

Selecting space efficient XML representations is important as encoding data in XML has high overhead, which affects storage space and query performance. Even when exploiting XML compression [15], it is desirable to develop an XML schema that encodes the information efficiently as this reduces the number of data elements and the file size before compression. It is useful to have a tool that indicates the desirability of a particular XML encoding with respect to others. Although there are multiple possible metrics [11] for desirability, including redundancy avoidance (normalization), readability, and query efficiency, the metric used in this paper is space efficiency. A cost-based approach has been used for the reverse mapping of XML documents to the relational model [4].

3. BACKGROUND

One of the challenges in relational to XML translation is that schema overhead is as big a factor as data redundancy. Thus, the ability to avoid encoding data items has a double benefit: the data item itself is not encoded and its accompanying tags are not encoded. Further, the translation should be easily expressible and preferably require minimal user input. The common weakness with most approaches is that the relational model is mapped to a different data model before conversion to XML and this procedure requires human involvement. Once the user has built the intermediate model, they have limited impact on the final result as the mapping is performed by translation rules.

This work is unique in that it does not require the user understand intermediate models for conversion and allows the user to specify constraints on the mapping that are respected by the algorithm. Note this work is not about XML normalization [1]. The assumption is that the relational schema has already been normalized, and the algorithm must only ensure redundancies do not get re-introduced during conversion.

We overview four categorizes of translation methods:

- **Flat Translation** - converts relations to XML without using nesting [12]. Each relation is translated to an element *E*, and each relational attribute is either translated to a subelement (element approach) or attribute (attribute approach) of *E*.
- **Query-based Translation** - conversion occurs by using a query extraction language that is an extension of SQL or a new XML query language. SilkRoute [8] and XML publishing using a relational database engine [14] are examples of this approach.
- **Model-based Translation** - converts the relational schema to an intermediate model that is then mapped to XML using conversion rules. Approaches include [3,5,7]. Challenges include that conceptual models are not always the starting point for conversion to XML, and the user must use the intermediate model.
- **Dependency-based Translation** - converts the relational schema using dependency information. Two algorithms, NeT and CoT [12], are defined. NeT determines optimal nestings of attributes from a single table, while CoT is used to determine nestings of multiple tables. CoT is a schema-level nesting algorithm (that uses inclusion dependencies), and NeT is a data-level nesting algorithm that scans the data to determine when nesting is appropriate. In most cases, the payoff for schema-level nesting is much greater than data-level nesting as most relational databases are highly normalized, and data-level nesting is very costly.

Commercial vendors have translation modules [2,10,13] that convert from relational to XML. Most of these systems rely on either simple flat translation or extraction queries. Simple flat translation is easy to implement, but not necessarily the best encoding, as it does not exploit the hierarchical nature of XML to improve any of the desirable properties listed previously. Specifying queries requires extensions to

the base SQL language or new query languages [8] and is a lot of work. The extraction queries convert relational data to XML, but they do not migrate entire relational schemas to XML.

In summary, flat translation is space-inefficient, whereas query or model-based translations require user input during the processing and are often complicated to specify. None of the previous approaches developed a cost metric to evaluate good XML schemas beyond avoiding redundancy. Our algorithm is the only one that allows the user to enter constraints on the final form of the schema that are respected by the algorithm. Although the CoT algorithm [12] is similar to ours as it also uses dependency information, it does not allow for user-specified constraints and does not allow user-specified cost functions to guide the translation. Since the CoT algorithm does not use a cost function, it only generates *some* nesting based on inclusion dependencies (not necessarily the lowest cost nesting).

This paper contributes by defining a fully automatic relational to XML schema translation algorithm that uses only foreign key dependencies extracted using standard technologies to produce an optimal, space-efficient XML schema. Unlike all previous approaches, the algorithm has the ability to incorporate user constraints on the form of the XML schema desired and produce the optimal encoding given those constraints. It can also be adapted to use any arbitrary cost function as specified by the user.

part (**p_partkey**, p_name, …, p_retailprice)
supplier (**s_suppkey**, s_name, …, *s_nationkey*, s_acctbal)
partsupp (***ps_partkey***, ***ps_suppkey***, …, ps_availqty)
customer (**c_custkey**,c_name, *c_nationkey*, …, c_comment)
orders (**o_orderkey**, *o_custkey*, …, o_orderdate, o_comment)
lineitem (***l_orderkey***, **l_linenumber**, *l_partkey*, *l_suppkey*, l_quantity, l_extendedprice, … , l_comment)
nation (**n_nationkey**, n_name, *n_regionkey*, n_comment)
region (**r_regionkey**, r_name, r_comment)

Figure 1. TPC-H Schema and Inclusion Dependencies (Primary keys are in bold; foreign keys are in italics.)

4. NESTING AND EFFICIENCY

Exploiting nesting in XML has three major advantages:

- Improves readability by grouping related concepts.
- Improves space efficiency by avoiding the repetition and encoding of foreign keys that can be implicitly defined by the hierarchical structure.
- Improves query efficiency by clustering concepts together and avoiding joins.

If nesting is used indiscriminately, it is possible to re-introduce redundancy that was eliminated by normalization. The focus of this work is only on nesting that does not introduce redundancy, however, such nesting introduces an interesting tradeoff between query/update performance versus space efficiency.

4.1 Space Efficiency Metric

The metric chosen for this work is space efficiency. The space savings of nesting results from the fact that foreign keys originally necessary in the document to link concepts, can now be omitted as their relationship is encapsulated by the hierarchical nesting of elements. For example, in TPC-H[1] (the TPC-H schema is in Figure 1), order information can be nested under customer information, and the foreign key *orders.o_custkey* no longer needs to be represented explicitly:

```
<!ELEMENT customer (c_custkey, c_name, ..., orders*)>
<!ELEMENT orders (o_orderkey, o_orderdate, ...)>
```

One challenge with nesting is that the foreign key must be non-nullable. In the previous example, if *orders.o_custkey* could be `null`, then orders without a customer would not be mapped into the final document. This could be resolved by introducing a dummy customer record with value of *custkey* = `null`, but that is often not desirable. Initially, we will consider only non-nullable foreign keys, and then in Section 5.1 we will discuss how to handle `null` foreign keys.

It is possible to quantify space savings via nesting on foreign keys. The space saved via nesting is directly proportional to the size of the foreign key and the number of tuples nested. Given two relations R and S, we will define a function that returns the amount of space saved by nesting R under S. Let $FK_R(S)$ denote the attribute(s) in R that constitute the foreign key to the primary key of S (PK_S). Define the function $nest(R,S,K)$ as the nesting of relational schema R under S on the foreign key K. Let $|R|$ denote the number of tuples of R. The ***space savings*** of nesting relational schema R under S on key K is denoted by $savings(R,S,K)$ and is approximated by $sizeOf(K)*|R|$, where $sizeOf(K)$ is the maximum schema size of the foreign key K of R.

More precisely, the exact savings of nesting in XML depends on the size of the data value of the foreign key, the tag size for the foreign key, and the XML overhead of tag specification. For attributes, the XML overhead is 4 characters (2 for quotation marks, 1 for "=", and 1 for the space separating the attribute tag name). For elements, the XML overhead is 5 characters (2 for "<", 2 for ">", and 1 for "/"). The tag size we will assume to

[1] http://www.tpc.org/tpch/spec/h130.pdf

be the length of the attribute name for the foreign key in the database. For element encoding, the tag name is used twice. Estimating the character size of a data item is difficult. In the worst case, it is the maximum schema size of the element. This leads to the following estimates for attribute and element space savings of nesting as given in Figure 2. K is the foreign key in R, and $len(K)$ is the length of the attribute name for K.

Encoding Type	Space Savings
Attribute	$(1/2*sizeOf(K)*len(K)+4)*\|R\|$
Element	$(1/2*sizeOf(K)+2*len(K)+5)*\|R\|$

Figure 2. Space Savings Formulas for Nesting

In practice the difference between attribute and element encoding is less important as the major savings is O(R), and the rest of the formula is simply a constant factor. Savings in terms of data values not in the XML document (without worrying about their model representation) is $|R|$*(# of foreign key attributes).

4.2 Nesting Graph

Using the *savings* function and extracting database schema information on primary and foreign keys, it is possible to build a graph representing the nesting possibilities and their desirability. All information necessary for calculating the *savings* function can be automatically extracted from the schema and queries counting the number of tuples in each relation.

A *Nest-Graph G = (V,E)* is a directed, weighted graph consisting of a set of nodes V and a directed edge set E, such that for each relation R in the database schema, there exists a node $V_R \in V$, and for each non-nullable foreign key $FK_R(S)$ there is an edge $e=(V_S,V_R,w) \in E$, where $w=savings(R,S,FK_R(S))$. The Nest-Graph for the TPC-H schema is in Figure 3. All primary and foreign keys occupy 4 characters of space, and the edge weights are $savings(R,S,K)=sizeOf(K)*|R|*(\#\ of\ FK\ attributes)$. Relation sizes are given in parentheses under their respective nodes in the graph.

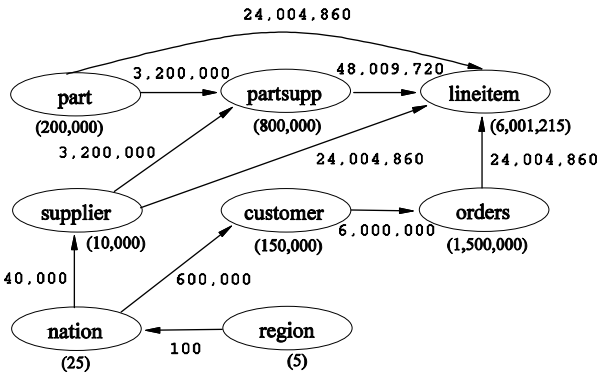

Figure 3. Nest-Graph for TPC-H Database Schema

5. XML MAPPING ALGORITHM

The mapping from the relational model to an XML DTD or XMLSchema has these general steps:

- Extract relational schema information including foreign keys and relation sizes from the database.
- Use the information to build a Nest-Graph G.
- Use the classical algorithm by Edmonds [6] to calculate the maximum weight arborescence T of G.
- The maximum weight arborescence is a maximal spanning tree (MST) T. Use T to generate the resulting nesting in the output XML schema. Given an edge $e=(S,R,w)$ of T, the two corresponding relations are $S(A_1,A_2,...,A_m)$ and $R(B_1,B_2,...,B_n)$ respectively. Let $A_1=PK_S$, $B_1=PK_R$, and $B_k=FK_R(S)$. For illustration, the primary keys and foreign keys are encoded as XML attributes, and all other relational attributes are encoded as XML elements.
  - For each edge $e=(S,R,w)$ of T, we will nest R under S (and omit B_k) in the XML DTD such as:

```
<!ELEMENT S (A_2, ..., A_m, R*>
  <!ATTLIST S A_1 ID>
<!ELEMENT R (B_2, B_3,..., B_k-1, B_k+1,...B_n>
  <!ATTLIST R B_1 ID>
```

  - For each edge $e=(S,R,w)$ of T where $e \in G$ *and* $e \notin T$, this relationship will be captured using ID/IDREF. Under the element R will be a IDREF attribute like:

```
<!ELEMENT R (B_2, B_3,..., B_k-1, B_k+1,..., B_n>
<!ATTLIST R B_k IDREF>
```

Edmonds' algorithm [6] is used to calculate the maximum weight arborescence T of a graph G. Note that there is a more efficient algorithm by Gabow *et al.* [9] for this problem that has cost O($|V|log|V|+|E|$), where $|V|$ is the number of nodes and $|E|$ is the number of edges. Given a directed, weighted graph G the algorithm will return the maximum spanning tree T of G if one exists. For our problem, the algorithm will select the nestings (edges) of maximum benefit. The tree constructed mimics the hierarchical model construct available in XML. Edmonds' algorithm cannot be directly applied to a Nest-Graph as it only returns a MST if one exists. A MST in a Nest-Graph will only exist if there is only one node with no incoming edges, and all other nodes are reachable from that node. Thus, we introduce a new node r' to G, and a set of edges E' where each $e=(r',V_i,0) \in E'$ for all i=1..$|V|$. This new node and edges connecting it to all others nodes with zero weight will guarantee that a MST will always be found, and the edges of zero weight will have no effect on the calculation of the optimal nesting.

Application of Edmonds' algorithm to the Nest-Graph for TPC-H produces the MST *T* given in Figure 4. By summing up the edges in *T*, we can calculate the space savings by introducing nesting in our XML schema. For TPC-H at scale factor 1, the space savings is 57,849,820 characters or 14,462,455 data values. Equivalently, the savings is 43.5% of all foreign keys or 12.4% of all data values. The DTD is given in Figure 5.

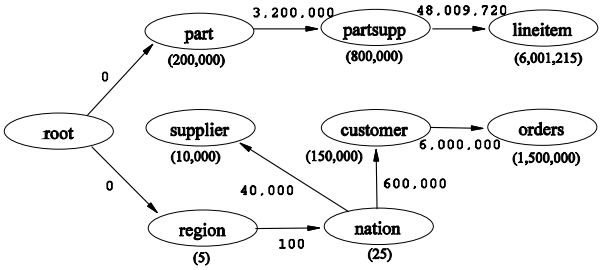

Figure 4. Maximum Spanning Tree for TPC-H

```
<!ELEMENT root (part*, region*)>
<!ELEMENT part (p_name, … , p_retailprice, partsupp*)>
   <!ATTLIST part p_partkey ID>
<!ELEMENT region (r_name, r_comment, nation*)>
   <!ATTLIST region r_regionkey ID>
<!ELEMENT partsupp (ps_availqty, … , lineitem*)>
   <!ATTLIST partsupp ps_suppkey IDREF>
<!ELEMENT nation (n_name, … , customer*, supplier*)>
   <!ATTLIST nation n_nationkey ID>
<!ELEMENT lineitem (l_linenumber, …, l_comment)>
   <!ATTLIST lineitem l_orderkey IDREF>
<!ELEMENT customer(c_name, … , c_comment, orders*)>
   <!ATTLIST customer c_custkey ID>
<!ELEMENT supplier (s_name, …, s_acctbal)>
   <!ATTLIST supplier s_suppkey ID>
<!ELEMENT orders (o_orderdate, … , o_comment)>
   <!ATTLIST orders o_orderkey ID>
```

Figure 5. Generated XML DTD for TPC-H Database

5.1 User-Directed Mapping

The presented algorithm determines the optimal nesting of relational schemas into an XML schema given no constraints. However, the optimal nesting in terms of space efficiency is not always the optimal nesting in terms of usability, readability, or other metrics. For this reason, it is useful for the user to be able to specify constraints on the XML schema that should be satisfied without having to specify a total mapping to the system. In this case, the system must construct the optimal, space efficient XML schema that respects the user constraints. Constraints can be incorporated into the mapping algorithm by appropriate modifications to the Nest-Graph. We examine four types of constraints:

1) Specification of non-nested nodes
2) Specification of edges (nestings) that must be present in the XML schema
3) Specification of edges (nestings) that should not be present in the XML schema
4) Handling nullable foreign keys

To specify such constraints, the user could potentially manipulate a graphical view of the Nest-Graph, but how the constraints are specified is not our focus.

The first constraint type is the specification of non-nested nodes. The user can specify that any node V_i is a non-nested node, even nodes that have incoming edges. If V_i has incoming edges, then the incoming edges of V_i are removed. This will cause V_i to be a non-nested node when the Edmonds' algorithm is run, and R_i will not be nested in the XML schema.

Constraints on edges are valuable because they allow the user to control the nesting process at a fine degree of precision. In many cases, the user is not interested in all the nestings, but some nestings may be especially important to be present or not to be present in the XML schema. For example, although there is a minor savings by nesting `customer` and `supplier` information under `nation` and `region` in TPC-H, this savings may not be enough to justify organizing the information in this way. Another example is that it is almost always more preferable for `lineitem` information to be nested under the corresponding `order`, even though more savings is achieved by nesting under `partsupp`.

Edges (nestings) that must be present in the XML schema are guaranteed by setting the edge weights to ∞. A nesting that must not be in the schema are insured by removing the edges from the graph. The algorithm will then determine the optimal mapping given the user specified constraints. Thus, the major benefit of this mapping method is that the user can specify as many or as few constraints on the XML schema, and the system will compute the optimal solution given the constraints.

Foreign keys that may be `null` can be handled by giving the user an option on their encoding. The user can select if nullable foreign keys should be always encoded (edges are present in Nest-Graph) or not encoded (edges are not present in Nest-Graph). If a foreign key may be `null` and it is nested, then when data translation is performed the system must insert a dummy `null` record in the parent element.

Finally, the algorithm can be generalized to accept any arbitrary cost function, including those that factor in query performance as well as space efficiency.

6. EXPERIMENTAL RESULTS

A Java program that uses JDBC to extract schema information including foreign keys was constructed to test the conversion algorithm. The program connects to a database using JDBC, extracts schema information, gathers relation size statistics, then uses that information to build a Nest-Graph. The Nest-Graph is transformed into a form compatible with Edmonds' algorithm, and the algorithm is run to find a MST. Then, the MST is converted into an XML DTD. The savings with respect to the number of data values not encoded are given in Figure 6.[2] Unlike the CoT algorithm [12], our algorithm is always guaranteed to find the optimal nesting.

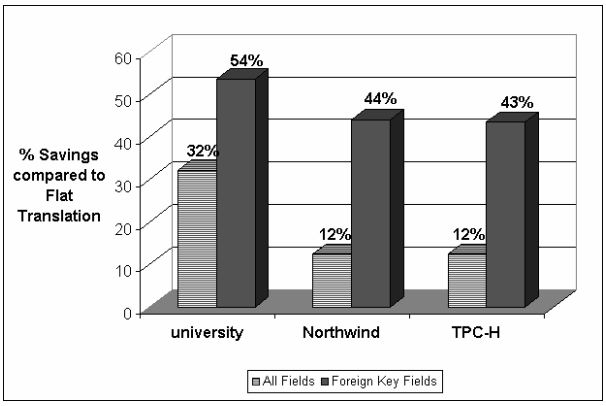

Figure 6. Percentage of Data Values Not Encoded

As can be seen in the results of encoding entire databases in XML, a significant number (40-50%) of foreign keys do not have to be encoded. When migrating large database instances to XML, this results in a significant savings. Elimination of all foreign keys is only possible if the Nest-Graph is a tree. Even when considering the total number of attributes, eliminating between 10 to 30% of all attributes using nesting is significant. The nesting has the added advantage of improving the readability of the data and reducing the number of joins required to connect related data.

7. FUTURE WORK AND CONCLUSIONS

Given the growing importance of representing data in XML format and its common preexistence in relational databases, it is important to have an efficient and user-friendly tool for automating construction of XML schemas from relational database schemas. This work describes a mapping tool that uses a space efficiency cost metric and user constraints to generate optimal schemas. It is an improvement over current approaches which either do not have formal methods for capturing user constraints (CoT [12]) or require the user to exactly specify the entire XML schema in the form of an extraction query [8]. The space efficiency cost-metric proposed is useful, although other metrics are possible including those that factor in query costs as well. The algorithm is the first mapping method to incorporate both automatic cost-based translation and user constraints.

Future work involves expanding the algorithm to consider cost functions and nestings that introduce redundancy to increase query performance at the sacrifice of space efficiency. Also, a more detailed investigation on the types of queries that can benefit from nesting will be performed.

[2] Northwind schema is from Microsoft Access. University schema was used to evaluate CoT.

References

[1] Arenas, M. and Libkin, L., "A Normal Form for XML Documents", *Proceedings of ACM PODS 2002*, pages 85-96.

[2] Banerjee, S., Krishnamurthy, V., Krishaprasad M., and Murthy, R., "Oracle 8i – The XML Enabled Data Management System", *Proceedings of ICDE 2000*.

[3] Bird, L., Goodchild, A., and Halpin, T., "Object Role Modelling and XML-Schema", *Proceedings of ER 2000*, pages 309-322.

[4] Bohannon, P., Freire, J., Roy, P., and Simeon, J., "From XML Schema to Relations: A Cost-Based Approach to XML Storage", *Proceedings of ICDE 200*2, pages 64-76.

[5] Du, W., Lee, M., and Ling, T., "XML Structures for Relational Data", *Proceedings of WISE 2001*, pages 151-160.

[6] Edmonds, J., "Optimum Branchings",*Journal of Research of the National Bureau of Standards*, 71B:233-240, 1967.

[7] Embley, D., and Mok, W., "Developing XML Documents with Guaranteed 'Good' Properties", *Proceedings of ER 2001*, pages 426-441.

[8] Fernandez, M., Kadiyska, Y., Suciu, D., Morishima, A., and Tan, W., "SilkRoute: A Framework for Publishing Relational Data in XML", *ACM TODS*, 27(4):438-493, 2002.

[9] Gabow, H., Galil, Z., Spence, T., and Tarjan, R., "Efficient Algorithms for finding minimum spanning trees in undirected and directed graphs", *Combinatorica*, 6(2):109-122, 1986.

[10] IBM DB2 XML Extender, 2002, http://www-3.ibm.com/software/data/db2/extenders/xmlext.

[11] Klettke, M., Schneider, L., and Heuer, A., "Metrics for XML Document Collections", *EDBT 2002 Workshops*, pages 15-28.

[12] Lee, D., Mani, M., Chiu, F., and Chu, W., "NeT & CoT: Translating relational schemas to XML schemas using semantic constraints", *Proc. of CIKM 2002*, pages 282-291.

[13] Rys, M., "State-of-the-Art XML Support in RDMBS: Microsoft SQL Server's XML Features", *IEEE Data Engineering Bulletin*, 24(2):3-11, 2001.

[14] Shanmugasundaram, J., Shekita, E., Barr, R., Carey, M., Lindsay, B., Pirahesh, H., and Reinwald, B., "Efficiently Publishing Relational Data as XML Documents", *Proceedings of VLDB 2000*, pages 65-76.

[15] Tolani, P. and Haritsa, J., "XGRIND: A query-friendly XML compressor", *Proceedings of ICDE 2002*.