Information Systems 35 (2010) 804–824

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Extracting a largest redundancy-free XML storage structure from an acyclic hypergraph in polynomial time

Wai Yin Mok[a,1], Joseph Fong[b,*], David W. Embley[c]

[a] Department of Economics and Information Systems, University of Alabama in Huntsville, Huntsville, AL 35899, United States
[b] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[c] Department of Computer Science, Brigham Young University, Provo, UT 84602, United States

ARTICLE INFO

Article history:
Received 25 May 2009
Received in revised form
18 December 2009
Accepted 22 April 2010

Keywords:
XML data redundancy
Large XML storage structures
XML-Schema generation
Acyclic hypergraphs

ABSTRACT

Given a hypergraph and a set of embedded functional dependencies, we investigate the problem of determining the conditions under which we can efficiently generate redundancy-free XML storage structures with as few scheme trees as possible. Redundancy-free XML structures guarantee both economy in storage space and the absence of update anomalies, and having the least number of scheme trees requires the fewest number of joins to navigate among the data elements. We know that the general problem is intractable. The problem may still be intractable even when the hypergraph is acyclic and each hyperedge is in Boyce–Codd normal form (BCNF). As we show here, however, given an acyclic hypergraph with each hyperedge in BCNF, a polynomial-time algorithm exists that generates a largest possible redundancy-free XML storage structure. Successively generating largest possible scheme trees from among hyperedges not already included in generated scheme trees constitutes a reasonable heuristic for finding the fewest possible scheme trees. For many practical cases, this heuristic finds the set of redundancy-free XML storage structures with the fewest number of scheme trees. In addition to a correctness proof and a complexity analysis showing that the algorithm is polynomial, we also give experimental results over randomly generated but appropriately constrained hypergraphs showing empirically that the algorithm is indeed polynomial.

1. Introduction

XML databases are emerging [4]. Two types of XML databases are native XML databases and XML-enabled databases. The fundamental unit of (logical) storage in native XML databases is an XML document [3]. Thus, designing XML documents for efficient retrieval and update has been a topic of recent research [8–10]. The fundamental unit of (logical) storage in XML-enabled databases is a relational table. This table-storage method requires various mapping rules to translate between XML document schemas and database schemas and employs middleware to transfer data between XML documents and databases [3,17,22]. A recent study shows that designing XML documents for efficient retrieval and update can also guarantee well-designed relational storage structures for XML-enabled databases [11]. Thus, for both native XML databases and XML-enabled databases, designing XML documents for efficient retrieval and update is an appropriate focus for study.

Similar to designing relational tables by normalizing relational schemas, designing XML documents for efficient retrieval and update is about normalizing XML storage schemas. Normalized XML storage schemas remove the

* Corresponding author.
E-mail addresses: mokw@email.uah.edu (W.Y. Mok), csjfong@cityu.edu.hk (J. Fong), embley@cs.byu.edu (D.W. Embley).
[1] Most of this research was conducted while W.Y. Mok was a Visiting Research Fellow at City University of Hong Kong.

0306-4379/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2010.04.004

possibility of redundancy with respect to constraints and typically make both retrieval and update more efficient. Thus, there has been a flurry of research work on normalization of XML documents [1,5,6,13,15,21,24–26].

This paper, which follows up on our previous work [6,15], is another step in this direction. Like [15], instead of generating XML DTDs or XML Schema specifications directly, we first generate XML storage structures. These storage structures, called scheme trees here and elsewhere [16], are simply generic hierarchical structures. After obtaining a set of scheme trees, we can apply the mapping method in [1], or equivalently, those cited in [15], to generate a DTD or the basic structural components of an XML Schema document. These mappings simply represent scheme trees syntactically in these XML specification schemes in a one-to-one correspondence. Therefore, under these mappings, there is redundancy in a scheme-tree instance if and only if there is redundancy in an XML document. Hence, our discussion in this paper only needs to focus on scheme trees and scheme-tree instances, without concern for the mapping to DTDs or to XML Schemas.

In [15] we showed that generating a minimum number of redundancy-free scheme trees from a conceptual-model hypergraph is NP-hard. Here we consider special-case conditions in an effort to find an efficient algorithm. First, we limit ourselves to regular hypergraphs [2,14], which are a special type of conceptual-model hypergraphs. Also, since it is known that checking whether relational schemas are in Boyce–Code normal form (BCNF) is intractable [12], we limit hypergraphs to those in which each hyperedge is in BCNF with respect to the given functional dependencies (FDs). Next, since cycles in hypergraphs introduce ambiguity and typically cause difficulties, we assume that hypergraphs are acyclic. Finally, we assume that the only multivalued dependencies (MVDs) are hypergraph-generated MVDs. Even with these assumptions, however, it is an open problem to find an algorithm that generates a minimum number of redundancy-free scheme trees in polynomial time. We therefore settle on a heuristic that resolves the issue for many practical cases and likely gives good results for all cases.

As the basis of our heuristic, we provide in this paper a polynomial-time algorithm that generates a largest scheme tree from an acyclic hypergraph and a set of FDs where each FD is embedded in some hyperedge and each hyperedge is in BCNF. As an approximation to generating a minimum number of redundancy-free scheme trees, we use this heuristic repeatedly on the remaining hyperedges not already included in generated scheme-tree storage structures. This heuristic always yields redundancy-free scheme trees and often, especially in practical cases, yields the fewest.

To illustrate our approach and to show some of the pitfalls involved, we present a motivating example. In this example, we rely on intuition for some undefined terms. Later in Section 2, we formally define these terms.

Example 1. Fig. 1(a) shows an acyclic hypergraph and an FD, *Retailer Item*$\rightarrow$*Price*, embedded in one of the hyperedges. Fig. 1(b) shows some possible relationships among instance values for the hyperedges in Fig. 1(a). For example, two of the relationships are “retailer r_1 sells item i_1 for \$3” and “manufacturer m_1 has factory f_1.” Figs. 2(a)–(c) show three possible sets of scheme trees and their associated instances taken from the relationships in Fig. 1(b). In Fig. 2(a), because there is only one scheme-tree instance, the data values are compactly stored. However, the instance data is redundant. Since manufacturer m_1 is necessarily stored twice, the dependent factories, which must be the same, are therefore redundantly stored more than once. In Fig. 2(b), even though no data redundancy is present in any of the scheme-tree instances, there are more trees than necessary. The largest redundancy-free scheme tree for this example is the one on the left in Fig. 2(c), which balances the requirements of data redundancy and compactness of data. Creating this scheme tree first followed by creating a scheme tree from the remaining hyperedge {*Manufacturer*, *Factory*} yields the fewest possible redundancy-free scheme trees.

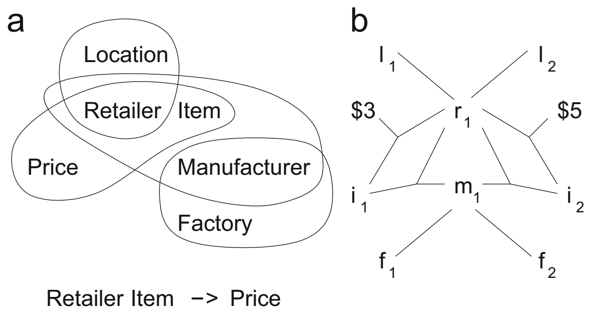

Fig. 1. The acyclic hypergraph and relationships of Example 1.

We give the details of our contribution of generating a largest possible scheme tree from an acyclic hypergraph in polynomial time as follows. We first lay the ground work by providing basic definitions in Section 2. Based on this foundation, we present the polynomial-time, scheme-tree generation algorithm in Section 3. Throughout Sections 2 and 3 we provide examples to motivate and illustrate definitions and algorithmic procedures. We present experimental data to verify our algorithm in Section 4 and formally prove our claims in Section 5. We make concluding remarks in Section 6.

2. Basic definitions

Since we limit ourselves to regular hypergraphs [2,14], we make the universal-relation-scheme assumption [20]. This is different from our previous work [15] for which we did not make such an assumption.

2.1. Acyclic hypergraphs

To make this paper self-contained, we borrow some definitions from previous work. The first four definitions are from [2].