Generating Compact Redundancy-Free XML Documents from Conceptual-Model Hypergraphs

Wai Yin Mok and David W. Embley

Abstract—As XML data becomes more and more prevalent and as larger quantities of data find their way into XML documents, the need for quality XML data organization will only increase. One standard way of structuring data well is to reduce and, if possible, eliminate redundancy, while at the same time making the storage structures as compact as possible. In this paper, we present a methodology to generate XML storage structures where conforming XML documents are redundancy-free, and for most practical cases, are also fully compact. Our methodology assumes the input is a conceptual-model hypergraph. For the special case that every edge in the hypergraph is binary, we present a simple algorithm, guaranteed to always generate redundancy-free storage structures. We show, however, that generating a minimum number of redundancy-free storage structures is NP-hard. We therefore provide heuristics to guide the process and observe that these heuristics result in satisfactory solutions, which are often optimal. We then present a general algorithm for n-ary edges and show that it generates redundancy-free storage structures. The general algorithm must overcome several problems that do not arise in the special case.

Index Terms—XML data redundancy, compact XML storage structures, XML scheme generation.

1 INTRODUCTION

In this paper, we present a methodology to generate XML storage structures where conforming XML documents are guaranteed to be redundancy-free with respect to functional and multivalued application constraints. Furthermore, for most practical cases, conforming XML documents are also as compact as possible. We assume that the XML storage structures produced are for XML documents representing some aspects of the real world—those for which conceptual modeling makes sense. Under this assumption, we argue that a good way to produce compact, redundancy-free storage structures for an application is to first produce a conceptual-model instance for A and then apply transformations guaranteed to produce storage structures with these properties.

Our motivation for this approach is rooted in the observation that practitioners routinely use mechanical-generation algorithms to convert conceptual entity-relationship model instances to sets of functioning relational model storage structures [2], [5], [21]. Many of these algorithms guarantee that the results produced have desirable redundancy-free and compactness properties [9], [15]. We propose the same for XML when the XML storage structures are for semistructured, data-rich documents, where conceptual modeling makes sense.

Finding and validating automated XML structure-generation algorithms that have desirable redundancy-free and compactness properties is not a simple matter of straightforwardly generalizing the ER-to-relational-table conversion algorithms. For XML algorithms, we must take into account both the hierarchical characteristics of XML storage structures and the semistructured nature of the data. Furthermore, because of the additional degrees of freedom provided in XML storage-structure specifications, there are likely to be many more reasonable designs to consider. Our proposal lends itself nicely to this increase in degrees of freedom. Not only can it automatically generate suggested XML storage structures, but it can also continuously evaluate and interactively provide feedback to XML storage-structure designers as they develop XML storage structures based on conceptual model instances.

Several other research efforts are closely related to our work. Perhaps the closest, in the sense that it takes the same approach, is [4]. Indeed, the authors of [4] make the same argument we make, namely: 1) that graphical conceptual-modeling languages offer one of the best—if not the best—human-oriented way of describing an application, 2) that a model instance should be transformed automatically into XML storage structures, and 3) that the transformation should maximize connectivity among the XML components and should minimize the redundancy in conforming XML documents. The authors of [4] use Object Role Modeling [13] as their conceptual model and give a set of 12 heuristics for generating the “best” XML schema. What is missing is the guarantee that their transformation achieves maximum connectivity and minimum redundancy. In this paper, we use a more generic conceptual model and a simpler set of heuristics to achieve similar results, but the
main contribution is the guarantee that conforming XML documents are redundancy-free and in most practical cases, they are as compact as possible as well.1

Other researchers have taken a different approach, attempting to arrive at much the same destination—redundancy-free XML storage structures. The authors of RRXS [6] investigate the problem of how to design a normalized relational schema for XML data. In [1], [22], [23], the authors propose various redundancy-preventing normal forms for XML. We differ fundamentally from these efforts because we take our constraints from a conceptual model instance rather than from specified XML FDs and XML MVDs, which are defined in various ways over paths in XML storage structures. (We believe that it is easier for designers to specify and understand FD and MVD constraints in terms of conceptual models than in terms of XML storage structures. We therefore also proffer our methodology as a way to avoid having to specify these more complex, low-level constraints.)

We present our contribution as follows: Section 2 provides fundamental ideas, foundational definitions, and motivating examples. Besides arguing that we can produce XML compact storage structures that yield only redundancy-free XML documents, we also provide examples to show that even for simple applications, it is easy to produce (and, thus, nontrivial to avoid) XML storage structures that have redundancy and have more hierarchical clusters than necessary. In Section 3, we present a simple algorithm that generates redundancy-free storage structures and prove that it is correct. We prove in Section 4, however, that generating a minimum number of redundancy-free storage structures is NP-hard. We observe, however, that for many, if not most practical cases, the generation algorithm produces as few redundancy-free storage structures as possible. We present a more general algorithm in Section 5 and show what problems are encountered in the generalization and explain how these problems can be resolved. We conclude in Section 6 and briefly mention future research directions.

2 FUNDAMENTALS

2.1 Technical Motivational Preliminaries

Since the input of our investigation is a conceptual-model instance, we obtain the constraints of interest to us from the given conceptual-model instance. The constraints of particular interest are the functional and multivalued constraints imposed by the conceptual-model instance. Restricting ourselves to constraints obtainable from conceptual-model instances ties us to the real world and frees us from some of the nuances of theoretical database research. For example, it is well-known in related hypergraph research that there are MVDs that are not hypergraph-generated [3]. It nevertheless makes sense to restrict ourselves for some applications to only hypergraph-generated MVDs. In addition, we do not use FDs and MVDs of traditional relational database theory [16]. These are based on the universal-relation assumption [14], [20] that each attribute has a unique meaning and that there is a unique relationship between any set of attributes. This assumption is not generally true [14] and is certainly not true of many typical, real-world, conceptual-model instances, in which objects play multiple roles in various connecting relationships. Consequently, for our methodology, we define functional and multivalued constraints, which are counterparts of FDs and MVDs, but which are restricted to those that can be defined in a conceptual-model instance and to those where the universal-relation assumption does not have to hold.

Another feature of our methodology is that instead of generating XML DTDs or XML Schema specifications directly, we first generate generic storage structures. These generic structures, called scheme trees, are simply generic hierarchical structures [19]. After obtaining a set of scheme trees, we can apply the mapping method in [1], or equivalently, the one in [11], 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.2

Finally, we mention our prior work on Nested Normal Form (NNF) [17], [18], which provides the theoretical basis for our current efforts. As ascertained in [18], a scheme tree is in NNF if and only if all scheme-tree instances over the scheme tree are redundancy-free with respect to a given set of FDs and MVDs. Nevertheless, NNF does not directly apply to our work here since NNF is based on ordinary FDs and MVDs, which, in turn, are based on the universal-relation assumption. Rather, we apply the essence of NNF, which yields the algorithms we seek.

2.2 CM Hypergraphs

As a motivating example and as an example to illustrate fundamental ideas and definitions, consider the conceptual-model diagram in Fig. 1. Although based on [8], the conceptual modeling approach we present here is generic. Users model the real world by constraint-augmented hypergraphs, which we call CM hypergraphs (conceptual-model hypergraphs).

Definition 1. A hypergraph is a pair \((V, E)\), where \(V\) is a set of vertices and \(E\) is a set of edges. Each edge \(e_i \in E\) associates with exactly one multiset \((\{V_i(1 \leq i \leq |E|)\})\) of vertices in \(V\) whose cardinality is at least two. (If \(|V_i| = 2\), the edge is binary; if \(|V_i| = 3\), the edge is ternary; and, in general, if \(|V_i| = n\), the edge is \(n\)-ary.)

1. Reference [10] is an earlier and shorter version of this paper. In this current paper, we provide more elegant algorithms. We also provide a full set of definitions, theorems, lemmas, and proofs rather than just conjectures with short justifications.

2. By limiting the scope of our paper to this objective, we do not mean to imply that the transformation to DTDs or XML Schemas is trivial and not worthy of study. Indeed, we have already undertaken this study [11], and we have implemented transformations to both DTDs and XML schemas. In doing so, we have recognized that the richness of the problem deserves further study that may result in XML storage structures with even more pleasing characteristics.
Definition 2. A CM hypergraph is a hypergraph with the following additional properties:

1. Every vertex $v$ is an object set, whose elements denote objects of interest in the world being modeled.
2. Every edge $e$ is a relationship set, whose elements are relations over the objects in the object sets of the multiset of vertices of $e$.
3. Any edge $e$ may be directed, in which case, the multiset for $e$ is partitioned into two multisets, one for the tail(s) of the directed edge and one for the head(s).
4. Every vertex $v$ of every edge $e$ has either a “mandatory” or an “optional” declaration, which dictates whether the objects in $v$, respectively, must or may participate in the relationships in $e$.

Example 1. In CM hypergraph diagrams, object sets (vertices) appear as named rectangles. The object set Hobby in Fig. 1 is an example; it may, for instance, denote the set [Chess, Hiking, Sailing]. Relationship sets (edges) appear as lines connecting object sets. Directed edges representing functional relationships have arrowheads on the range side. In Fig. 1, a graduate student enrolls in only one program (e.g., PhD or MS but not both) and has only one faculty-member advisor. For every relationship set $R$ connected to an object set $S$, participation of objects in $S$ in relationships in $R$ is either optional or mandatory, denoted, respectively, by an “o” on the edge near the connection or the absence of an “o.” In Fig. 1, a faculty member need not have hobbies and need not have advisees, but must be in a department. In the general conceptual model, edges may be recursive (e.g., one faculty member may mentor another)—thus, the need for multisets. Further, relationship sets may connect more than two object sets, but we restrict ourselves initially to edges with just two connections.

2.4 Scheme-Tree Instances

Because scheme trees have a nested structure and because we allow multisets for attributes of tables in the nested structure, scheme-tree-instance definitions are nontrivial. Seen as nested tables, however, the simplicity returns. Fig. 3 shows a sample scheme-tree instance for the first scheme tree in Fig. 2 displayed as a nested table.

Definition 4. Let $U$ be a set of object sets. Let $\text{dom}(O)$ denote the domain of an object set $O \in U$. Let $T$ be a scheme tree over $U$. A scheme-tree instance over $T$ is recursively defined as follows:

1. If $T$ has only the root node $X = \{O_1, \ldots, O_m\}$, where $X$ is a nonempty multiset of object sets in $U$, then $r$ is a scheme-tree instance over $T$ if $r$ is a (possibly empty) set of partial functions $\{t_1, \ldots, t_m\}$ such that $t_i(O_j) \in \text{dom}(O_j)$.
2. If $T$ has more than one node, then let $T_1, \ldots, T_n$, $n \geq 1$, be the $n$ subtrees of $T$ such that each of the root nodes of the $T_i$s is a child node of the root node of $T$. If $\{t_1, \ldots, t_m\}$, $m \geq 0$, is the set of partial functions of the root node of $T$, then $r$ is a scheme-tree instance

(Grad Student, Program, Faculty Member, (Hobby)*)*

<table>
<thead>
<tr>
<th>Pat</th>
<th>PhD</th>
<th>Kelly</th>
<th>Hiking</th>
<th>Skiing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracy</td>
<td>MS</td>
<td>Kelly</td>
<td>Hiking</td>
<td>Sailing</td>
</tr>
<tr>
<td>Lynn</td>
<td>MS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3. Sample scheme-tree instance for the first scheme tree in Fig. 2.
over $T$ if $r = \bigcup_{i=1}^{m} (t_j \cup \bigcup_{i=1}^{m} s_j)$, where $s_j$ is the scheme-tree instance over $T_i$ for the $j$th partial function $t_j$.

**Example 3.** We can match the formal notation in the definition with the scheme-tree instance in Fig. 2 as follows: Let the set \{ $t_1, t_2, t_3$ \} be the three root-level partial functions. Thus, for $t_1$, $t_1(\text{Grad Student}) = \text{Pat}$, $t_1(\text{Program}) = \text{PhD}$, and $t_1(\text{Faculty Member}) = \text{Kelly}$. For $t_2$, $t_2(\text{Grad Student}) = \text{Tracy}$, $t_2(\text{Program}) = \text{MS}$, and $t_3(\text{Faculty Member}) = \text{Kelly}$; and for $t_3$,

$$t_3(\text{Grad Student}) = \text{Lynn}$$

and $t_3(\text{Program}) = \text{MS}$.

Further, each of these root-level partial functions has an associated scheme-tree instance for $\text{Hobby}$. For $t_2$, for example, we have $s_2$, denoting the scheme-tree instance for the first (and only) subtree of $t_2$. Recursively, the definition now lets us define the set of partial functions for $s_2$ as we defined the initial set of partial functions. Since there are two, we can let \{ $t'_1, t'_2$ \} be the two partial functions where $t'_1(\text{Hobby}) = \text{Hiking}$ and $t'_2(\text{Hobby}) = \text{Sailing}$. Unfolded, we can view $t_2$ in a different way:

$$\{(\text{Grad Student}, \text{Tracy}), (\text{Program}, \text{MS}), (\text{Faculty Member}, \text{Kelly}), (\text{Hobby}, \text{Hiking}), (\text{Hobby}, \text{Sailing})\}.$$ 

### 2.5 Syntactic and Semantic Covers

Returning to our main goal—generating “good” scheme-tree forests from CM hypergraphs—we first state the conditions required for a scheme-tree forest to represent a CM hypergraph. Certainly, as a minimum, every vertex and edge must appear in the scheme-tree forest. We capture this requirement with our definition of “syntactic coverage” (Definition 10, which in turn is based on several preliminary definitions—Definitions 5 through 9). Moreover, we must also be able to properly represent the data of the CM hypergraph as scheme-tree instances. We capture this requirement with our definition of “semantic coverage” (Definition 12, which is based on the definition of a valid, populated, conceptual-model instance—Definition 11).

**Definition 5.** Let $M$ be a CM hypergraph and $F$ be a scheme-tree forest. A vertex $V$ of $M$ appears in a scheme tree $T$ of $F$ if $V$ is in a node of $T$.

**Definition 6.** In a hypergraph diagram, an edge-vertex connection is the single point conjoining an edge and a vertex.

Edge-vertex connections are unique. Note, however, that because edges may be recursive and thus connect more than once to the same vertex, edge-vertex pairs are not necessarily unique. In a CM hypergraph, we may label the unique edge-vertex connections \{ $C_1, \ldots, C_{10}$ \}, where $n = \sum_{i=1}^{m} k_i$, for a hypergraph with $k_3$ 2-ary (binary) edges, $k_3$ 3-ary (ternary) edges, $\ldots$, $k_m$ $m$-ary edges, where $m$ is the maximum arity of any edge in the hypergraph.

**Definition 7.** A path of a scheme tree $T$ is a sequence of nodes from the root node of $T$ to a leaf node of $T$.

**Definition 8.** Let $M$ be a CM hypergraph and $F$ be a scheme-tree forest. An $n$-ary ($n \geq 2$) edge $E$ of $M$ is a partial function $E = \{(C_1, V_1), \ldots, (C_n, V_n)\}$ from the set of edge-vertex connections of $M$ to the set of vertices of $M$. An edge $E$ of $M$ appears in a scheme tree $T$ of $F$ if a (contiguous) subpath of a path of $T$ includes all $n$ occurrences of the vertices of $E$.

**Definition 9.** For a scheme tree $T$ with more than one edge included in a path $P$ of $T$, we inductively define the edges to be properly connected in $P$ and, thus, in $T$ as follows: An initial edge $E$ is properly connected in $P$ if $E$ appears in $P$ and at least one of $E$’s vertices is in the root node of $T$. An edge $E_2$ connected to an edge $E_1$ in $M$ on vertices $V_1, \ldots, V_k$ is properly connected in $P$ if 1) $E_1$ is properly connected in $P$, 2) $E_2$ appears in $P$, and 3) there are edge-vertex connections $C_1, \ldots, C_k$ of $E_2$, respectively, for vertices $V_1, \ldots, V_k$ of $P$ (i.e., $E_1$ and $E_2$ share their common vertices in $P$).

**Definition 10.** Let $M$ be a CM hypergraph. A scheme-tree forest $F$ syntactically covers $M$ if

1. the nodes in each scheme tree in $F$ consist only of object sets of $M$;
2. every vertex and edge of $M$ appears at least once in some scheme tree in $F$;
3. for every scheme tree $T$ in $F$, connected edges in $M$ appearing in the same path are properly connected in $T$;
4. if a node $N$ in a scheme tree of $F$ contains more than one vertex of $M$, every vertex of $N$ must associate with some other vertex in $N$ via an edge or edge component of $M$, and
5. every parent-child link in every scheme tree in $F$ must represent an edge or edge component of $M$ between at least one vertex in the parent node and at least one vertex in the child node.

**Example 4.** The scheme-tree forest in Fig. 2 syntactically covers the CM hypergraph in Fig. 1, and it is clear from the context how paths associate with edges. As an example to help understand the more subtle parts of the definition, consider Fig. 4 which contains a CM hypergraph, a tree that syntactically covers it, $T_{\text{valid}}$, and a tree that does not syntactically cover it, $T_{\text{invalid}}$. Written as partial functions from the set of edge-vertex connections \{ $C_1, \ldots, C_{10}$ \} to the set of object sets \{ $A, B, C, D$ \}, the edges of the CM hypergraph in Fig. 4 are

\[
\{(C_1, A), (C_2, B), (C_3, A), (C_4, B), (C_5, C), (C_6, C), (C_7, B), (C_8, B), (C_9, C), (C_{10}, D)\}.
\]

These edges are all properly connected in $T_{\text{valid}}$. Observe, however, that it is not clear from the context which $C$ is which

4. Definition 1 tells us that an edge associates with a multiset of vertices. Definition 8 refines Definition 1 by saying exactly how edges associate with vertices. Note that whereas the edge-vertex connections are unique (forming a true set), the vertices of an edge need not be unique (in general, forming a multiset).

5. If not clear from the context, we are obligated to designate how paths associate with edges in CM hypergraphs by specifying exactly how the vertices and their connections appear in a path of a scheme tree.
Definition 11. A populated CM hypergraph is valid if it satisfies every CM hypergraph constraint.

Definition 12. Let \( M \) be a valid, populated, CM hypergraph. Let \( F \) be a scheme-tree forest that syntactically covers \( M \). \( F \) semantically covers \( M \), or is valid with respect to \( M \), if there exists a scheme-tree instance for each scheme tree in \( F \), such that 1) an object \( o \) is in an object set \( O \) of \( M \) if and only if \( o \) appears as a value in some scheme-tree instance for \( O \), 2) a relation \( r \) is in a relationship set \( R \) of \( M \) if and only if \( r \) appears as a relationship in some scheme-tree instance for \( R \), and 3) two objects \( o_1 \) and \( o_2 \) are related by a sequence of relationships on edges \( E_1, \ldots, E_n \) of \( M \) if and only if \( o_1 \) and \( o_2 \) appear together (i.e., are in a tuple of the total unnesting of the scheme-tree instance for \( T \)).

Example 5. Consider a valid, populated, CM hypergraph \( M \) which has two edges \( AB \) and \( BC \) which respectively have the populated relations \( \{(1, 3), (2, 3)\} \) and \( \{(3, 4)\} \). Thus, 1, 3, and 4 are related in \( M \) as are 2, 3, and 4. The scheme-tree instance in Fig. 5a is valid with respect to \( M \). On the other hand, the scheme-tree instance in Fig. 5b is invalid with respect to \( M \) since 2 and 4 are related by a sequence of relationships in edges of \( M \) but 2 and 4 do not appear together in either of the unnested tuples \( \{(A, 1), (B, 3), (C, 4)\} \) or \( \{(A, 2), (B, 3), (C, \perp)\} \), which are the tuples in the total unnesting.

2.6 Redundancy-Free Scheme Trees

Having defined scheme trees, scheme-tree instances, and what it means for a scheme-tree instance to be valid with respect to a valid, populated, CM hypergraph, we are ready to define redundancy in scheme-tree instances.

Definition 13. Let \( U \) be a set of object sets. Let \( T \) be a scheme tree over \( U \) and let \( t \) be a scheme-tree instance over \( T \). Let \( t' \) be the scheme-tree instance obtained by replacing a data value \( v \) in \( t \) with some symbol, say \( \phi \), such that \( \phi \) is not a value in \( t \). The value \( v \) is redundant in \( t \) with respect to a constraint \( C \) that holds for \( T \) if \( C \) and the other data values in \( t \) imply \( \phi = v \).

Two kinds of constraints are of interest in this paper: functional constraints and multivalued constraints. Functional constraints, which are the same as FDs of the relational database theory [16], arise as a result of functional relationship sets in a CM hypergraph.

Definition 14. Let \( M \) be a CM hypergraph and let \( F \) be a scheme-tree forest that syntactically covers \( M \). Let \( X \rightarrow Y \) be a functional edge in \( M \) that (properly) appears in a scheme tree \( T \) in \( F \). For any scheme-tree instance \( t \) over \( T \), the functional constraint \( X \rightarrow Y \) holds for \( T \) if and only if the FD \( X \rightarrow Y \) holds in the total unnesting of \( t \) projected on the functional edge \( X \rightarrow Y \) in \( M \).

![Fig. 4. CM hypergraph and syntactically covering scheme tree.](image)

![Fig. 5. Valid and invalid scheme-tree instances. (a) Valid. (b) Invalid.](image)
Suppose $F$ is a scheme-tree forest that semantically covers a valid, populated, CM hypergraph $M$. If a scheme tree $T$ in $F$ includes two connected edges of $M$, then these two edges may cause redundancy in the scheme-tree instances over $T$. To see this, assume $E_1$ and $E_2$ are two connected edges of $M$ that appear in $T$ (e.g., let $E_1 = AB$ and $E_2 = BC$ in the example in Fig. 5). Let $t$ be the scheme-tree instance over $T$ as dictated by Definition 12 (e.g., the valid scheme-tree instance in Fig. 5a). If $e_1$ and $e_2$ are two joinable relationships on $E_1$ and $E_2$, respectively, and both $e_1$ and $e_2$ appear in $t$, then by Condition 3 of Definition 12, all data values of $e_1$ and $e_2$ must appear in a tuple in the total unnesting of $t$ (e.g., $e_1 = (1, 3)$ and $e_2 = (3, 4)$ in Fig. 5a). Now, if there is a relationship $e_3$ on $E_1$ such that $e_3$ also appears in $t$ and $e_3$ and $e_2$ are also joinable, then $e_2$ might be forced to appear a second time in $t$ to satisfy Condition 3 of Definition 12 (e.g., for our specific example $e_3 = (2, 3)$ forces $e_2 = (3, 4)$ to appear a second time). A second such appearance causes redundancy (e.g., we can replace either of the $4$s in Fig. 5a with $\phi$ and know that $\phi = 4$ based on the constraints of joinability and the other $4$ not replaced by $\phi$). To capture this idea, we define multivalued constraints, which are, in turn, based on a definition of individual joinable relationships.

**Definition 15.** Let $M$ be a CM hypergraph. Let $E_1$ and $E_2$ be edges of $M$ that share at least one object set, and let the edge-vertex connections of $E_1$ and $E_2$, respectively, be $C_1, \ldots, C_n$ and $C_{n+1}, \ldots, C_m$. Let $O = \{O_1, \ldots, O_k\}$ be the set of object sets in $M$ that have at least one connection for both $E_1$ and $E_2$. Let $e_1$ and $e_2$ be individual relationships for $E_1$ and $E_2$, respectively. The relationships $e_1$ and $e_2$ are joinable if for every object set $O_q$ in $O$ there exists an object $o_q$ in $O_q$ such that $o_q \in e_1[C_i] = e_2[C_j]$ for some connection $C_i$ of $O_q$ among the $E_1$‘s connections, $\{C_1, \ldots, C_n\}$, and some connection $C_j$ of $O_q$ among the $E_2$‘s connections, $\{C_{n+1}, \ldots, C_m\}$.

**Definition 16.** We denote a multivalued constraint over connected edges $E_1$ and $E_2$ as

$$\triangleright_{C_i=C_{i+1}=\ldots=C_{i+n}}(C_1 \cdots C_n, C_{n+1} \cdots C_m),$$

where the edge-vertex connections of $E_1$ and $E_2$, respectively, are $C_1, \ldots, C_n$ and $C_{n+1}, \ldots, C_m$ such that $n \geq 2$, $m \geq n + 2$, and $\min(n, m - n) \geq k > 0$. The multivalued constraint holds if for any representation of the relationship sets of $E_1$ and $E_2$ that allows joinable relationships $e_1$ of $E_1$ and $e_2$ of $E_2$ to be a single joined relationship in which each value of a common connection ($C_1$ and $C_{n+1}$, ..., and $C_k$ and $C_{n+k}$) appears only once, $e_1$ and $e_2$ are joined as a single relationship.\(^7\)

Definition 16 is fundamental to our work and deserves some comment.

\(^7\) The last sentence of this definition is intentionally loose. To make it tight, we need careful definitions for every representation of interest. In this paper, we are only interested in scheme trees. We can make the definition tight for scheme trees as follows. The multivalued constraint holds if for any scheme-tree forest that syntactically and semantically covers $M$ as defined in Definitions 10 and 12, there exists a scheme tree $T$ in which $E_1$ and $E_2$ are connected, then joinable relationships $e_1$ of $E_1$ and $e_2$ of $E_2$ must appear as a tuple in the total unnesting of any scheme-tree instance $t$ over $T$ projected on $E_1E_2$.\(^7\)
unavoidable. Since this is also true for the joinable relationships (8, 4, 1) and (4, 2) and indeed for any joinable relationships from DAA and AB, the multivalued constraint $\bowtie C_1 \bowtie C_4(DAA, AB)$ holds and, as we shall see, can never cause redundancy in any valid instance. For $\bowtie(B, BC)$, we see that (1, 2) and (2, 3) are joinable and so are (4, 2) and (2, 3). Here, however, (2, 3) has to appear twice in order to satisfy Definition 16. If we remove either one of the (2, 3)'s from the scheme-tree instance, either (1, 2, 3) or (4, 2, 3) will not appear in a tuple of the total unnesting. Thus, although we can make the multivalued constraint $\bowtie(A, BC)$ hold, we must duplicate the (2, 3) relationship in the scheme-tree instance. As we shall see, this duplication causes redundancy. Further, although we need not duplicate the (6, 7) for the current instance, we would have to duplicate it if another relationship for AB having a 6 for the B component such as (4, 6) were to be inserted. In general, as we shall see, the multivalued constraint $\bowtie(A, BC)$ will always cause potential redundancy for the scheme tree $(A, (B, (C^*))^*, (A, D^*))^*$.

Example 8. As examples of scheme-tree forests syntactically and semantically covering the CM hypergraph in Fig. 1, consider the following:

1. $(\text{Department}, (\text{Faculty Member, (Hobby)})*, (\text{Grad Student, Program, (Hobby)})*)^*$,
2. $(\text{Faculty Member, Department, (Hobby)})*, (\text{Grad Student, Program, (Hobby)})*)^*$,
3. $(\text{Hobby, (Faculty Member)})*, (\text{Grad Student})* (\text{Grad Student, Program, Faculty Member})^* (\text{Department, Faculty Member})^*$,
4. $(\text{Hobby, (Faculty Member, Department)})*, (\text{Grad Student, Program, Faculty Member})^*$, and
5. $(\text{Faculty Member, Department, (Hobby, (Grad Student)})* (\text{Grad Student, Faculty Member, Program})^*$.

We claim (and will shortly prove) the following about these scheme-tree forests. The first two sample scheme-tree forests are redundancy-free with respect to functional and multivalued constraints. The third scheme-tree forest, as well as the scheme-tree forest in Fig. 2, also never allow redundancy with respect to functional and multivalued constraints. Both, however, have more than one scheme tree and are thus not as compact as the first two scheme-tree forests. The fourth sample scheme-tree forest allows redundancy based on functional constraints. Since faculty members are listed repeatedly for each additional Hobby in which they participate, functionally determined department values can be redundant. Similarly, functionally determined program values as well as functionally determined faculty advisors for grad students can be redundant. The fifth scheme-tree forest also allows redundancy based on multivalued constraints. Whenever faculty members have the same hobbies, all the grad students that also share these hobbies are listed.

Example 9. Consider the model instance in Fig. 1 as the input to Algorithm 1. If we select Department as the root node, we make Faculty Member a child node of Department and designate it as a continuation attribute and then make Hobby a child node of Faculty Member; further, since Faculty Member is a continuation attribute, we make Grad Student a child node of Faculty Member and designate it as a continuation attribute and then add Program in the node with Grad Student and finally make Hobby a child node of the node containing Grad Student. The result is the first scheme tree in Example 8. If we select Faculty Member as the starting node, we generate the second scheme tree in Example 8. If we select Grad Student as the starting node, proceed as far as we can, and then select Faculty Member from the remaining unmarked nodes and proceed, we can generate the scheme-tree forest in Fig. 2. Similarly, we can generate the third scheme-tree forest in Example 8, by starting with Hobby, then Grad Student, and then Department. We cannot, however, generate either the fourth or the fifth scheme-tree forest in Example 8.

Observe from our discussion of Examples 8 and 9 that Algorithm 1 disallows the sample scheme-tree forests that have redundancy. Indeed, we claim, and will prove in 8. “Canonical” is a standard form defined in Section 4.
Section 4 that Algorithm 1 generates only scheme-tree forests that have no redundancy with respect to functional and multivalued constraints.

Although Algorithm 1 can guarantee no redundancy, generating a minimum number of redundancy-free scheme trees from a canonical, binary CM hypergraph is NP-hard, as we shall prove in Section 4. Therefore, we cannot have polynomial-time algorithms for a minimum number of redundancy-free scheme trees. We shall, however, provide a heuristic, greedy algorithm that usually generates the fewest number of scheme trees from a canonical, binary CM hypergraph. We shall, however, provide a heuristic, greedy algorithm that usually generates the fewest number of scheme trees from a canonical, binary CM hypergraph.

4 ASSUMPTIONS, REQUIREMENTS, AND GUARANTEES

Unfortunately, Algorithm 1 does not work for any CM hypergraph. Two conditions are required: 1) canonical and 2) binary. We discuss the canonical requirement in this section and show how to remove the binary requirement in the next section.

Definition 17. Let \( E \) be an edge (\( V \) be a vertex) of a CM hypergraph \( H \) with a valid interpretation. Let \( H' \) be \( H \) without \( E \) (without \( V \) and all edges incident on \( V \)). Edge \( E \) (vertex \( V \)) is redundant if 1) \( H' \) preserves the information in \( H \) (i.e., if there exists a procedure to construct \( H' \), including its data instance, from \( H' \)) and 2) \( H' \) preserves the constraints of \( H \) (i.e., if the constraints of \( H' \) imply the constraints of \( H \)).

Definition 18. A binary CM hypergraph \( H \) is canonical if 1) no edge of \( H \) is redundant, 2) no vertex of \( H \) is redundant, and 3) bidirectional edges of \( H \) represent bijections.\(^9\)

We now turn our attention to proving that Algorithm 1 generates redundancy-free scheme trees with respect to the functional and multivalued constraints of a given canonical, binary CM hypergraph. The essential idea of the proof is to show that each relationship in a populated CM hypergraph instance appears once and only once in any proper scheme-tree instance of a scheme tree generated by the algorithm. In doing so, we often mention subtuples and sequences of subtuples.

Definition 19. Each partial function in a scheme-tree instance is a subtuple. If \( t \) is a subtuple of the root node of a scheme tree \( T \), then \( t \) plus the set of all subtuples recursively associated with \( t \) is a tuple of the scheme-tree instance over \( T \). If \( t_1, \ldots, t_n \) are subtuples of a scheme-tree instance for a scheme tree \( T \) such that for every \( i (1 \leq i < n) \), \( t_i \) is a subtuple for node \( N_i \) in \( T \), \( N_i \) is the parent node of \( N_{i+1} \) in \( T \), and \( t_{i+1} \) is a subtuple in the scheme-tree instance that is associated with \( t_i \), then \( t_1, \ldots, t_n \) is a sequence of subtuples.

Example 10. In Fig. 6, let the first \( A \) be \( A_1 \) and the second \( A \) be \( A_2 \). Then, \( \{ (A_1, 1), (B, 2), (C, 3), \} \) and \( \{ (A_2, 4), (D, 5) \} \) are subtuples over the nodes \( A_1, B, C, \) and \( A_2, D \), respectively. Further, \( \{ (A_1, 1), (B, 2), (C, 3) \} \) is a sequence of subtuples and so is \( \{ (A_1, 1), (A_2, 4), (D, 5) \} \). On the other hand, \( \{ (C, 3), (A_2, 4), (D, 5) \} \) is not a sequence of subtuples because they are not subtuples for nodes of a contiguous subpath. Neither is \( \{ (A_1, 4), (B, 6), (C, 7) \} \) because \( \{ (B, 6) \} \) is not in a scheme-tree instance that is associated with the subtuple \( \{ (A_1, 4) \} \).

Lemma 1. Let \( F \) be a scheme-tree forest that semantically covers a canonical, CM hypergraph \( M \). Let \( X \rightarrow Y^{10} \) be a functional edge of \( M \) that appears in (not necessarily distinct) scheme trees \( T \) and \( T' \) in \( F \). Let \( A \in Y \). Suppose \( t \) and \( t' \) (not necessarily distinct) are the scheme-tree instances over \( T \) and \( T' \), respectively. \( F \) has redundant data values with respect to \( X \rightarrow Y \) if and only if there is a relationship \( r \) on \( X \rightarrow Y \) in \( M \) such that \( r \) appears in two distinct sequences of subtuples \( s_1, s_2, \ldots, s_n \) in \( t \) and \( s'_1, s'_2, \ldots, s'_m \) in \( t' \) and \( r(A) \) is contained in two distinct subtuples \( s_p \) (\( 1 \leq p \leq n \)) and \( s'_q \) (\( 1 \leq q \leq m \)).

Proof. For the if-part, since \( r \) appears in both sequences of subtuples, \( r(XA) \) also appears in both of them. Since \( r(A) \) is contained in \( s_p \) and \( s'_q \), \( s_p(A) = s'_q(A) = r(A) \). Thus, if we replace \( s'_q(A) \) by a symbol, say \( \phi \), we can derive \( \phi = r(A) \) by using the constraint \( X \rightarrow A \), the equality on the \( X \) values in the two sequences, and the value \( s_p(A) \). For the only-if-part, suppose \( F \) has redundant data values with respect to \( X \rightarrow Y \). Then, \( F \) has redundant data values for \( A \) with respect to \( X \rightarrow A \) where \( A \in Y \). If for every relationship \( r \) on \( X \rightarrow Y, r \) appears in a unique sequence of subtuples, then \( r(XA) \) is contained in the same sequence of subtuples which means \( r(A) \) is not redundant. Thus, there exists a relationship \( r \) on \( X \rightarrow Y \) such that \( r \) appears in at least two distinct sequences of subtuples. Further, if \( r(A) \) is contained in the same subtuple of these sequences of subtuples, \( r(A) \) is not redundant. Thus, \( r(A) \) is contained in at least two distinct subtuples in these sequences. \( \Box \)

Lemma 2. Let \( F \) be a scheme-tree forest that semantically covers a canonical, CM hypergraph \( M \) and let \( T \) be a scheme tree in \( F \). Suppose \( t \) is the scheme-tree instance over \( T \) and \( E_1 \) and \( E_2 \) are two connected edges of \( M \) that appear in \( T \). If \( F \) has redundant data values with respect to \( t \ Vladimir(D, 2) \), then there is a relationship \( r \) on \( E_2 \) such that \( r \) appears in at least two distinct sequences of subtuples in \( t \).

Proof. If \( F \) has redundant data values with respect to \( t \ Vladimir(D, 2) \), then there exists a relationship \( r \) on \( E_2 \) such that some values of \( r \) are forced to appear in two distinct subtuples in \( t \). Since all values of \( r \) are related trivially, then by Condition 3 of Definition 12, \( r \) must appear in at least two distinct sequences of subtuples in \( t \).

With Lemmas 1 and 2 in place, we are ready to address Algorithm 1 directly. As a fundamental requirement, Algorithm 1 must halt, which it clearly does.

Lemma 3. Let \( F \) be a scheme-tree forest generated by Algorithm 1 from a canonical, binary, CM hypergraph \( H \). \( F \) syntactically covers \( H \).

Proof. Condition 1 of Definition 10 is trivially satisfied since Algorithm 1 never adds an object set that is not in \( H \) to a


10. If the object sets are not unique, we include edge-vertex connections to make them unique. To simplify the discussion, we use this convention throughout.
scheme tree. Therefore, we are left to show that $F$ satisfies the other conditions of Definition 10.

We proceed by induction on the number $n$ of edges in $H$. When $n = 0$, there is no edge in $H$. Thus, all vertices in $H$ stand alone, which means the while-loop is never entered. Every stand-alone vertex will be marked and will become a root node of a stand-alone scheme tree. Since every vertex appears and since there are no edges, Condition 2 of Definition 10 is satisfied. Condition 3 of Definition 10 is vacuously satisfied. Since each scheme tree has only a single vertex in it, Conditions 4 and 5 do not apply. Hence, $F$ syntactically covers $H$ when $n = 0$.

Assume this lemma is true if $H$ has $k$ edges for some $k \geq 0$. Now, suppose Algorithm 1 selects $E = (A, B)$ in $H$ as the $k + 1$st edge. By the induction hypothesis, Algorithm 1 generates a scheme-tree forest $F$ that syntactically covers $H$ without $E$. We have two cases: 1) both $A$ and $B$ are not continuation attributes in all scheme trees in $F$, or 2) one of them (or both), say $A$, is a continuation attribute in a scheme tree $T$ in $F$. For Case 1, since $E$ is unmarked, a new scheme tree with only $A$ and $B$ in it will be created. Thus, $A, B$, and $E$ all appear in this new scheme tree. Further, since $E$ is the initial edge of this new scheme tree, $E$ is properly connected. Therefore, Conditions 2 and 3 of Definition 10 are satisfied. Since this new scheme tree is created from $E$, and only $E$, Conditions 4 and 5 are also satisfied. For Case 2, since $A$ is a continuation attribute in $T$, $E$ will be marked in the while-loop and later $B$ will be added to $T$ by one of the four mutually exclusive cases in the If-Else-if-Else-if-Else statement. In all four cases, $B$ is added either to the node that contains $A$ or as a new child node to the node that contains $A$. Therefore, Conditions 2, 4, and 5 of Definition 10 are satisfied. Since $A$ is a continuation attribute in $T$ and $A$ is also in $E$, after $B$ is added into $T$, the edges that appear in $T$ are still properly connected, satisfying Condition 3. □

Lemma 4. Let $F$ be a scheme-tree forest generated by Algorithm 1 from a canonical, binary, CM hypergraph $H$. $F$ semantically covers $H$. Further, if a vertex (an object set) $V$ is designated as a continuation attribute in a scheme tree $T$ in $F$, then the $V$-column of $T$ can be populated with all objects in $V$.

Proof. We proceed by induction on the number $n$ of edges in $H$. When $n = 0$, every vertex in $H$ stands alone, and Algorithm 1 creates a scheme tree for each vertex $V$ of $H$. Since each $V$ is the first (and only) attribute in a scheme tree, we can populate the $V$-column with all objects in $V$ (but nothing else). Thus, Condition 1 of Definition 12 is satisfied. Since there are no edges, Conditions 2 and 3 of Definition 12 are vacuously satisfied.

Assume this lemma is true if $H$ has $k$ edges for some $k \geq 0$. Now, suppose Algorithm 1 selects $E = (A, B)$ in $H$ as the $k + 1$st edge. By the induction hypothesis, Algorithm 1 generates a scheme-tree forest $F$ that semantically covers $H$ and each continuation attribute $V$ in $F$ can be populated with all objects in $V$. We have two cases: 1) neither $A$ nor $B$ is a continuation attribute in any scheme tree in $F$, or 2) one of them (or both), say $A$, is a continuation attribute in a scheme tree $T$ in $F$. For Case 1, either $A$ or $B$, say $A$ is selected and designated as the first continuation attribute of a new scheme tree $T'$ in the until-loop before the while-loop. Since $A$ is the first continuation attribute of $T'$, we can populate the $A$-column of $T'$ with all objects in $A$ (but nothing else). Thus, Condition 1 of Definition 12 is satisfied for $A$. When the while-loop is entered, $E$ is marked and $B$ is added to $T'$. For each $A$-value $a$ in the $A$-column, we can populate the $B$-column with a $B$-value $b$ if and only if $(a, b)$ is part of a tuple in the total unnesting of the scheme-tree instance on $T'$ projected on $E$. Since all objects in $A$ can be present, then all relationships in $E$ can be present. Therefore, Condition 2 of Definition 12 is satisfied for $E$. Since we populate the $B$-column with $b$ for each occurrence of $a$ in the $A$-column where $(a, b)$ is a relationship on $(A, B)$ in $H$, Condition 3 of Definition 12 is also satisfied. There are two cases for vertex $B$: 1) the $B$-$E$ connection is mandatory, 2) the $B$-$E$ connection is optional. For Case 1, since all relationships on $E$ can be present, then all objects in $B$ can be present. Condition 1 of Definition 12 is thus satisfied for $B$. For Case 2, $B$ is not marked. Indeed, $B$ is never marked until either Case 1 holds or until $B$ is the first continuation attribute in a scheme tree and is therefore treated like $A$. For Case 2, if $A$ is a continuation attribute in a scheme tree $T$, then by the induction hypothesis, all objects in $A$ can be present in the scheme-tree instance on $T$. The remaining argument for Case 2 is similar to that for Case 1 except that if the $B$-$E'$ connection is mandatory for some edge $E' \neq E$, $B$ may have already been marked and, thus, the objects-can-be-present condition for $B$ may have already been satisfied. □

Lemma 5. Let $F$ be a scheme-tree forest generated by Algorithm 1 from a canonical, binary, CM hypergraph $H$. If $R$ is a relationship set (an edge) in $H$, then for every relationship $r$ on $R$, $r$ appears in a unique sequence of subtuples.

Proof. $R$ appears in a scheme tree $T$ in $F$ by Lemma 3. Since every edge is marked only once, $R$ only appears in $T$ and in no other scheme tree. $F$ semantically covers $H$ by Lemma 4, which means that, with respect to $H$, every relationship $r$ on $R$ appears in the scheme-tree instance $t$ on $T$. We are left to show that every relationship $r$ on $R$ appears only once in $t$.

This lemma follows if we can prove

$$V \to Ancestor(N_V),$$

where $V$ is a continuation attribute in a scheme tree $T$ in $F$, $N_V$ is the node in $T$ that contains $V$, and $Ancestor(N_V)$ is the union of all ancestor nodes of $N_V$ in $T$ including $N_V$. To see that the lemma follows, observe that a relationship set $R = (A, B)$ is added to $T$ if and only if one of its vertices, say $A$, is a continuation attribute in $T$. Suppose $(a, b)$ is a relationship on $R$. If $A \to Ancestor(N_A)$, then the $A$-value $a$ associates with a unique value for every attribute in every ancestor node of $N_A$ including $N_A$. Thus, using Definition 4 since each embedded scheme-tree instance in $t$ is a set, $(a, b)$ appears in a unique sequence of subtuples in $t$.

We now proceed to prove $V \to Ancestor(N_V)$ for each continuation attribute $V$ by induction on the number $n$ of edges in $H$. Suppose $n = 0$. Then, $H$ has no edges. Algorithm 1 outputs a scheme tree $T$ for each vertex $V$ in $H$ such that $T$ consists only of $V$. Further, $V$ is designated as a continuation attribute in $T$. Thus, the basis is established. Consider adding a relationship set $R = (A, B)$ to a scheme tree $T$ in $F$. One of its vertices, say
A, must be a continuation attribute in \( T \). By the induction hypothesis, \( A \rightarrow \text{Ancestor}(N_A) \). In the four mutually exclusive cases in the if-Else-Elseif-Else statement, \( B \) is added to \( N_A \) if \( A \leftarrow B \) or \( A \rightarrow B \). Thus, it is still true that \( A \rightarrow \text{Ancestor}(N_A) \). Further, \( B \) is designated as a new continuation attribute in \( T \) if \( A \leftarrow B \). Thus, it is also true that \( B \rightarrow \text{Ancestor}(N_B) \) since \( N_A = N_B \) in this case. \( B \) is added as a child node to \( N_A \) if \( B \rightarrow A \) or none of the former cases is true. Thus, it is still true that \( A \rightarrow \text{Ancestor}(N_A) \) since \( N_A \) remains the same. Further, \( B \) is designated as a new continuation attribute in \( T \) if \( B \rightarrow A \). Thus, it is also true that \( B \rightarrow \text{Ancestor}(N_B) \). \hfill \Box

**Theorem 2.** Let \( F \) be a scheme-tree forest generated by Algorithm 1 from a canonical, binary, CM hypergraph \( H \). No scheme-tree instance over any scheme tree in \( F \) can have redundant data values with respect to the functional constraints and the multivalued constraints of \( H \).

**Proof.** Since \( H \) is canonical, by Definition 18 no edge and no vertex of \( H \) is redundant. Thus, no set of relationships for an edge and no set of values for a vertex is derivable from other edge relationship sets and vertex value sets. Thus, we are left to prove that every relationship \( r \) for every edge appears in some scheme-tree instance of \( F \) such that no redundant data values exist with respect to the functional and multivalued constraints of \( H \). By Lemma 5, every relationship \( r \) in any scheme-tree instance for any scheme tree of \( F \) appears in a unique sequence of subtuples. Thus, by Lemma 1, no scheme-tree instance of \( F \) can have redundancy with respect to the functional constraints of \( H \), and by Lemma 2, no scheme-tree instance of \( F \) can have redundancy with respect to the multivalued constraints of \( H \). \hfill \Box

Having resolved the redundancy question, we now address the compactness problem.

**Theorem 2.** Generating a minimum number of redundancy-free scheme trees from a canonical, binary CM hypergraph \( H \) is NP-hard.

**Proof.** It suffices to find an NP-hard special case. Suppose \( H \) does not have any functional edges. Thus, since \( H \) is binary, \( H \) is an ordinary graph. Let \( F \) be the resulting scheme-tree forest of applying Algorithm 1 on \( H \). Since \( H \) does not have any functional edges, each path of a scheme tree in \( F \) corresponds to a unique edge in \( H \). Hence, the set of root nodes in \( F \) is a vertex cover\(^{11}\) of \( H \). Since finding a minimum vertex cover is NP-hard [12], this theorem follows. \hfill \Box

Instead of searching exhaustively, we believe adding the following to Algorithm 1 should suffice to generate a minimum number of redundancy-free scheme trees in most practical cases.

- Before the Until statement, add the following statements:
  - Compute the functional closure of each vertex using only fully functional edges (i.e., using only functional edges whose domain side is not optional);
  - Order the vertices so that those in the greatest number of closures appear first (the order among those in the same number of closures does not matter);
  - Discard from the tail-end of the ordered list, those vertices included in only a single closure;
  - Order the discarded vertices so that those with the greatest number of incident edges appear first;
  - Append this list of ordered “discarded” vertices to the tail-end of the first ordered list (the order among those in the same number of closures does not matter) (note that either one of the two ordered lists being joined together may be empty);

- Change the If-Else statement that selects the root node of a new scheme tree in Algorithm 1 to:

From the ordered list of vertices, select the first unmarked vertex \( V \) in \( H \) to be the root node of a new scheme tree \( T \).

If there is no unmarked vertex left in the list, then select the marked vertex \( V \) such that \( V \) is contained in an unmarked edge and \( V \) comes before any other vertices contained in unmarked edges in the list.

We call Algorithm 1 with this modification Algorithm 1.1. Algorithm 1.1 is a heuristic, greedy algorithm that successively finds and generates scheme trees from the largest remaining hierarchical structure in a canonical, binary CM hypergraph.

**Example 11.** For the CM hypergraph in Fig. 1, fully functional closures are:

\[
\text{Department}^+ = \{\text{Department}\}, \quad \text{Faculty Member}^+ = \{\text{Faculty Member}, \text{Department}\}, \quad \text{Grad Student}^+ = \{\text{Grad Student}, \text{Faculty Member}, \text{Department}, \text{Program}\}, \quad \text{Program}^+ = \{\text{Program}\},
\]

and \( \text{Hobby}^+ = \{\text{Hobby}\} \). Thus, \( \text{Department} \) is included in three closures, \( \text{Faculty Member} \) and \( \text{Program} \) are included in two, and \( \text{Grad Student} \) and \( \text{Hobby} \) are included in one. Since the last two vertices on the list are included in only a single closure, they are ordered according to their respective number of incident edges: \( \text{Grad Student} \) has three and \( \text{Hobby} \) has two. Hence, the order is \( \text{Department}, \text{Faculty Member}, \text{Program}, \text{Grad Student}, \text{and Hobby} \). Observe that Algorithm 1.1 produces the first scheme tree in Example 8.

An alternate criteria for selecting the starting node for a new scheme tree that often gives intuitively better trees is:

Let \( V \) be the selected vertex in Algorithm 1.1.

If \( V \) has exactly one incident unmarked edge \( E \) and \( E \) is fully functional from \( W \) to \( V \), i.e. \( W \rightarrow V \), select \( W \), i.e., let \( V \) be \( W \) instead;

Else Select \( V \);

We call Algorithm 1 with this modification Algorithm 1.2.
Example 12. Algorithm 1.2 produces the second scheme tree in Example 8.

Example 13. To see that these heuristic algorithms do not always produce the most compact scheme-tree forest, consider the following CM hypergraph with six edges: AB, CD, EF, BG, DG, and FG. Algorithms 1.1 and 1.2 both generate four scheme trees: \((G, (B)^*, (D)^*, (F)^*)^*, (B, (A)^*)^*, (D, (C)^*)^*, \) and \((F, (E)^*)^*\). Searching exhaustively, however, it is easy to see that only three scheme trees are necessary: \((B, (A)^*, (G)^*)^*, (D, (C)^*, (G)^*)^*, \) and \((F, (E)^*, (G)^*)^*\).

Although an exhaustive consideration of all real-world conceptual models is impossible and consideration of even a small fraction is impractical, the heuristics in Algorithms 1.1 and 1.2 have always worked for the real-world conceptual models we have encountered.

5 N-ary Algorithm

In this section, we generalize Algorithm 1 for CM hypergraphs with \(n\)-ary edges. Three new problems arise in the generalization: 1) \(n\)-ary edges may be compositions of edges with lesser arity, 2) connecting sets of attributes between relationship sets may have different meanings, and 3) there are more degrees of freedom for scheme-tree configurations. We discuss each of these three problems in turn.

Edge Decomposition. As one example of edge decomposition, consider the edge connecting Name, Address, Phone, and Major in Fig. 8a. If the phone of the person identified by the Name-Address pair is the home phone at the address of the person, the 4-ary edge can be reduced by making the phone dependent only on the address as Fig. 8b shows. As another example of edge decomposition, consider the 5-ary edge in Fig. 8a. Assuming that the schedule depends only on the course itself, we can decompose the edge as Fig. 8b shows. To accommodate these necessary reductions and decompositions, we now augment our definition of what it means for a CM hypergraph to be canonical.

Definition 20. A CM hypergraph \(H\) is canonical if

1. no edge of \(H\) is redundant,
2. no vertex of \(H\) is redundant,
3. bidirectional edges represent bijections, and
4. every \(n\)-ary edge is fully reduced.

Example 14. The CM hypergraph in Fig. 8a is noncanonical—neither the 4-ary nor the 5-ary edge is fully reduced. Assuming that none of the ternary edges can be further reduced, the CM hypergraph in Fig. 8b is canonical.

Different Meanings. Consider Name-Address pairs in Fig. 8b. Suppose there are two names \(n_1\) and \(n_2\) and two addresses \(a_1\) and \(a_2\). Further suppose that in the relationship set with Major, \(n_1\) relates to \(a_1\) and \(n_2\) relates to \(a_2\), but in the relationship set with Course, \(n_1\) relates to \(a_2\) and \(n_2\) relates to \(a_1\). Under these assumptions, we cannot have the scheme tree \((\text{Major}, (\text{Name, Address, (Course)^*})^*\), which we would expect should be permissible. We cannot have this scheme tree because under Major our scheme-tree instance would have the tuples \(\{(\text{Name}, n_1), (\text{Address, a_1})\}\) and \(\{(\text{Name}, n_2), (\text{Address, a_2})\}\). To nest courses under these tuples, however, we need \(\{(\text{Name, n_1), (\text{Address, a_2})}\}\) and \(\{(\text{Name, n_2), (\text{Address, a_1})}\}\).

In general, to provide for nesting, the projections on the intersecting attribute sets between two edges must be identical for every valid interpretation. This condition holds when the projections on the intersecting object sets between two edges have the “same meaning.” For the natural interpretation of the CM hypergraph in Fig. 8b, Name-Address pairs have the same meaning in both the Major relationship set and the Course relationship set—in both, the pair identifies an individual student. Thus, we would not have the example we contrived above to illustrate that the Name-Address pairs could be different.

Definition 21. Let edges \(E_1\) and \(E_2\) have a nonempty intersection of vertices, and let \(X \subseteq E_1 \cap E_2\) such that \(X \neq \emptyset\). Let \(r_{E_1}\) and \(r_{E_2}\), respectively, be relations for \(E_1\) and \(E_2\). If for every valid interpretation \(\pi_X(r_{E_1}) \subseteq \pi_X(r_{E_2})\) holds, then \(E_1\) has the same meaning as \(E_2\) on \(X\). If for every valid interpretation, both \(\pi_X(r_{E_1}) \subseteq \pi_X(r_{E_2})\) and \(\pi_X(r_{E_1}) \supseteq \pi_X(r_{E_2})\) hold, then \(E_1\) and \(E_2\) have exactly the same meaning on \(X\).

Degrees of Freedom. Consider the ternary is-taking-course relationship set in the CM hypergraph in Fig. 8b. The scheme trees we may create for this relationship set include \((\text{Course, Name, Address}^*)^*\) and \((\text{Course, (Name, Address)}^*)^*\) among others. Whereas there are only three possible scheme trees for a nonfunctional binary relationship set, there are 13 possible scheme trees for a nonfunctional ternary relationship set such as the is-taking-course relationship set in Fig. 8b. For an \(n\)-ary edge in general, the number of scheme-tree configurations is exponential in terms of \(n\).

These new problems make it more difficult to specify a scheme-tree generation algorithm. The basic idea for the generalization of Algorithm 1, however, is still to search for and establish the largest hierarchical structures embedded within the CM hypergraph. Fig. 9 gives our \(n\)-ary algorithm. At first glance, the algorithm appears to be considerably different from the algorithm in Fig. 7. It is, however, the natural generalization of Algorithm 1. The next few examples explain the differences and also serve to illustrate the various parts of Algorithm 2.

Algorithm 1 begins by constructing a new scheme tree with a single vertex as the root node. Example 15 shows,
Example 15. For the CM hypergraph in Fig. 8b, Algorithm 2 should allow \( \{ \text{Name}, \text{Address} \} \), among other choices, to be the initial root node for a new scheme tree. Then, with this choice, as in Algorithm 1 we may add \( \text{Major} \) in the same node (since it is functionally determined, i.e., \( \text{Name}, \text{Address} \rightarrow \text{Major} \)) and \( \text{Course} \) as a child node since it is not functionally determined. Note that the set of nodes \( \{ \text{Name}, \text{Address} \} \), which semantically represents a single concept, namely, a person, acts as a single-concept node, and that Algorithm 2 treats the single concept analogous to the way Algorithm 1 does.

It is clear that Condition 1 of the while-loop in Fig. 9 must hold. Otherwise, we may not be able to semantically cover the CM hypergraph.

Condition 2 of the while-loop in Fig. 9 addresses the issue of different meanings. In Algorithm 1 this problem does not arise. When the edges are binary, we can guarantee that any scheme-tree instance has a proper set of values by observing optional and mandatory constraints alone. For the \( n \)-ary case, we must also have a semantic guarantee whenever we wish to graft in an edge using two or more connecting vertices.

Definition 22. Let \( E \) be an edge in a canonical, CM hypergraph \( M \). Let \( T \) be a scheme tree such that some edges in \( M \) appear in \( T \). Let \( \mathbb{T}_S \) be the set of attributes in \( T \). For a valid interpretation \( I \) of \( M \), let \( r_E \) be the relation for \( E \) for \( I \) and \( t \) be a scheme-tree instance created from \( I \) for \( T \). If for every valid interpretation \( I \) of \( M \), there exists an \( X \subseteq E \cap \mathbb{T}_S \) such that \( \pi_X(r_E) \subseteq \pi_X(t) \) holds where \( t \) is the total unnesting of \( t \), then \( E \) has the same meaning as \( T \) on \( X \), denoted by \( \text{Same}(E, T, X) \).

Given an edge \( E \) of a canonical, CM hypergraph \( H \), a nonempty subset \( X \) of \( E \), and a scheme tree \( T \), \( \text{Same}(E, T, X) \) must hold in order to allow us to add \( E \) to \( T \). In turn, whether \( \text{Same}(E, T, X) \) holds or not depends on the constraints of \( H \), as Example 16 shows.

Example 16. For Fig. 8b, let \( E_1 \) be the relationship set \( \{ \text{Course}, \text{Name}, \text{Address} \} \) and let \( E_2 \) be the relationship set \( \{ \text{Major}, \text{Name}, \text{Address} \} \). Suppose that one of the constraints of the canonical, CM hypergraph in Fig. 8b is that \( E_1 \) has the same meaning as \( E_2 \) on \( \{ \text{Name}, \text{Address} \} \). Notationally, we can let a user specify this constraint by stating in the diagram that \( \{ \text{Course}, \text{Name}, \text{Address} \} \) has the same meaning as \( \{ \text{Major}, \text{Name}, \text{Address} \} \) on \( \{ \text{Name}, \text{Address} \} \) or stating the stronger “exactly the same meaning” constraint. Alternatively, the algorithm can run interactively and synergistically ask about “same meaning” as needed. In either case, let \( T \) be a scheme tree created from \( E_2 \) with \( \text{Major} \) as the root node and \( \{ \text{Name}, \text{Address} \} \) as a child node of \( \text{Major} \). If \( E_1 \) has the same meaning as \( E_2 \) on \( \{ \text{Name}, \text{Address} \} \), \( \text{Same}(E_1, T, \{ \text{Name}, \text{Address} \}) \) holds, and Algorithm 2 can add \( \text{Course} \) as a child node to \( \{ \text{Name}, \text{Address} \} \).

Example 17 demonstrates that it is not easy to extend the concept of continuation attributes for binary edges in Algorithm 1 to continuation sets of attributes for \( n \)-ary edges in Algorithm 2.

Example 17. Consider three ternary edges \( E_1 = ABC \), \( E_2 = ABD \), and \( E_3 = ABE \), where \( E_1 \) and \( E_2 \) have exactly the same meaning on \( AB \), but \( E_3 \) has a different meaning on \( AB \). We might, for example, create a single-path scheme tree \( T \) starting with \( E_1 \) where \( A \) is the root node, \( B \) is a child node of \( A \), and \( C \) is a child node of \( B \). Since \( E_1 \) and \( E_2 \) have exactly the same meaning on \( AB \), \( \text{Same}(E_2, T, AB) \) holds. Thus, we might consider \( AB \) as a continuation set of attributes. Indeed, it can be designated as such for \( E_2 \) since we can add \( D \) as another child node to \( B \). For \( E_3 \), however, \( AB \) is not a continuation set of attributes, and we cannot add the attribute \( E \) as a child node of \( B \) since \( E_3 \) has a different meaning on \( AB \). If \( \text{Same}(E_3, T, A) \) holds, a correct solution is to add \( B E \) as an additional child node to \( A \) instead. Thus, whether a set of attributes can be designated as a continuation set of attributes depends on the edges involved. Because of this difficulty, the concept of continuation sets of attributes is of little use for Algorithm 2.

Example 18 illustrates the need for Condition 3 in Algorithm 2 and demonstrates that we should be careful about the way we graft edges into a growing scheme tree. In particular, not only do we need the same meaning for the intersecting set of vertices \( X \) for the edge \( E \) we are grafting into the tree as Example 16 shows, we also need \( X \) to be a nonempty maximal subset of \( E \) on which \( E \) and \( T \) have the
same meaning. The reason X needs to be defined this way is to avoid representing an edge component more than once, as Example 18 shows.

Example 18. Consider two ternary edges \( E_1 = ABC \) and \( E_2 = ABD \). Suppose \( E_1 \) and \( E_2 \) have exactly the same meaning on \( AB \). Let \( T \) be a scheme tree created from \( E_1 \) with \( A \) as the root node and \( BC \) as a child node of \( A \). Since \( E_1 \) and \( E_2 \) have exactly the same meaning on \( AB \), \( Same(E_2, T, A) \) holds. Suppose we add \( BD \) as another child node of \( A \), then \( AB \) would appear twice in \( T \) with exactly the same meaning. Now, however, since \( E_1 \) and \( E_2 \) have exactly the same meaning on \( AB \), then every row in \( \pi_{AB}E_1 \) and \( \pi_{AB}E_2 \) appears more than once in a scheme-tree instance over \( T \), and thus we have redundancy. A better scheme tree for \( E_1 \) and \( E_2 \) has \( A \) as the root node, \( B \) as a child node of \( A \), and \( C \) and \( D \) as child nodes of \( B \). In this scheme tree, \( AB \) appears just once and, thus, every row in \( \pi_{AB}E_1 \) (or \( \pi_{AB}E_2 \)) appears only once.

In Condition 4, \( N_X \) is defined as the highest node in a scheme tree such that \( X \subseteq Ancestor(N_X) \). We do not need \( N_X \) in Algorithm 1 since continuation attributes serve that purpose. Without \( N_X \), Example 19 shows that it is possible for \( E \) not to appear (properly) in a path.

Example 19. Consider the edges \( E_1 \) and \( E_2 \) in Example 18 again. Let \( T \) be a scheme tree created from \( E_1 \) with \( A \) as the root node, \( B \) as a child node of \( A \), and \( C \) as a child node of \( B \). When we add \( E_2 \) to \( T \), \( X = AB \) and \( N_X \) is the node that contains \( B \). Thus, \( D \) becomes another child node of \( B \). Now, suppose \( X \rightarrow C \). Without knowing \( N_X \) and since \( E_2 \rightarrow E_1 \), \( D \) can be a child node of \( C \) without causing redundancy with respect to the functional and multivalued constraints derived from \( E_1 \) and \( E_2 \). If this happens, however, \( E_2 \) does not appear (properly) in a path since the vertices of \( E_2 \) are not contiguous.

Example 20. As a final example, we apply the algorithm to the canonical, CM hypergraph in Fig. 8b. We assume that the constraint “\((Course, Name, Address)\) which has exactly the same meaning as \((Major, Name, Address)\) on \{Name, Address\}” is given. Algorithm 2 can select \( \{Name, Address, Major\} \) as its first edge \( E \) and can select \( \{Name, Address\} \) as its nonempty subset of vertices \( V \) of \( E \). With this choice, Algorithm 2 marks both \( Name \) and \( Address \), makes \( V \) become the root node of a new scheme tree \( T \), and selects \( X = V \) for the first iteration of the while-loop. With \( X = V = \{Name, Address\} \), all of the conditions hold for \( E \). Inside the while-loop, \( E - X = Major \). Since \( Name, Address \rightarrow Major, X \rightarrow E \), and Algorithm 2 adds \( Major \) to the root node. In the next iteration of the while-loop, Algorithm 2 can only select \( \{Course, Name, Address\} \) as its second edge \( E \). Setting \( X \) equal to \( \{Name, Address\} \) again satisfies all conditions of the while-loop. Note that \( Same(E, T, X) \) must hold and that it does by our assumption. In this iteration, however, \( E - X = Course \) and, since \( X \not\rightarrow Course \), \( Course \) becomes a child node of the root node of \( T \). In the next iteration, Condition 4 fails for both remaining edges (i.e., \( Course \not\rightarrow Name, Address \)). Thus, Algorithm 2 begins building a new scheme tree. Assuming it selects the edge \( Address \rightarrow Phone \) and lets \( V = Address \), it then adds \( Phone \) to the root, but can add nothing else. Thus, Algorithm 2 begins building another new scheme tree with the only remaining edge, \((Course, Day, Time)\). Assuming it lets \( V = \{Course, Time\} \), it then adds \( Day \) as a child node under the root node. The final generated scheme-tree forest is thus

\[ \{(Name, Address, Major, (Course)\}^*, (Address, Phone)^*, (Course, Time, (Day))\}. \]

We now prove that Algorithm 2 is correct.

Lemma 6. Let \( F \) be a scheme-tree forest generated by Algorithm 2 from a canonical, CM hypergraph \( H \). \( F \) syntactically covers \( H \).

Proof. Condition 1 of Definition 10 is trivially satisfied since Algorithm 2 never adds an object set that is not in \( H \) to a scheme tree. To show that \( F \) satisfies the other conditions of Definition 10, we proceed by induction on the number \( n \) of edges in \( H \). The argument for the case that \( n = 0 \) is similar to that of the proof for Lemma 3. Inductively, assume this lemma is true if \( H \) has \( k \) edges for some \( k \geq 0 \).

Now, suppose Algorithm 2 selects \( E \) in \( H \) as the \( k + 1 \)st edge. We have two cases: Either all of the conditions of the while-loop in Algorithm 2 hold for a scheme tree \( T \) or not. In the first case, Algorithm 2 adds \( E \) to \( T \). In adding \( E \) to \( T \), either \( E - X \) is added to \( N_X \) or becomes a list whose head is a child node of \( N_X \). Therefore, if \( N_X \) is a leaf node of \( P \), \( E \) appears (properly) in \( P \); and if \( N_X \) is not a leaf node, \( E \) appears (properly) in a new path in \( T \). Hence, Conditions 3, 4, and 5 of Definition 10 are satisfied. In the second case, when all conditions of the while-loop do not hold and there still exists an unmarked edge, Algorithm 2 creates a new scheme tree from \( E \). Clearly, Conditions 3, 4, and 5 of Definition 10 are all satisfied for this single-edge scheme tree. Since all edges are eventually chosen and satisfy one of the two cases, all edges appear in \( F \), and Condition 2 of Definition 10 is satisfied for edges. After every edge is marked, if unmarked vertices remain, similar to the base case, each vertex becomes the root node of a scheme tree. Any single-vertex scheme tree satisfies Conditions 3, 4, and 5 vacuously. Since every marked vertex marked is included in \( F \) either as part of an edge configuration or as a single-vertex scheme tree, every vertex appears in \( F \), and Condition 2 of Definition 10 is satisfied for vertices. □

Lemma 7. Let \( F \) be a scheme-tree forest generated by Algorithm 2 from a canonical, CM hypergraph \( H \). \( F \) semantically covers \( H \).

Proof. We proceed by induction on the number \( n \) of edges in \( H \). The argument for the case that \( n = 0 \) is similar to that of the proof for Lemma 4. Inductively, assume this lemma is true if \( H \) has \( k \) edges for some \( k \geq 0 \). Now suppose Algorithm 2 selects \( E \) in \( H \) as the \( k + 1 \)st edge. We have two cases: Either all of the conditions of the while-loop in Algorithm 2 hold for a scheme tree \( T \) or not. In the first case, Algorithm 2 adds \( E \) to \( T \). By Condition 1 of the while-loop in Algorithm 2, \( Same(E, T, X) \) holds for some
nonempty subset $X$ of $E$. Thus, since $E$ is joinable with $T$ on $X$ when $E$ is added to $T$, every relationship in $E$ is a tuple in the total unnesting of the scheme-tree instance of $T$ projected on $E$. Therefore, Condition 2 of Definition 12 is satisfied. Since every relationship in every edge is present in the total unnesting of the scheme-tree instance, every object in a marked object set in a marked edge is present. If there is an unmarked object set left after all edges have been marked, it becomes a scheme tree itself. Thus, Condition 1 of Definition 12 is satisfied. Since $\text{Same}(E, T, X)$ holds and $X \rightarrow \text{Ancestor}(N_X)$, each relationship in $E$ only participates in the join once and, thus, Condition 3 of Definition 12 is satisfied. In the second case, when all the conditions of the while-loop do not hold and, thus, there exists an unmarked edge $E$, Algorithm 2 creates a new scheme tree from $E$. Clearly, Conditions 1, 2, and 3 of Definition 12 are all satisfied for this single-edge scheme tree. After every edge is marked, if unmarked vertices remain, similar to the base case, each vertex becomes the root node of a scheme tree. Each single-vertex scheme tree clearly satisfies Condition 1 of Definition 12 and satisfies Conditions 2 and 3 vacuously. \qed

Lemma 8. Let $F$ be a scheme-tree forest generated by Algorithm 2 from a canonical, CM hypergraph $H$. If $R$ is a relationship set (an edge) in $H$, then for every relationship $r$ on $R$, $r$ appears in a unique sequence of subtuples.

Proof. $R$ appears in a scheme tree $T$ in $F$ by Lemma 6. Since every edge is marked only once, $R$ only appears in $T$ and in no other scheme tree. $F$ semantically covers $H$ by Lemma 7, which means that, with respect to $H$, every relationship $r$ on $R$ appears in the scheme-tree instance $t$ on $T$. In addition, since Condition 4 of the while-loop in Algorithm 2 stipulates that $X \rightarrow \text{Ancestor}(N_X)$ and $R$ joins with $T$ on $X$, when $R$ is added to $T$, then every relationship $r$ on $R$ appears in a unique sequence of subtuples in $t$. \qed

Theorem 3. Let $F$ be a scheme-tree forest generated by Algorithm 2 from a canonical, CM hypergraph $H$. No scheme-tree instance in $F$ can have redundant data values with respect to the functional constraints of $H$ and the multivalued constraints of $H$.

Proof. The proof for this theorem is almost identical to the proof for Theorem 1 and is thus omitted. \qed

Since binary edges are $n$-ary edges, by Theorem 2 exhaustive search is also required to generate a minimum number of redundancy-free scheme trees for the $n$-ary case. Nevertheless, we are able to use heuristics similar to those for Algorithm 1.1 to obtain compact scheme trees. Because Algorithm 2 adds edges rather than vertices to a scheme tree, however, the closures must be over the edges rather than the vertices.

Example 21. Consider the canonical, CM hypergraph in Fig. 8b, and let

1. $E_1 = \{\text{Address, Phone}\}$,
2. $E_2 = \{\text{Course, Day, Time}\}$,
3. $E_3 = \{\text{Name, Address, Major}\}$, and
4. $E_4 = \{\text{Course, Name, Address}\}$.

We generate four closures:

1. $E_1^+ = \{\text{Address, Phone}\}$,
2. $E_2^+ = \{\text{Course, Day, Time}\}$,
3. $E_3^+ = \{\text{Name, Address, Major, Phone}\}$, and
4. $E_4^+ = \{\text{Name, Address, Major, Phone, Course}\}$.

We then observe that $E_1$ is in three closures, $E_3$ is in two closures, and both $E_2$ and $E_4$ are in one closure. Therefore, $E_1$, $E_3$, $E_2$, and $E_4$ is an ordering of these edges sorted according to number of appearances in edge closures. For the vertices, we next observe that $\text{Phone}$ and $\text{Address}$ are both in three closures, $\text{Name}$, $\text{Major}$, and $\text{Course}$ are all in two closures while $\text{Day}$ and $\text{Time}$ are in one closure. Sorting first by the number of closures and then by the number of edges in which these vertices appear, $\text{Address}$ comes before $\text{Phone}$, and $\text{Major}$ comes after $\text{Name}$ and $\text{Course}$. Hence, we order the vertices as follows: $\text{Address}$, $\text{Phone}$, $\text{Name}$, $\text{Major}$, $\text{Course}$, $\text{Day}$, and $\text{Time}$. Based on these edge and vertex orderings, we select $E_1$ as the first edge and let $\text{Address}$ be the root node of a new scheme tree $T$. We then graft $E_1$ into $T$. Because $\text{Address} \rightarrow \text{Phone}$, $\text{Phone}$ is added to the root node. The next edge is $E_3$. $E_3$ extends $T$ by creating a list of the remaining vertices of $E_3$ not already in the scheme tree, say $\text{Name}$ first and then $\text{Major}$. It then adds this list to $T$, making $\text{Name}$ a child node of the root node and $\text{Major}$ a grandchild node. Since the next edge in the list, $E_2$, cannot be added to $T$, we select the next edge in the order, which is $E_4$. $\text{Major}$ then becomes another child node of $\text{Name}$. Since $E_2$ cannot be added to $T$, it becomes a scheme tree itself. The generated scheme-tree forest is thus

$\{(\text{Address, Phone, (Name, (Major)*, (Course)*))*},$\n
$(\text{Course, (Time, (Day)*))*}\}$.

Note that we have only two scheme trees, whereas Algorithm 2 can generate three scheme trees, as Example 20 shows. As a postprocessing step, we can combine any parent and child node in a path of a scheme tree if we do not introduce redundancy. We can thus merge the child node $\text{Major}$ with the parent node $\text{Name}$ in the first scheme tree. The resulting scheme tree is $(\text{Address, Phone, (Name, Major, (Course)*)})*$. We can also merge all the nodes in the second scheme tree. The resulting scheme tree is $(\text{Course, Time, Day})*$. \qed

6 Conclusions

We summarize the contributions of this paper as follows:

1. We defined a design methodology that can generate redundancy-free XML documents from a conceptual-model hypergraph.
2. We proved that generating a minimum number of redundancy-free scheme trees from a CM hypergraph is NP-hard.
3. We gave simple heuristics that typically generate a minimum number of scheme trees in practical cases and make the scheme-tree forest as compact as possible without introducing redundancy.
4. We provide two algorithms. One works for the special case that every edge is binary, and the other one works for the general case that allows $n$-ary edges ($n \geq 2$).

As for future work, there are several directions we can pursue:

1. Beyond techniques we have already suggested [11], explore other possibilities for mapping scheme trees to XML DTDs and XML schema specifications.
2. Exploit some of the advanced features of XML-Schema to generate even better schema specifications for XML documents.
3. Do research to discover algorithms that handle CM hypergraphs with generalization/specialization and inclusion dependencies.
4. Beyond the generation tools we have already implemented, build an interactive tool to help users develop XML schema specifications. The tool should alert users to violations of redundancy-free and compactness properties, show them alternatives, and aid them in mapping CM hypergraphs to good, intuitively pleasing XML schemas.

ACKNOWLEDGMENTS

The authors would like to thank the managing editor and the reviewers for their helpful comments. W.Y. Mok was supported in part by the UAH Research Mini-Grant. D.W. Embley was supported in part by the US National Science Foundation under grant numbers 0083127 and 0414644.

REFERENCES