ALGORITHMS AND IMPLEMENTATION OF FUNCTIONAL DEPENDENCY DISCOVERY IN XML

A thesis presented in partial fulfilment of the requirements for the degree of

MASTER OF INFORMATION SCIENCES

IN

INFORMATION SYSTEMS

at Massey University, Palmerston North, New Zealand

Zheng Zhou

May 2006

Acknowledgement

I wish to take this opportunity to give my sincere thanks to Associate Professor, Dr. Sven Hartmann for his kind and enlightening guidance and constructive suggestions throughout this research and valuable recommendations on improvement of the thesis.

It has been my pleasure to meet those wonderful people at the University and I have developed a solid friendship with many of them; without their support in varied forms, it would have been an arduous year for me. Particularly, I am thankful to Madre Chrystall for her inspiring encouragement, valuable technical advice on MySQL and kind proofreading. I also benefited from discussions with Thu Trinh who shared with me her knowledge on Latex. Furthermore, the Massey Masterate Scholarship which I was awarded in 2005 academic year has made my life financially manageable.

Finally, I owe my entire life and all my success to my parents for their incomparable enormous *love*; they are appreciated, respected and treasured by me forever and a day. My heart is always warmed also by the fondness from my aunt and my sister. They light up my life and are my everything.

Zheng Zhou
9 May 2006

Contents

Chapter 1

Introduction

1.1 Background

Following the advent of the web, there has been a great demand for data interchange between applications using internet infrastructure. XML (eXtensible Markup Language) provides a structured representation of data empowered by broad adoption and easy deployment. As a subset of SGML (Standard Generalized MarkupLanguage), XML has been standardized by the World Wide Web Consortium (W3C) [Bray et al., 2004]. XML is becoming the prevalent data exchange format on the World Wide Web and increasingly significant in storing semi-structured data. After its initial release in 1996, it has evolved and been applied extensively in all fields where the exchange of structured documents in electronic form is required.

As with the growing popularity of XML, the issue of functional dependency in XML has recently received well deserved attention. The driving force for the study of dependencies in XML is it is as crucial to XML schema design, as to relational database(RDB) design [Abiteboul et al., 1995].

1.2 Motivations

As semi-structured data has become prevalent with the growth of the Internet and other on-line information repositories, many organisational databases are presented on the web as semi-structured data. Designing a 'good' semi-structured database is increasingly crucial to sustain data integrity and prevent data redundancy, inconsistency and updating anomalies. Redundant information caused by functional dependencies in XML may give rise to such problems. Therefore, identifying XML functional dependencies and thus achieving normalisation becomes vital in good XML design.

Often in design practice, we are facing a task of finding all possible functional dependencies satisfied by a given XML document, which may imply business rules. Thus emerges a new research direction: the *XML dependency discovery* problem, on which however, little investigation has been conducted so far though a breakthrough would be of prominent value in practice.

XML schema plays a substantial role in discovering functional dependencies of XML data, since they are defined on top of schematic information, as with relational databases.

In addition, it is well-known that XML schema information specifies the internal structure of an XML document, which realises the promise of XML as the universal data representation format enabling free electronic data interchange (EDI) and integration of disparate data sources. It is also critical in the efficient storage of XML data as well as formulation, optimisation and query processing [Garofalakis et al., 2000]. Unfortunately, in practice many XML documents are not associated with schema definitions, giving rise to the task of inferring the schematic information from XML documents.

Our preliminary feasibility studies on XML dependency discovery have suggested the 'divide-and-conquer' strategy, leading to the following problem decomposition:

1. **XML Schema Extraction**

   This is determined by the fact explained previously that XML schema information is essential but absent in most cases. Certain generalisation of input data is often required in schema extraction; ideally the extracted schema should, on one hand, tightly represent the data, and be concise and compact on the other hand. As the two requirements essentially contradict each other, finding an optimal tradeoff is a difficult and challenging task [Chidlovskii, 2001].

2. **XML-Relation Data Transformation**

   Next, semantic data held in the original XML instance is extracted into a tabular format with the help of its schema. The inspiration for such a transformation comes from appreciation of over 20 years of work invested in relational database technology. Relational functional dependencies have been well explored and some inference algorithms with satisfactory performance are already in existence, which we can leverage to assist in discovering functional dependencies in XML. There have been some research endeavours on mapping XML documents to relational tables, as further illustrated in the section 'Related Work'.

3. **Relational FD Inference**

The final step is to apply some well-developed relational functional dependency inference algorithms to the data in relational format after the transformation to achieve our ultimate goal.

Figure 1.1 shows the entire work flow:

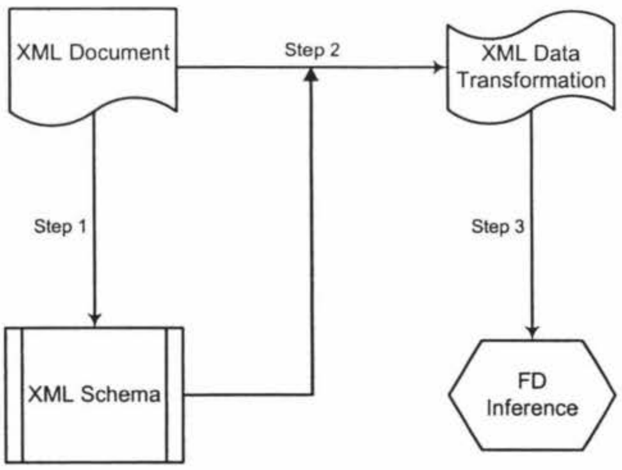


Figure 1.1: High Level Work Flow

1.3 Related Work

1.3.1 Hypothesis Research

The inference of structure out of semi-structured data has been long-standing in the XML research area [Sakakibara, 1997]. Some approaches investigated possible solutions derived from theoretical grammatical inference and were very powerful at the conceptual level. [Ahonen, 1996] presented a technique based on machine learning, with the help of finite-state automata describing the given instances completely. These automata were modified by considering certain context conditions, corresponding to generalisation of the underlying language, which were then converted into regular expressions to construct the grammar. Although traditional grammatical inference methods for DTD generation stated in [Ahonen, 1996] are theoretically appealing (as they guarantee to infer languages falling within certain language classes), it is not clear whether the structure within the limited context is valid in practice, i.e., its theoretical appeal may not necessarily translate into practical applicability.

[Young-Lai, 1996] discussed a grammatical inference method generating stochastic finite automata using an adapted stochastic method and attempted to improve it by isolating low frequency data components and allowing adjustment at the generalisation level. This approach was derived from more recent work in grammatical inference, with the base algorithm known as Alergia [Carrasco and Oncina, 1994]. As with the methods of Ahonen, Alergia has strong theoretical significance. Again, though, our interest lies in practical performance. Moreover, none of them even touched the problem of how to present schema information with high understandability to the user.

[Garofalakis et al., 2000] proposed XTRACT, a specialized DTD induction system consisting of a generation module, a factoring module and an MDL[1] (Minimum Description Length) module. XTRACT employed generalisation and factorisation of regular expressions, to derive a pool of candidate DTDs, and then used the MDL principle as the basis to make a final selection. Still, XTRACT requires human intervention and judgement in making a choice out of all candidates.

[1]The reader of interest in MDL is referred to [Rissanen, 1978, Rissanen, 1989] for further details.

1.3.2 Schema Derivation Algorithms

Some research focusing on XML practice has also been going on, mainly centring on DTD and schema tree extraction. [Chen, 1991] talked about generation of '*de-facto grammar*', which simply aggregated structures of all XML instances to be the DTD. The *de-facto grammar* is obviously far too simple and limited.

[Chidlovskii, 2002] modelled the XML schema as extended context-free grammars and developed an extraction algorithm inspired by methods of grammatical inference. The algorithm was also said to cope with the schema determinism requirement imposed by XML DTDs and XML Schema languages. He defined *(range) Extended Context Free Grammar* (ECFG) as a 5-tuple $G = (T, N, D, \delta, Start)$, where T, N and D are disjoints set of terminals, non-terminals and datatypes; $Start$ is an initial non-terminal and δ is a finite set of production rules. The rules take the form $A \rightarrow a$ for $A \in N$, where a is a range regular expression over terms, and one term is a terminal-nonterminal-terminal sequence like $t\ B\ t'$, briefly $t : B$, where $t, t' \in T$ and $B \in N \cup D$. The extraction algorithm firstly generalized ECFG from XML content, which was then transformed to an XML schema definition. Details of the algorithm are shown in Figure 1.2:

```
0. Represent XML documents as set $I$ of structured examples.
1. Induce an extended context-free grammar $G$ from $I$:
    1.1  Create the initial set of nonterminals $N$:
    1.2  Merge nonterminals in $N$ with the similar content and context;
    1.3  Determine tight datatypes for terminals in $G$;
    1.4  Generalize contents in nonterminals into range REs.
2. Transform the result ECFG $G$ into an XML Schema definition $S$.
```

Figure 1.2: ECFG Extraction Algorithm (Adapted from [Chidlovskii 2002, p. 292])

In addition to work concerned with the problem of DTD inference, there have also been many papers published on related topics. Most notable amongst these is work within the Lore semistructured database project to infer DataGuides [Goldman and Widom, 1997]. This included the MakeDataGuide algorithm to construct a strong DataGuide over a source database as shown in Figure 1.3 — A DataGuide is strong *iff* it shares exactly the same set of label paths as in the source, nothing more or less. Despite its simplicity and high understandability at the conceptual level, the algorithm does not even mention any technical aspects, such as data structure, i.e., how the schematic information can be actually stored and utilised.

```
// Input: o, the oid of the root of a source database
// Effect: dg is set to be the root of a strong DataGuide for o
targetHash = global empty hash table, to map source target sets to DataGuide
objects
dg = global oid
MakeDataGuide(o) {
    dg = NewObject()
    targetHash.Insert(o, dg)
    RecursiveMake(o, dg)
}
RecursiveMake(t1, d1) {
    p = set of <label, oid> children pairs of each object in t1
    foreach (unique label l in p) {
        t2 = set of oids paired with l in p
        d2 = targetHash.Lookup(t2)
        if (d2 != nil) {
            add an edge from d1 to d2 with label l
        } else {
            d2 = NewObject()
            targetHash.Insert(t2, d2)
            add an edge from d1 to d2 with label l
            RecursiveMake(t2, d2)
        }
    }
}
```

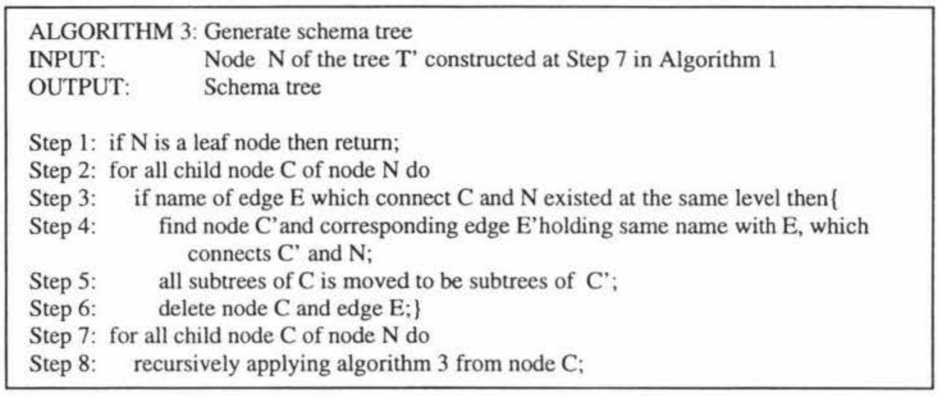
Figure 1.3: Algorithm MakeDataGuide (Adapted from [Goldman et al. 1997 , Figure 4, p. 8])

1.3.3 Schema Tree Inference Algorithms

The subject of schema tree and related issues, such as tree extraction, are not recent in research area. A labelled tree specifying nesting relationships between labelled vertices was referred to as XML schema tree elements [Cruz et al., 2004]. A schema tree was also defined as an ordered tree representing the XML schema in terms of a set of constructors:

sequence (','), *repetition* ('*'), *union* ('|'), < *tagname* > (corresponding to a tag) and < *simpletype* > (corresponding to base types) [Ramanath et al., 2003].

[Chen et al., 2002] stated a schema tree generation algorithm as displayed below:



```
ALGORITHM 3: Generate schema tree
INPUT:       Node N of the tree T' constructed at Step 7 in Algorithm 1
OUTPUT:      Schema tree

Step 1: if N is a leaf node then return;
Step 2: for all child node C of node N do
Step 3:     if name of edge E which connect C and N existed at the same level then{
Step 4:         find node C'and corresponding edge E'holding same name with E, which
                    connects C' and N;
Step 5:         all subtrees of C is moved to be subtrees of C';
Step 6:         delete node C and edge E;}
Step 7: for all child node C of node N do
Step 8:     recursively applying algorithm 3 from node C;
```

Fig.9: Generate schema tree algorithm

Figure 1.4: Schema Tree Generation (Adapted from [Chen et al. 2002 , Figure 9, p.84])

There are at least three deficits in this algorithm: firstly, it only considers, compares and processes identical elements appearing at the same level (in Step 3), exclusive of the scenario with one element occurring at different levels. Secondly, it just simply aggregates subtrees of all occurrences of an element (node) (in Step 5), which will merely give a document tree at most, instead of a schema tree as supposed. Third, structural information captured is rather poor; only a collection of possible sub-element names, without any knowledge of element order, whether they are optional, compulsory, or iterating.

1.3.4 XML-RDB Mapping

Researchers have already shown their interest in transforming data in XML format into relational database. [Christophides et al., 1994] proposed a one-to-one mapping from each element declaration in the DTD to a relation. It is apparently a simple way of generating corresponding relational schema but likely leads to excessive fragmentation.

[Shanmugasundaram et al., 1999] suggested analyse a DTD and automatically convert it to a set of relational schemata. To do this, the original DTD should be firstly simplified by discarding element order information before generating the final relation schema:

- **Basic approach** Generate a DTD graph after grouping or flattening element frequency specifications and the respective element graph on which the relational schemata are decided;
- **Shared approach** Create a separate relation for each element node represented by multiple relations in the basic approach, and share this relation; or
- **Hybrid approach** Same as the shared approach except for some variance in element processing.

Their work will also result in excessive fragmentation of DTDs, causing unnecessary data scatter, which incurs unaffordable cost from joins when multiple relations need to be accessed.

A new inlining algorithm was put forward by [Lu et al., 2003], featuring modeling XML attributes as XML elements since they can be treated as elements without further nesting structure. It comprises similar steps as the others: Create a DTD graph after DTD simplification and inline as many descendant elements as possible to an XML element to eliminate redundancy caused by shared elements in the generated schema, which is to be eventually generated based on the inlined DTD graph. Such an inlining algorithm can relatively reduce redundancy in comparison to the shared approach introduced previously, though data scatter is still present.

[Yan and Fu, 2001] described construction of schema prototype trees representing the structure of a simplified DTD and subsequent generation of relational schema prototypes. They also briefly mentioned functional dependency and candidate key detection and relational schema prototype normalisation techniques.

In a summary of relevant literature, most of them are of little practical significance and applicability as their studies are no more than academic research, although their complexity varies. Furthermore, none of them even addressed how to render the schema information to the user. Nearly all research on XML-RDB mapping is somewhat a schema-awared approach, requiring the existence of the DTD to operate, which most XML documents in practice lack. Excessive data fragmentation is another common awkwardness; we end up with an unmanageably large number of relations, complicating the situation since many functional dependencies may span several relations.

1.4 Our Contribution

In our research, we delved into both schema extraction and the XML-Relation data transformation problem. A novel data representation model, ER-XML (Element Relationship model for XML) was devised, utilising an implementation-focused algorithm capable of being directly applied to XML practice. ER-XML can also help to extract and identify cardinality constraints. The data structure invented was properly designed to facilitate graphical representation generation as well as compatibility validation. As for XML-Relation transformation, we have developed an entire set of algorithms, *SVT-Trans* with the help of ER-XML and the concept of 'Almost Copy' in XML tree, which retrieves semantic data from an original XML document and places them into a relational format using recursion computation. The output of SVT-Trans can be directly exploited by relational functional dependency discovery algorithms. A prototype system was also successfully implemented and a case study was provided which demonstrated correctness and soundness of our work.

1.5 Thesis Outline

Following the introduction and review of related work, Chapter 2 of the thesis presents some essential XML and affiliated technology. We investigate XML schema extraction and illustrate the ER-XML model and the ER-XML Extraction (EXE) algorithm in Chapter 3. Chapter 4 discusses in detail SVT-Trans designed to convert XML data into a relation. Some preliminary definitions are also covered. Design decisions and architecture of the prototype system, XFD-Miner, unfold in Chapter 5, with a case study in Chapter 6. Finally, Chapter 7 summarises our work and points out future research directions.