Noname manuscript No.
(will be inserted by the editor)

A graph-theoretic approach to map conceptual designs to XML schemas

Massimo Franceschet · Donatella Gubiani · Angelo Montanari · Carla Piazza

Received: date / Accepted: date

Abstract We propose a mapping from a database conceptual design to a schema for XML enjoying the following properties: information and integrity constraints are preserved, no redundancy is introduced, different hierarchical views of the conceptual information are available, the resulting XML structure is highly connected and nested, and the design is reversible. We investigate two different ways to nest the XML structure: a maximum *density* nesting, that minimizes the number of schema constraints used in the mapping of the conceptual schema, thus reducing the validation overhead, and a maximum *depth* nesting, that keeps low the number of costly join operations that are necessary to reconstruct information at query time using the mapped schema. We propose graph-theoretic linear-time algorithms to find a maximum density nesting in the general case, as well as a maximum depth nesting in the acyclic case. Furthermore, we show that the problem of finding a maximum depth structure in the general case is NP-complete and, notably, it admits no constant ratio approximation algorithm. We complement our investigation with an implementation of the devised translation and an empirical evaluation that shows that the mapping we propose, compared to a flat relational-style design, leads to significant improvements in both query and validation performances. A detailed analysis of related work concludes the paper.

Keywords conceptual data modeling · Entity Relationship model · XML Schema · native XML databases · graph theory · computational complexity

Department of Mathematics and Computer Science, University of Udine, Via delle Scienze 206, 33100 Udine, Italy

1 Introduction

In the information age we are living, an increasing share of information is by nature *unpredictable*, *hierarchical*, and *hybrid*. Unpredictable information defies regular patterns: it is unstructured or, at most, semistructured. Hierarchical information is structured as complex entities which, recursively, might embed other complex entities. Hybrid information mixes both data and text alike. Many advocated the use of *Extensible Markup Language* (XML) to represent information of this nature. This ignited the development of XML databases to store very large data in XML format and to retrieve them with universal and efficient query languages.

An *XML database* is a data persistence software that allows one to store data in XML format. XML databases can be partitioned into two major classes: *XML-enabled databases*, which map XML data to a traditional database (such as a relational database), and *native XML databases*, which define a logical model for an XML document and store and retrieve documents according to it. The design of any database follows a consolidated methodology comprising conceptual, logical, and physical modeling of the data. This paper is a contribution toward the development of design methodologies and tools for native XML databases. In particular, we focus on the mapping from conceptual designs to logical schemas for native XML databases. Specifically, our contributions are the following:

– *Mapping*. We propose a mapping from conceptual designs to schemas for XML. We adopt the well-known *Entity Relationship model* (ER for short) extended with specialization (specialization is particularly relevant in the design of semistructured data), as the conceptual model for native XML databases. Moreover, we opt for *W3C XML Schema Language*

(XML Schema for short) as the schema language for XML. The main alternative to XML Schema is the Document Type Definition (DTD) language, but the latter is strictly less expressive than the former. In particular, DTD lacks expressive means to specify integrity constraints, which are fundamental in database design. The mapping we propose enjoys the following properties: information and integrity constraints of the ER model are preserved, no redundancy is introduced, different hierarchical views of the conceptual information are permitted, the resulting structure is both highly connected and nested, and, finally, the design is reversible;

– *Graph-theoretic investigation of the XML nesting problem.* We provide a graph-theoretic interpretation of the XML nesting problem, that is, the problem of finding the best way to nest the XML elements corresponding to entities and relationships of the ER schema. In this context, we thoroughly study the problems of finding the maximum density nesting forest and the maximum depth nesting forest. The former is a nesting forest with the highest number of edges, which corresponds to the nesting that minimizes the number of schema constraints used in the mapping of the conceptual schema, thus reducing the validation overhead to the minimum. The latter is a nesting forest with the largest value for the summation of node depths. Such a nesting minimizes the number of expensive join operations that are necessary to reconstruct information at query time using the mapped schema, thus reducing the average query evaluation time.
– *System implementation.* We fully implement the devised mapping and we embed it into ChronoGeoGraph [1,2], an ongoing project with the goal of developing a framework for the conceptual and logical design of spatio-temporal XML and relational databases.
– *Experimental evaluation.* We perform an extensive experimental analysis whose output shows that the highly connected and nested structures provided by the developed system have a significant impact on both validation and query performances of XML databases that adhere to the mapped schema. We test both native and XML-enabled databases using data from the XMark benchmark [3].

The paper is organized as follows. Section 2 motivates the use of native XML databases and briefly reviews the major XML technologies. Section 3 illustrates the mapping from conceptual designs to XML schemas. The structure nesting problem is investigated in Section 4. Section 5 describes the implementation and the experimental evaluation of the devised mapping. Related work is discussed in Section 6. Conclusions are drawn in Section 7.

2 Native XML databases

Much information around in these days is *semistructured*, *hierarchical*, and *hybrid* in nature.

Semistructured data has a loose structure (schema): a core of attributes are shared by all objects associated with a semistructured schema, but many individual variants are possible. For instance, consider a bibliography containing references to academic publications. All references have in common a small core of attributes, like authors, title, and publication year. Different reference types, however, have many specific attributes. For instance, books have publishers, journal papers have volume numbers, conference contributions have titles of the proceedings and conference addresses, and theses have hosting institutes. Moreover, some of these attributes might be structured in different ways. For example, we might specify the name of an author as a unique token or as an arbitrarily long list of tokens, including space for possibly multiple first, middle, and last name components.

Hierarchical data is composed of atomic elements and compound elements. Atomic elements have a simple flat content. By contrast, compound elements contain nested sub-elements, either atomic or compound. There is no limit to the nesting level of information. The resulting structure is a hierarchy of information, possibly representable as a tree of objects. For instance, a Web page written in XHTML has a hierarchical structure in which tag elements might contain other tag elements. As another example, consider the biological taxonomic hierarchy in which a species, the most basic rank, nests inside a genus, which in turn nests inside a family, and so on.

The term *hybrid* refers to the fact that information often mixes both data and text alike. For instance, a bibliographic reference might contain records, such as authors, title, year, as well as narrative information, like the paper abstract. Similarly, objects representing people in a social network mix attributes, like name and occupation, with possibly long and structured descriptions, like a CV.

Many researchers have proposed XML as the most appropriate formalism to work with this kind of information. XML, a standard of the World Wide Web Consortium since 1998, has the following unique strengths as a data format [5]:

1. *Simple syntax.* XML is a well-defined format whose documents are easy to create, manipulate, parse by

computer software, and read by humans equipped with a basic text editor. Moreover, XML is portable across different computer architectures and programming languages;
2. *Semistructured data.* The flexibility of the XML data model allows one to represent unstructured information as well as data with a loose schema;
3. *Support for nesting.* The hierarchical nature of XML makes it possible to represent complex structures in a natural way;
4. *Support for hybrid information.* Both data-centric and text-centric information can be easily represented within the same XML document.

Unlike XML, the relational model allows one to store information in a structured and flat fashion. It follows that even simple and highly-related information, like an invoice with date and number accompanied by a list of invoice items (each one with description, quantity, and price attributes), must be spanned over multiple tables and potentially expensive queries, joining the scattered information through the key and foreign key mechanisms, must be used to reconstruct information at query time.

An *XML database* is a data persistence system that allows one to manage data in XML format. It must ensure those features of a traditional database system that are vital for real-world applications, including a universal, efficient, and scalable query language, data integrity constraints, transaction management, data privacy, backup, and recovery. Two major classes of XML databases exist:

(a) *XML-enabled databases.* These systems map XML data to a traditional database, typically a relational database. XML data can be stored in different manners: an XML document can be shredded into relational fields, records and tables, it can be entirely stored as character data in records, or it can be saved as external character large objects.
(b) *Native XML databases.* A native XML database defines a logical model for an XML document, and it stores and retrieves documents according to that method. Examples of such models are the XPath data model, the XML Infoset, and the models underlying DOM and SAX.

A plethora of XML technologies has been developed since XML has been recommended by W3C in 1998. In most cases, these technologies are integrated in an XML database. They include schema definition languages, like DTD and W3C XML Schema, which allow one to define the structure and the integrity constraints of the data, query languages, such as XPath, XQuery, and XQuery Full-Search, which make it possible to express simple element-retrieval queries as well as structured, possibly full-search statements, update languages, like W3C XQuery Update Facility, to perform database updates, transformation languages, such as XSLT, to transform XML documents or query-results retrieved from the database, and programmatic interfaces, like XQuery API for Java, to access an XML database from Java independently from the particular database driver, and Java Architecture for XML Binding, to marshal Java objects into XML and the inverse. A notable example of XML database, implementing most of the mentioned features, is BaseX [6].

3 Mapping conceptual designs to XML schemas

As we already pointed out, the XML data model is both *hierarchical* and *semistructured*: XML elements may be simple elements containing character data or they may nest other child elements, generating a tree-like structure; moreover, elements of the same type may have different structures, e.g., some child elements may be absent or repeated an arbitrary number of times. In the XML encoding of the ER conceptual model we are going to describe, we will deeply exploit the hierarchical and semistructured nature of the XML data model.

3.1 XML schema notation

Representing XML Schema using its own syntax requires substantial space and the reader (and sometimes the developer as well) gets lost in the implementation details. For this reason, we embed ER schemas into a more succinct *XML schema notation* (XSN) whose expressive power lies in between DTD and XML Schema. XSN allows one to specify sequences and choices of elements as in DTD. As an example, the *sequence* definition `author(name, surname)` specifies an element author with two child elements name and surname; the *choice* definition `contact(phone | email)` specifies that the element contact has exactly one child element which is either phone or email. The sequence and choice operators can be combined and nested. For instance, to state that an author can be described either by name, surname, and affiliation or by id and affiliation, we may use the expression `author(((name, surname) | id), affiliation)`. For the sake of simplicity, all pieces of information are encoded using XML elements and the use of XML attributes is avoided.

XSN extends DTD with the following three constructs:

- *Occurrence constraints.* They specify the minimum and maximum number of occurrences of an item. The minimum constraint is a natural number; the maximum constraint is a natural number or the character N denoting an arbitrarily large natural number. The notation is `item[x,y]`, where x is the minimum constraint, y is the maximum constraint, and item is a single element, a sequence, or a choice. We will use the following shortcuts borrowed from DTD:
  - ∗ when both x and y are equal to 1, the occurrence constraint may be omitted, that is, the definition `item` equals `item[1,1]`;
  - ∗ when x = 0 and y = 1, the occurrence constraint may be abbreviated as ?, that is, the definition `item?` equals `item[0,1]`;
  - ∗ when x = 0 and y = N, the occurrence constraint may be abbreviated as *, that is, the definition `item*` equals `item[0,N]`;
  - ∗ when x = 1 and y = N, the occurrence constraint may be abbreviated as +, that is, the definition `item+` equals `item[1,N]`.
- *Key constraints.* If A is an element and KA is a child element (or an attribute) of A, then the notation `KEY(A.KA)` means that KA is a key for A. Keys consisting of more than one element/attribute are allowed. For instance, in `KEY(A.K1, A.K2)` the pair (K1, K2) is a key for A. Moreover, it is possible to define keys over the union of different elements. For instance, the constraint `KEY(A.K | B.K | C.K)` means that the element K must be unique over the union of the domains identified by the elements A, B, and C.
- *Foreign key constraints.* If A is an element with key KA, B is an element, and FKA is a child element (or an attribute) of B, then `KEYREF(B.FKA --> A.KA)` means that FKA is a foreign key of B referring to the key KA of A.

In key and keyref definitions, if A.KA is ambiguous, that is, if there exists another element A with a child element KA in the database, then KA can be prefixed by an unambiguous path to element A in the XML tree.

As an example of XSN schema, suppose we want to specify that a bibliography contains authors and papers. An author has a name and possibly an affiliation. An affiliation is composed of an institute and an address. A paper has a title, a publication source, a year, and one or more authors. Moreover, name is the key for element author, title is the key for element paper, and the author child element of paper is a foreign key referring to the name child element of author. Relevant information is captured by the following concise XSN definition:

```
bibliography((author | paper)*)
  author(name, affiliation?)
    affiliation(institute, address)
  paper(title, source, year, author+)
KEY(author.name), KEY(paper.title)
KEYREF(paper.author --> author.name)
```

It is worth noticing that DTD only allows the specification of [0,1], [0,N], and [1,N] occurrence constraints; moreover, it offers a limited key/foreign key mechanism by using ID-type and IDREF-type attributes, which turns out to be too simple for our goals. For instance, it is not possible to restrict the scope of uniqueness for ID attributes to a fragment of the entire document and only individual attributes can be used as keys.

The mapping of XSN into XML Schema is straightforward: sequence and choice constructs directly correspond to *sequence* and *choice* XML Schema elements; occurrence constraints are implemented with *minOccurs* and *maxOccurs* XML Schema attributes; key and foreign key constraints are captured by *key* and *keyref* XML Schema elements, respectively.

3.2 Mapping ER to XSN

An ER schema basically consists of entities and relationships between them [7]. Both may have attributes, which can be either simple or compound, single-valued or multi-valued. Some entities are weak and are identified by owner entities through identifying relationships. General entities may be specialized into more specific ones. Specializations may be partial or total, disjoint or overlapping. Relationships may involve two or more entities. Each entity participates in a relationship with a minimum and a maximum cardinality constraint. Integrity constraints associated with an ER schema comprise multi-valued attribute occurrence constraints, relationship participation and cardinality ratio constraints, specialization constraints (sub-entity inclusion, partial/total and disjoint/overlapping constraints), as well as key constraints.

The proposed mapping has the following properties:

- it preserves information and as much integrity constraints of the original conceptual schema as possible. An extension to the standard XML Schema validator has been implemented in order to capture the constraints that are lost in the translation due to lack of expressiveness of the XML Schema language;
- it does not introduce any redundancy in the mapped schema: information in the original ER schema is represented only once in the logical XML design;
- it supports different hierarchical views of conceptual information; this allows one to adapt the structure

of the logical schema on the basis of typical (most frequent) transactions of the DBMS;
– it achieves maximum connectivity and deep nesting in the structure used to embed the elements of the conceptual design. As we will show in Section 4, this makes it possible to optimize both validation and query performances;
– it allows us to reverse the design: from the logical XML schema, it is possible to go back to the original conceptual ER schema.

3.2.1 The database element

The first step of the translation is the creation of a *database element*. This is a container for all unnested entity elements of the schema and it corresponds to the document element in an XML instance of the schema. It is defined as an unbounded choice among all unnested entity elements. An instance is the bibliography element in the bibliographic example above.

3.2.2 Entities

Each entity is mapped to an element with the same name. Entity attributes are mapped to child elements. The encoding of compound and multi-valued attributes exploits the flexibility of the XML data model: compound attributes are translated by embedding the sub-attribute elements into the compound attribute element; multi-valued attributes are encoded using suitable occurrence constraints. As opposed to the relational mapping, no restructuring of the schema is necessary.

3.2.3 Relationships

Each binary relationship has two cardinality constraints of the form (x, y), where x is a natural number, that specifies the minimum participation constraint, and y is a positive natural number or the special character N (which represents an arbitrarily large natural number), that specifies the maximum participation (or cardinality ratio) constraint. Typically, x is either 0 or 1, and y is either 1 or N. Hence, we have $2^4 = 16$ possible cases.

Let us consider two entities A, with key KA, and B, with key KB, and a binary relation R between A (conventionally, the left entity) and B (the right entity) with left participation constraint (x_1, y_1) and right participation constraint (x_2, y_2). We denote such a case with the notation $A \overset{(x_1,y_1)}{\longleftrightarrow} R \overset{(x_2,y_2)}{\longleftrightarrow} B$. The encodings for all the typical cases are given in the following order: first one-to-one relationships; then, one-to-many relationships; finally, many-to-many relationships.

1. $A \overset{(0,1)}{\longleftrightarrow} R \overset{(0,1)}{\longleftrightarrow} B$. There are two possible mappings:

```
A(KA,R?)               B(KB,R?)
 R(KB)                  R(KA)
B(KB)                  A(KA)
KEY(A.KA)              KEY(B.KB)
KEY(B.KB)              KEY(A.KA)
KEY(R.KB)              KEY(R.KA)
KEYREF(R.KB-->B.KB)    KEYREF(R.KA-->A.KA)
```

The two mappings are equivalent in terms of number of used constraints. Notice that the constraint `KEY(R.KB)` captures the right maximum participation constraint in the left mapping by forcing the elements KB of R to be unique, that is, each B element is assigned to at most one A element by the relationship R. Similarly for the constraint `KEY(R.KA)` in the right solution. Attributes of the relationship R (if any) are included in the element R that represents the relationship.

2. $A \overset{(0,1)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$. The suggested view is the left one shown below. The element B is fully embedded into element A; hence, no foreign key constraint is necessary and the right minimum cardinality holds by construction. The right maximum cardinality is captured by `KEY(B.KB)`. Notice that the embedding is not possible whenever the right minimum constraint is 0 (as in case 1 above), as, in this case, it would lead to the loss of all B elements that are not associated with any A element. The embedding is not possible whenever the right maximum constraint is greater than 1 as well, as, in this case, it would introduce data redundancy and would violate the key constraint `KEY(B.KB)`. The solution on the right uses an additional key constraint to capture the left maximum cardinality constraints and an extra foreign key constraint; moreover, it looses the chance to nest the resulting structure.

```
A(KA,R?)               B(KB,R)
  R(B)                   R(KA)
    B(KB)              A(KA)
KEY(A.KA),KEY(B.KB)    KEY(B.KB), KEY(A.KA),KEY(R.KA)
                       KEYREF(R.KA-->A.KA)
```

3. $A \overset{(1,1)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$. There exist two symmetric mappings:

```
A(KA,R)                B(KB,R)
  R(B)                   R(A)
    B(KB)                  A(KA)
KEY(A.KA),KEY(B.KB)    KEY(B.KB),KEY(A.KA)
```

An additional flat encoding is possible but not suggested:

```
A(KA,R)
  R(KB)
B(KB)
KEY(A.KA),KEY(B.KB),KEY(R.KB)
KEYREF(R.KB-->B.KB), KEYREF(B.KB-->R.KB)
```

In such a case, the right maximum constraint is coded with `KEY(R.KB)`, while the foreign key constraint `KEYREF(B.KB --> R.KB)` is used to capture the right minimum constraint. The latter claims that each B element must appear inside an R element of A, that is, each B element must be associated with at least one A element. Notice that this foreign key constraint is possible since R.KB is a key (in XML Schema, a foreign key cannot point to something that is not a key).

4. $A \overset{(0,1)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$. The preferred mapping is shown on the left below. The solution on the right uses an extra constraint to capture the left maximum cardinality constraint.

```
A(KA,R?)                   B(KB,R*)
  R(KB)                      R(KA)
B(KB)                      A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB),KEY(A.KA),KEY(R.KA)
KEYREF(R.KB-->B.KB)        KEYREF(R.KA-->A.KA)
```

5. $A \overset{(0,1)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$. The suggested mapping is given on the right below. As in the right solution to the previous case, the constraint `KEY(R.KA)` captures the left maximum cardinality constraint. The left solution does not capture the right minimum cardinality constraint, which must be dealt with as an external constraint (added with clause CHECK). To force such a missing constraint, that is, to constrain each B instance to be associated with at least one A instance, one may be tempted to add the foreign key `KEYREF(B.KB --> R.KB)`. Unfortunately, a foreign key is allowed in XML Schema only if it refers to a key and R.KB cannot be a key: the same B instance can be associated with more than one A instance and thus there may exist repeated B instances under A, a situation that clearly violates the key constraint.

```
A(KA,R?)                   B(KB,R+)
  R(KB)                      R(KA)
B(KB)                      A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB),KEY(A.KA),KEY(R.KA)
KEYREF(R.KB-->B.KB)        KEYREF(R.KA-->A.KA)
CHECK("right min")
```

6. $A \overset{(1,1)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$. The best solution is the right one below that uses the full nesting of elements. The opposite embedding, on the left, needs an extra keyref constraint and it does not achieve full element nesting.

```
A(KA,R)                    B(KB,R*)
  R(KB)                      R(A)
B(KB)                          A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB),KEY(A.KA)
KEYREF(R.KB-->B.KB)
```

7. $A \overset{(1,1)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$. The preferred mapping is the right one below. The opposite embedding fails to capture the right minimum cardinality constraint (which must be dealt with as an external constraint) and it uses an additional foreign key constraint.

```
A(KA,R)                    B(KB,R+)
  R(KB)                      R(A)
B(KB)                          A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB), KEY(A.KA)
KEYREF(R.KB-->B.KB)
CHECK("right min")
```

8. $A \overset{(0,N)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$. There exist two symmetric mappings:

```
A(KA,R*)                   B(KB,R*)
  R(KB)                      R(KA)
B(KB)                      A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB),KEY(A.KA)
KEYREF(R.KB-->B.KB)        KEYREF(R.KA-->A.KA)
```

9. $A \overset{(0,N)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$. The best mapping is the right one. The opposite embedding fails to capture the right minimum cardinality constraint (which must be dealt with as an external constraint).

```
A(KA,R*)                   B(KB,R+)
  R(KB)                      R(KA)
B(KB)                      A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB),KEY(A.KA)
KEYREF(R.KB-->B.KB)        KEYREF(R.KA-->A.KA)
CHECK("right min")
```

10. $A \overset{(1,N)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$. There exist two symmetric mappings:

```
A(KA,R+)                   B(KB,R+)
  R(KB)                      R(KA)
B(KB)                      A(KA)
KEY(A.KA),KEY(B.KB)        KEY(B.KB),KEY(A.KA)
KEYREF(R.KB-->B.KB)        KEYREF(R.KA-->A.KA)
CHECK("right min")         CHECK("left min")
```

It is worth pointing out that both solutions need an external constraint to check the minimum participation constraint: this is the only case in the mapping of relationships where we have to resort to external constraints in the preferred mapping.
An alternative bi-directional solution is the one that pairs the two described mappings:

```
A(KA,R1+)
 R1(KB)
B(KB,R2+)
 R2(KA)
KEY(A.KA),KEY(B.KB)
KEYREF(R1.KB-->B.KB)
KEYREF(R2.KA-->A.KA)
CHECK("inverse relationship")
```

Such a solution captures all integrity constraints specified at conceptual level. It imposes, however, the verification of an additional *inverse relationship constraint*, namely, if an instance x of A is inside an instance y of B, then y must be inside x in the inverse relationship. Such a constraint is not expressible in XML Schema.

Each of the six cases we did not explicitly deal with is the inverse of one of the above-described ones. For instance, $A \overset{(0,N)}{\longleftrightarrow} R \overset{(0,1)}{\longleftrightarrow} B$ is the inverse of case 4.

The general translation pattern for binary relationships can be summarized as follows. The cardinality constraint associated with the entity whose corresponding element includes the element for the relationship, say A, can be forced by `occurs` constraints, while the way in which the cardinality constraint associated with the other entity, say B, is imposed depends on its specific form. Inclusion of the element for B into the element for R suffices for the cardinality constraint $(1,1)$. Such a solution cannot be exploited in the other three cases. All of them require the addition of a `keyref` constraint of the form `KEYREF(R.KB-->B.KB)`. In addition, the cardinality constraint $(0,1)$ needs a `key` constraint of the form `KEY(R.KB)`, and an external constraint of the form `CHECK("right min")` (or of the form `CHECK("left min")`) must be included to capture the cardinality constraint $(1,N)$.

The rules to translate binary relationships can be generalized to relationships of higher degree as well as to relationships with non-typical cardinality constraints, e.g., the constraint $(2,10)$. For instance, let R be a ternary relationship among A, B, and C, where A participates in R with constraint $(1,N)$, B participates in R with constraint $(0,1)$, and C participates in R with constraint $(1,1)$. The preferred mapping is the following one:

```
A(KA,R+)
  R(KB,C)
    C(KC)
B(KB)
KEY(A.KA),KEY(B.KB),KEY(C.KC)
KEY(R.KB)
KEYREF(R.KB-->B.KB)
```

As a general rule, the translation of a relationship R of degree n, with $n > 2$, has the following structure: the outermost element corresponds to an entity that participates in R with cardinality constraint $(1,N)$. If there is not such an entity, we choose an entity that participates with cardinality constraint $(0,1)$. If there is not such an entity as well, we choose one that participates with cardinality constraint $(0,N)$. If all entities participate with cardinality constraint $(1,1)$, we will choose one of them. Then, the element corresponding to R is nested in the outermost element and it includes the elements, or the references to the elements, corresponding to all the other entities. Such a translation avoids whenever possible the addition of external constraints and it minimizes the number of internal constraints.
It goes without saying that, as an alternative, we can preliminarily apply reification to replace every relationship of higher degree by a corresponding entity related to each participating entity by a suitable binary relationship, and then exploit the translation rules for binary relationships.

It is worth pointing out that, thanks to its hierarchical nature, the XML logical model allows one to capture a larger number of constraints specified at conceptual level than the relational one. For all cardinality constraints of the form $(1,N)$, indeed, there is no way to preserve the minimum cardinality constraint 1 in the mapping of ER schemas into relational ones (as we will see later on, the same happens with specializations [7]).

3.2.4 Weak entities and identifying relationships

A weak entity always participates in the identifying relationship with cardinality constraint (1,1). Hence, depending on the form of the second cardinality constraint, one of the cases discussed above applies. The key of the element for the weak entity is obtained by composing the partial key with the key of the owner entity; moreover, the owner key in the element for the weak entity must match the corresponding key in the element for the owner entity. For instance, suppose we have $A \overset{(0,N)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$, where B is weak and owned by A. The translation is:

```
A(KA,R*)
  R(B)
    B(KB, KA)
KEY(A.KA),KEY(B.KB, B.KA)
CHECK(B.KA=A.KA)
```

It is worth pointing out that the external constraint `CHECK(B.KA = A.KA)` cannot be avoided. Indeed, suppose we remove the owner key KA from the element for the weak entity B, thus obtaining:

```
A(KA,R*)
  R(B)
    B(KB)
KEY(A.KA),KEY(B.KB,A.KA)
```

Unfortunately, the key constraint `KEY(B.KB, A.KA)` cannot be expressed in XML Schema: on the one hand, if we point the selector of the key schema element at the level of the element A, then the field pointing to KB is not valid, since it selects more than one node (A may be associated with more than one B element if the relationship is one-to-many). On the other hand, if we point the selector at the level of the element B, then the field referring to KA is not valid as well, since it must use the parent or ancestor axes to ascend the tree, but such axes are not admitted in the XPath subset supported by XML Schema.[1]

Such a translation can be generalized to weak entities with identifying relationship of degree greater than 2 and with more than one identifying relationships.

[1] Apparently, the authors of [8] repeatedly missed this point.

3.2.5 Specialization

The hierarchical nature of the XML data model can be fully exploited in the mapping of specializations. Let us consider a parent entity A, with key KA, and two child entities B, with attributes attB, and C, with attributes attC. If the specialization is partial-overlapping, then the possible mappings are the following:

```
A(KA,B?,C?)         A(KA)
  B(attB)           B(KA,attB)
  C(attC)           C(KA,attC)
KEY(A.KA)           KEY(A.KA),KEY(B.KA),KEY(C.KA)
                    KEYREF(B.KA-->A.KA)
                    KEYREF(C.KA-->A.KA)
```

The preferred solution is the left one, where both B and C elements are embedded inside the element A. Neither key nor foreign key constraints are necessary. The partial-overlapping constraint is captured by using the occurrence specifiers: an element A may contain any subset of {B, C}.
If the specialization is total-overlapping, the mappings are as follows (in flat solution to the right, the entity A is discarded since the specialization is total):

```
A(KA, (B,C?) | C)   B(KA,attB)
  B(attB)           C(KA,attC)
  C(attC)           KEY(B.KA),KEY(C.KA)
KEY(A.KA)
```

If the specialization is partial-disjoint, the mappings are as follows:

```
A(KA, (B|C)?)       A(KA)
  B(attB)           B(KA,attB)
  C(attC)           C(KA,attC)
KEY(A.KA)           KEY(A.KA),KEY(B.KA|C.KA)
                    KEYREF(B.KA-->A.KA)
                    KEYREF(C.KA-->A.KA)
```

The key constraint `KEY(B.KA | C.KA)` in the right solution forces disjointness: the value for KA must be unique across the union of B and C domains.
Finally, a total-disjoint specialization can be encoded as follows:

```
A(KA, (B|C))        B(KA,attB)
  B(attB)           C(KA,attC)
  C(attC)           KEY(B.KA|C.KA)
KEY(A.KA)
```

The generalization to specializations involving $n > 2$ child entities is immediate in all cases except for the total-overlapping case. Let $a_1, \ldots, a_n$ be the child entities of a total-overlapping specialization. We indicate with $\rho(a_1, \ldots, a_n)$ the regular expression allowing all non-empty subsets of child entities. Such an expression can be recursively defined as follows:

$$\rho(a_1, ..., a_n) = \begin{cases} a_1 & \text{if } n = 1 \\ (a_1, a_2?, ..., a_n?)|\rho(a_2, ..., a_n) & \text{if } n > 1 \end{cases}$$

```
publication(title, year, citations, reference*,
            authorship+, (article | book)?)
  reference(title)
  authorship(name, contribution)
  article(pages, abstract, (journal | conference))
    journal(name, volume)
    conference(name, place)
  book(ISBN)
publisher(name, address, publishing+)
  publishing(title)
author(name, affiliation+)
  affiliation(institute, address)

KEY(publication.title), KEY(publisher.name)
KEY(author.name), KEY(publishing.title)
KEYREF(reference.title --> publication.title)
KEYREF(authorship.name --> author.name)
KEYREF(publishing.title --> publication.title)
```

Fig. 2 The mapping of the citation-enhanced bibliographic database.

The size of the expression $\rho(a_1, \ldots, a_n)$ is $n \cdot (n + 1)/2$. Furthermore, the regular expression is deterministic, in the sense that its standard automata-theoretic translation is a deterministic automaton. This is relevant since both DTD and XML Schema content models must be deterministic.

As in the case of higher-degree relationships, we can actually replace specialization of a parent entity into k child ones by k total functional binary identifying relationships (one for each child entity), and then apply translation rules for weak entities and identifying relatonships.

So far we have discussed simple specializations: each child entity inherits from exactly one parent entity. In multiple specializations, a child entity may have more than one parent entity. Multiple specializations break the above nesting strategy. If only simple specializations are used, the resulting structure is a tree that can be naturally embedded in the tree-like XML data model. On the contrary, that for multiple specializations is a directed acyclic graph, which cannot be directly dealt with such a data model. However, to encode multiple specializations, we can use a flat encoding similar to the relational mapping [7]. This approach uses key and foreign key constraints to encode the inclusion constraints between child and parent entities and it avoids key duplications in the child entity when the parent entities have a common ancestor in the specialization lattice.

We conclude the section with a simple example: the mapping of the ER schema given in Figure 1, which describes a citation-enhanced bibliography (a typical semi-structured data instance), is reported in Figure 2. The XML Schema version is available at the ChronoGeoGraph web site [1].

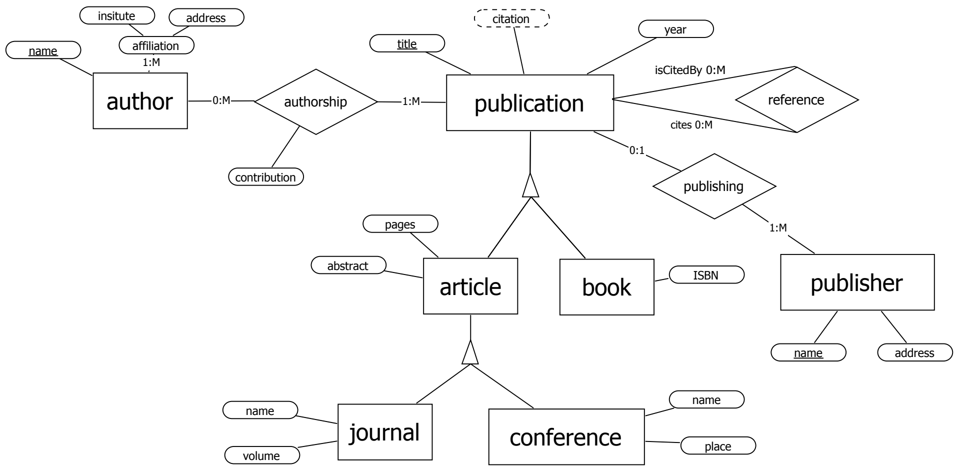

Fig. 1 A citation-enhanced bibliographic database.

4 The XML nesting problem

In the following, we present an algorithm that maps ER schemas into highly-nested XML Schema documents. To keep the algorithm as simple as possible, we preliminarily restructure the ER schema by removing higher-order relationships and specializations. Every relationship R with degree k greater than 2 is replaced by a corresponding entity E_R and k total functional binary relationships linking E_R to the entities participating in R. Every specialization of a parent entity E into k child entities $E_1, \ldots, E_k$ is replaced by k total functional binary (identifying) relationships linking E to the (weak) entities $E_1, \ldots, E_k$.

Then, translation rules described in the previous section are applied to the elements of the restructured ER schema. For each ER construct, the choice of the specific translation rule to apply depends on the way in which the construct occurs in the schema. Indeed, we do not translate ER constructs in isolation, but an ER schema including a number of related constructs. Consider, for instance, the case of an entity E that participates in two relationships R_1 and R_2 with cardinality constraints $(1, 1)$. As the element for E cannot be included both in the element for R_1 and in that for R_2, the preferred translation rule can be applied to one of the relationships only, while for the other relationship we must resort to the alternative translation rule. As we will see, in order to select the relationship to which the preferred translation rule must be applied, the proposed algorithm takes into account the effects of the different choices on the nesting degree of the resulting XML structure.

Increasing the nesting degree of the XML structure has two main advantages. The first one is the reduction of the validation overhead thanks to the reduction of the number of constraints in the mapped schema; the second one is the decrease in the number of (expensive) join operations needed to reconstruct information at query time: highly nested XML documents can be better exploited by tree-traversing XML query languages like XPath. As a source of exemplification, consider the following example. Suppose we want to model a one-to-one relationship *manages* between an entity *manager*, with attributes *ssn* and *name* (*ssn* is the key), and an entity *department*, with attributes *name* and *address* (*name* is the key). A manager manages exactly one department and a department is directed by exactly one manager. Given the corresponding ER fragment, the XML Schema documents produced by the flat and nested mappings are the following:

```
// flat mapping
manager(ssn, name, manages)
  manages(name)
department(name, address)
KEY(manager.ssn)
KEY(department.name)
KEY(manages.name)
KEYREF(department.name  --> manages.name)
KEYREF(manages.name --> department.name)

// nested mapping
manager(ssn, name, manages)
  manages(department)
    department(name, address)
KEY(manager.ssn)
KEY(department.name)
```

Notice that nesting saves three constraints over five (one key and two foreign keys). Furthermore, suppose we want to retrieve the address of the department directed by Bob Strunk. Two XPath versions of this query are given below, one working over the flat schema and the other tailored to the nested one:

```
// flat mapping
```

```
/department[name=/manager[name="Bob Strunk"]/manages
           /name]/address

// nested mapping
/manager[name="Bob Strunk"]/manages/deparment/address
```

The first version of the query joins, in the first filter, the name of the current department and the name of the department directed by Bob Strunk. This amounts to jump from the current node to the tree root. The second version of the query fluently traverses the tree without jumps. Similarly if the query is written in XQuery. We expect the second version of the query to be processed more efficiently by XML query processors.

According to the rules for the translation of binary relationships given in Section 3, nesting comes into play in the translation of total functional relationships only, that is, relationships such that (at least) one of the participating entities has cardinality constraint $(1, 1)$. As a general policy, the translation algorithm introduces a nesting whenever possible. However, as we already pointed out, a conflict arises when an entity participates in two or more relationships with cardinality constraint $(1, 1)$ (*nesting confluences*). In addition, *nesting loops* may occur. Both nesting confluences and nesting loops must be broken to obtain a hierarchical nesting structure. We call the problem of finding the best nesting structure that eliminates nesting confluences and nesting loops the *nesting problem.* In the following, we provide a graph-theoretic formalization of such a problem and we propose and contrast possible solutions to it.

Let S be the restructured ER schema. We build a directed graph (digraph for short) $G = (V, E)$, whose nodes are the entities of S that participate in some total functional binary relationship and $(A, B) \in E$ if there is a total functional relationship R relating entities A and B such that B participates in R with cardinality constraint $(1, 1)$. The direction of the edges models the entity nesting structure, that is, $(A, B) \in E$ if (the element for) entity A contains (the element for) entity B. We call G the nesting graph of S. A nesting confluence corresponds to a node in the graph with more than one predecessor and a nesting loop is a graph cycle.

A *spanning forest* is a subgraph F of G such that: (i) F and G share the same node set; (ii) in F, each node has at most one predecessor; (iii) F has no cycles. A *root* in a forest is a node with no predecessors. The *depth* of a node in a forest is the length of the unique path from the root of the tree containing the node to the node. Notice that a root has depth 0. The nesting problem can be formalized in terms of the following two problems: (i) given a digraph G, find a spanning forest with the maximum number of edges (*maximum density problem*), and (ii) given a digraph G, find a spanning forest with the maximum sum of node depths (*maximum depth problem*). Both problems always admit a solution, which is not necessarily unique. The reader might wonder if a spanning forest with maximum density is also a spanning forest with maximum depth. Unfortunately, the answer in negative, as shown by the example depicted in Figure 3.

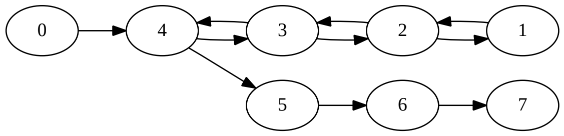

Fig. 3 A maximum density spanning forest for the given digraph is obtained by removing edges (1,2), (2,3), and (3,4). It consists of one tree, with 7 edges, and the sum of node depths is 19. A maximum depth spanning forest is the simple path from node 1 to node 7 plus node 0. It consists of 2 trees, with 6 edges in total, and the sum of node depths is 21. Both solutions are unique.

It is not difficult to see that the maximum depth problem is close to the Hamiltonian path problem. Given a graph G, the Hamiltonian path problem over G is the problem of deciding whether there exists a path in G that visits each node exactly once. The Hamiltonian path problem is NP-complete (see, e.g., [9]). We first show that the Hamiltonian path problem can be reduced to the maximum depth one. From the NP-completeness of the former, it immediately follows that the latter is hard and, unless $P = NP$, there exists no efficient algorithm that solves it.

Theorem 1 *Let G be a digraph. The maximum depth problem for G is NP-complete.*

Proof Let us introduce some notations. Given a digraph G and a spanning forest F for G, we denote by S_F the sum of node depths in F. We say that F is a chain if $|V| - 1$ nodes in F have one child (successor) and 1 node is a leaf.

We first prove that given a digraph G and a spanning forest F for G, it holds that:

(1) $S_F \leq \frac{|V|\cdot(|V|-1)}{2}$;
(2) if F is not a chain, then $S_F < \frac{|V|\cdot(|V|-1)}{2}$.

Let $n = |V|$. Depths of nodes in F range from 0 to $n-1$, and thus we may partition F nodes as follows: k_0 nodes at depth 0, k_1 nodes at depth $1, \ldots, k_{n-1}$ nodes at depth $n-1$, where $\sum_{j=0}^{n-1} k_j = n$ and $\sum_{j=0}^{n-1} k_j \cdot j = S_F$.

We prove by induction on the depth $i \in [0, n-1]$ that if S_F is the maximum possible value over all the graphs having n nodes, then $k_i = 1$.

Base Case ($i = 0$). Nodes at depth 0 do not contribute to S_F. Hence, it is convenient to have the minimum possible number of such nodes. Since there must be at least one root in F, the value for k_0 which maximizes S_F is 1. By contradiction, suppose that G is the complete digraph over n nodes. If F is a spanning forest with more than one node at depth 0, then we can find F' such that $S_{F'} > S_F$ as follows: we choose one of the nodes at depth 0 and we add to F all the edges going from that node to the other nodes at depth 0.

Inductive Step ($0 < i \leq n-1$). We assume that $k_0 = k_1 = \ldots = k_{i-1} = 1$, and we prove that $k_i = 1$. By inductive hypothesis, there is one node at depth $i-1$. Moreover, since $i \leq n-1$, by the inductive hypothesis, it also holds that $\sum_{j=0}^{i-1} k_j \leq n-1$. Since we cannot have nodes at depth greater than i without having at least one node at depth i, we can conclude that there is at least one node at depth i. Let G be the complete digraph over n nodes. If F is a spanning forest with more than one node at depth i, say, $v_1^i, \ldots, v_k^i$, then we can find F' such that $S_{F'} > S_F$ as follows: we choose one node at depth i, say v_1^i, we remove from F all the edges connecting the (only) node at depth $i-1$ to $v_2^i, \ldots, v_k^i$, and we add all the edges going from v_1^i to $v_2^i, \ldots, v_k^i$. In such a way, $v_2^i, \ldots, v_k^i$, as well as all their descendants, increase their depth by 1 and there are not nodes whose depth is decreased. This allows us to conclude that, in order to maximize S_F, k_i must be equal to 1.

It immediately follows that $S_F \leq \sum_{j=0}^{n-1} 1 \cdot j = \frac{n \cdot (n-1)}{2}$ (item (1)).

Item (2) easily follows as well. We have shown that, in order to get the maximum possible value for S_F, that is, $\frac{n \cdot (n-1)}{2}$, there must be exactly one node at depth i, for each i from 0 to $n-1$, which amounts to say that F must be a chain. Hence, if F is not a chain, we get $S_F < \frac{n \cdot (n-1)}{2}$ (item (2)).

We now prove that given a digraph G, the following problems are equivalent:

(i) G has an Hamiltonian path;
(ii) every solution F of the maximum depth problem for G is such that $S_F = \frac{|V| \cdot (|V|-1)}{2}$;
(iii) every solution F of the maximum depth problem for G is a chain.

In order to prove the equivalence, we show that (i) implies (ii), (ii) implies (iii), and (iii) implies (i).
(i) implies (ii). If G has an Hamiltonian path H, then H is a spanning forest for G. Moreover, since H is a chain, $S_H = \frac{|V| \cdot (|V|-1)}{2}$. As, by item (1), $\frac{|V| \cdot (|V|-1)}{2}$ is an upper bound over all the possible spanning forests, it follows that H is a solution of the maximum depth problem, and thus all the solutions have sum of depths $\frac{|V| \cdot (|V|-1)}{2}$.
(ii) implies (iii). It immediately follows from item (1) and item (2) (by contraposition).

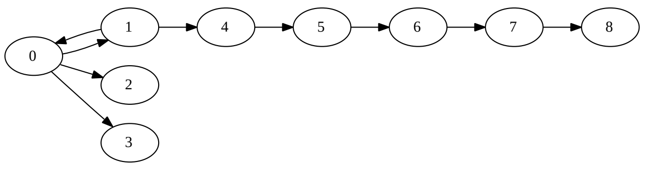

Fig. 4 A maximum depth spanning forest for the given digraph can be obtained by removing edge (1,0). It consists of one tree, whose sum of node depths is 23. Its maximum out-degree is 3. A minimum out-degree spanning forest can be obtained by removing edge (0,1). It consists of one tree whose sum of node depths is 20 and maximum out-degree is 2.

(iii) implies (ii). If every solution of the maximum depth problem is a chain, then we can extract at least one chain from G, and any such chain is an Hamiltonian path for G.

Hence, we have that G has an Hamiltonian path if and only if $S_F = \frac{|V| \cdot (|V|-1)}{2}$ for every solution F of the maximum depth problem for G. As S_F can be computed from F in polynomial time, it immediately follows that the maximum depth problem for G is NP-hard.

To conclude, let us consider the problem of deciding whether a digraph has a spanning forest of depth k. It is easy to see that such problem is in NP, since given a spanning forest F, S_F can be computed in polynomial time. As S_F has an upper bound which is polynomial in the size of the digraph (item 1), the corresponding optimization problem, that is, the maximum depth problem for G, is in NP. □

As a matter of fact, the maximum depth problem is close to various other problems studied in the literature. As an example, the problem of finding a spanning tree whose maximum out-degree (number of children of a node) is minimum is a generalization of the Hamiltonian path problem and different approximation algorithms have been proposed to solve it (see, e.g., [10, 11]). One may expect a spanning forest with minimum out-degree to be a maximum depth spanning forest, and vice versa. Unfortunately, this is not the case, as shown by the digraph in Figure 4. In [12], it has been shown that, given an indirected graph, the problem of finding a longest path is not constant approximable in polynomial time, unless $P = NP$. In [13], such a result is extended to the case of cubic Hamiltonian graphs. In [14], the above results are exploited to prove that, given an indirected graph, the problem of finding a spanning tree with maximum sum of distances from a specified root is not constant approximable in polynomial time, unless $P = NP$. The maximum depth problem we are interested in differs from such a problem in three respects: (i) it refers to digraphs, (ii) spanning forests, instead of spanning trees, are considered, and (iii) there is not an input root. However, the result in [14] can be easily generalized to the case in which the graph is connected and the root is not given in input, as it builds on the results reported in [13], where the graphs are Hamiltonian (and, thus, connected) and there is not an input root. Hence, it holds that, given a connected undirected graph, the problem of finding a rooted spanning tree which has maximum sum of distances from its root (*undirected maximum depth problem*) is not constant approximable in polynomial time, unless $P = NP$. Theorem 2 shows that such a result can actually be tailored to digraphs.

As a preliminary step, we prove a meaningful property of strongly connected digraphs. The strongly connected components of a digraph are maximal sets of mutually reachable nodes. A digraph is strongly connected if it consists of one strongly connected component.

Lemma 1 *Let $G = (V, E)$ be a strongly connected digraph $G = (V, E)$ such that $(u, v) \in E$ if and only if $(v, u) \in E$. It holds that if F is a maximum depth spanning forest for G, then F is a tree.*

Proof Let us assume, by contradiction, that F consists of n trees $T_1, \ldots, T_n$, with $n > 1$. Since G is strongly connected, there is at least one node r in T_1 which reaches at least one node s belonging to T_j, for some $j \neq 1$, that is, $(r, s) \in E$. Let h be the depth of r in T_1, k be the depth of s in T_j, S_r be the subtree of T_1 rooted at r, and S_s be the subtree of T_j rooted at s. If $h \geq k$, then we can remove S_s from T_j and add it to T_1 using the edge (r, s). In such a way, we get a new forest where the depths of the nodes belonging to S_s are increased, while all the other depths remain unchanged. This contradicts the assumption that F is a maximum depth spanning forest. Otherwise ($h < k$), from $(r, s) \in E$, it immediately follows that $(s, r) \in E$ as well. Hence, the same argument can be applied: we remove S_r from T_1 and add it to T_j using the edge (s, r). Again, this contradicts the assumption that F is a maximum depth spanning forest. □

Theorem 2 *Unless $P = NP$, there is no constant ratio approximation algorithm for the maximum depth problem.*

Proof Let μ be the function that maps any undirected graph G into a corresponding digraph $\mu(G)$ such that, for all $u, v \in V$, there exists a pair of edges (u, v) and (v, u) in $\mu(G)$ if (and only if) there exists an edge between u and v in G. Moreover, let π be the function that maps any undirected rooted tree T into a corresponding directed rooted tree $\pi(T)$ such that, for all $u, v \in V$, there exists an edge (u, v) in $\pi(T)$ if (and only if) there exists an edge between u (the parent) and v (the child) in T. Clearly, π is a bijection. Let G be an undirected graph and T be a rooted spanning tree for G with sum of depths equal to t. We have that $\pi(T)$ is a rooted spanning tree for $\mu(G)$ with sum of depths equal to t. Moreover, if S is a rooted spanning tree for $\mu(G)$ with sum of depths equal to s, then $\pi^{-1}(S)$ is a rooted spanning tree for G with sum of depths equal to s. Finally, if G is connected, then $\mu(G)$ is strongly connected and hence, by Lemma 1, each maximum depth spanning forest for $\mu(G)$ consists of a single tree. Hence, the existence of a constant ratio approximation algorithm for the maximum depth problem would imply the existence of a constant ratio approximation algorithm for the undirected maximum depth problem, and this last may exist only if $P = NP$. □

We now focus our attention on the relationships between the maximum depth problem and the maximum density one. Let us consider the case of Directed Acyclic Graphs (DAG). As a matter of fact, the digraph depicted in Figure 4, showing that maximum density spanning forests are in general different from maximum depth ones, is not a DAG. We can ask ourselves whether the same may happen with DAGs. It is easy to show that there exist maximum density spanning forests which do not maximize the sum of node depths even if the graph is a DAG. Nevertheless, the next theorem shows that, for any given DAG, a maximum depth spanning forest is also a maximum density spanning forest.

Theorem 3 *Let $G = (V, E)$ be a DAG and let F be a maximum depth spanning forest for it. Then, F is a maximum density spanning forest for G.*

Proof If F is a tree, then the thesis immediately follows. Let F consist of $n > 1$ trees $T_1, \ldots, T_n$ with roots $r_1, \ldots, r_n$, respectively. Suppose, by contradiction, that F does not maximize density. Then, there exists a spanning forest F' consisting of m trees $S_1, \ldots, S_m$, with $m < n$. By the pigeonhole principle, there exist $i, j \leq n$ and $k \leq m$ such that both r_i and r_j belong to S_k. Hence, at least one between r_i and r_j is not the root of S_k. Without loss of generality, we may assume that r_i is not the root of S_k. Let p be the predecessor of r_i in S_k. The edge (p, r_i) is in S_k and hence in G. Since G is a DAG and (p, r_i) is an edge of G, p does not belong to the tree T_i of F rooted at r_i. Hence, if we add to the forest F the edge (p, r_i), we get a new forest F'' with $n-1$ trees. In F'', each node belonging to $F \setminus T_i$ has the same depth as in F, while each node belonging to T_i increases his depth by $depth_F(p) + 1$, where $depth_F(p)$ is the depth of p in F. Hence, F'' has a depth greater than F, which contradicts the hypothesis that F is a maximum depth spanning forest. □

Nodes of a DAG can be partitioned into different strata using a suitable notion of rank (see, e.g., [15]).

Definition 1 Let $G = (V, E)$ be a DAG. For each $v \in V$, we define $rank_G(v)$ as follows:

$$rank_G(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ max\{rank_G(u) + 1 \mid (v, u) \in E\} & \text{otherwise} \end{cases}$$

According to Definition 1, the rank of a node v is the length of the longest path from v to a leaf. Now, given a DAG G, let G^{-1} be the DAG obtained by reversing all the edges of G. For all $v \in G$, $rank_{G^{-1}}(v)$ is the length of the longest path in G from a root to v. Hence, $rank_{G^{-1}}(v)$ can be viewed as the maximum depth at which we can push v in a spanning forest for G.

In the following, we provide a linear time algorithm, called Maximum_Density, that solves the maximum density problem for digraphs as well as the maximum depth problem for the subclass of DAGs. Given a digraph G, the algorithm first computes the graph H of its strongly connected components. The nodes of H are the strongly connected components of G and (C_j, C_i) is an edge of H if and only if there exist $u \in C_j$ and $v \in C_i$ such that (u, v) is an edge in G. H is always a DAG, and Maximum_Density operates on it by taking advantage of the notion of rank. Maximum_Density consists of the following steps:

1. compute the graph H of the strongly connected components of G (let $C = \{C_1, \dots, C_n\}$ be the set of nodes of H);
2. compute a maximum density spanning forest $K = (C, E_K)$ for H as follows:
   (i) compute H^{-1} and, for each node C_i, the rank $rank_{H^{-1}}(C_i)$;
   (ii) for each node C_i in H, if C_i is not a root node in H, then pick a node C_j such that (C_j, C_i) is in H and $rank_{H^{-1}}(C_j) = rank_{H^{-1}}(C_i) - 1$ and add the edge (C_j, C_i) to E_K;
3. compute a set of edges E' as follows: for each edge $(C_j, C_i) \in E_k$, pick an edge (u, v) such that $(u, v) \in E$, $u \in C_j$ and $v \in C_i$ and add (u, v) to E';
4. for each strongly connected component C_i of G:
   (a) if there is an edge (u, v) in E' with v in C_i, then compute a tree $T_i = (C_i, E_i)$ rooted at v and spanning C_i;
   (b) else pick a node v in C_i and compute a tree $T_i = (C_i, E_i)$ rooted at v and spanning C_i;
5. output the forest $F = (V, E' \cup E_1 \cup E_2 \cup \ldots \cup E_n)$.

Lemma 2 *The spanning forest K generated by step 2 of the algorithm Maximum_Density is a maximum depth spanning forest for H.*

Proof By Definition 1, we have that if C_i is not a root, then there exists at least one node C_j such that (C_j, C_i) is in H and $rank_{H^{-1}}(C_j) = rank_{H^{-1}}(C_i) - 1$. Moreover, K is a spanning forest for H, since for each node C_i of H, K contains at most one incoming edge (C_j, C_i). We show that K maximizes the depth by proving that for each node C_i of H, C_i has maximum depth in K, that is, its depth is equal to $rank_{H^{-1}}(C_i)$. We proceed by induction on $rank_{H^{-1}}(C_i)$. If $rank_{H^{-1}}(C_i) = 0$, then C_i is a leaf in H^{-1} and a root in H. In such a case, the maximum depth for C_i in any spanning forest for H is 0, as it has no incoming edges in H. Hence, C_i is a root in K, that is, it is at depth 0 in K, and thus the thesis holds. Let us assume that the thesis holds for all nodes of rank at most h and let $rank_{H^{-1}}(C_i) = h + 1$. At step 2, Maximum_Density picks a node C_j such that (C_j, C_i) is in H and $rank_{H^{-1}}(C_j) = h$. By the inductive hypothesis, C_j is at depth h in K, and thus we have that C_i is at depth $h + 1$ in K. □

Since H is a DAG, from Theorem 3, it follows that K is also a maximum density spanning forest for H.

Theorem 4 *Let G be a digraph. The algorithm Maximum_Density computes a maximum density spanning forest for G in linear time.*

Proof First, we observe that, given a spanning forest F, every root in F has no incoming edges and any node in F which is not a root has exactly one incoming edge. Hence, given a digraph G with n nodes, a spanning forest F for G has $n - k$ edges if and only if it has k roots. It immediately follows that the maximum density problem is equivalent to the problem of finding a spanning forest with the minimum number of roots.

We prove the thesis by induction on the number of strongly connected components of G.

Basic case. If G has one strongly connected component only, then H has one node and no edges, and thus K has no edges and E' is empty. Maximum_Density picks a node v of G and it computes a tree T rooted at v and spanning G. Hence, Maximum_Density outputs the tree T, which is a maximum density spanning forest.

Inductive step. Let us assume the thesis to be true for digraphs with n strongly connected components and let G be a digraph with $n + 1$ strongly connected components $C_1, \dots, C_{n+1}$. Since H is a DAG, at least one node in H is a leaf. Without loss of generality, we assume C_1 to be a leaf. Let $G \setminus C_1$ be the subgraph of G obtained by removing all nodes in C_1 and all edges involving nodes in C_1. Since C_1 is a leaf in H, its removal does not affect the computation, that is, the forest F is a possible output of Maximum_Density on G if and only if the forest $F \setminus C_1$, obtained by removing all nodes in C_1 and all edges involving nodes in C_1, is a possible output of Maximum_Density on $G \setminus C_1$. By the inductive hypothesis, the thesis holds for $G \setminus C_1$. Let k be the number of roots of the maximum density spanning forest for $G \setminus C_1$ computed by Maximum_Density (which is also a spanning forest with the minimum number of roots). Two cases are possible:

(a) in G there exists an edge (u, v) with $u \notin C_1$ and $v \in C_1$;
(b) there exist no such edges in G.

In case (a), since C_1 is a leaf and it is reachable from at least one (other) strongly connected component, a maximum density spanning forest for G is a spanning forest for G having k roots. In this case, by Lemma 2, we have that the spanning forest K for H generated by step 2 of the algorithm is a maximum depth one, and thus C_1 is not a root of K. Hence, the execution of step 3 of the algorithm adds an edge (u, v) to E' for some $v \in C_1$. Then, at the subsequent step, the algorithm computes a tree T_1, rooted at v, which spans C_1. By the inductive hypothesis, Maximum_Density is correct on $G \setminus C_1$, and thus $(V \setminus C_1, (E' \setminus \{(u, v)\}) \cup E_2 \cup \ldots \cup E_{n+1})$ has k roots. Since (u, v) is in E' and T_1 is rooted at v, $(V, E' \cup E_1 \cup E_2 \cup \ldots \cup E_{n+1})$ has k roots, that is, Maximum_Density is correct on G.
In case (b), since C_1 is an isolated node in H, a maximum density spanning forest for G is a spanning forest for G having $k + 1$ roots. In this case, Maximum_Density picks a node v in C_1 and it computes a tree T_1, rooted at v, which spans C_1. By inductive hypothesis, Maximum_Density is correct on $G \setminus C_1$, and thus $(V \setminus C_1, E' \cup E_2 \cup \ldots \cup E_{n+1})$ has k roots. Hence, $(V, E' \cup E_1 \cup E_2 \cup \ldots \cup E_{n+1})$ has $k + 1$ roots, that is, Maximum_Density is correct on G.

To prove that Maximum_Density has a linear time complexity, it suffices to observe that: (i) H can be computed in linear time by exploiting Tarjan's algorithm; (ii) H^{-1} can be computed in linear time exploiting a visit over H; (iii) all ranks over H^{-1} can be computed in linear time by taking advantage of a DFS-visit over H^{-1} (see [15]); (iv) the edges of K can be chosen in linear time by exploiting a visit over H; (v) each edge (C_j, C_i) of K can be replaced by a suitable edge (u, v) in constant time, by keeping pointers to the edges of G; and (vi) since a spanning tree of C_i can be computed in time linear in the size of C_i, the time required to compute $T_1, \ldots, T_n$ is linear in the size of G. □

By Theorems 1 and 2, we know that there is no guarantee about the goodness of the spanning forest computed by Maximum_Density with respect to the maximum depth problem. There are two critical aspects: (i) the use of the acyclic graph of strongly connected components, and (ii) the problem of determining a maximum depth spanning tree for any given strongly connected component. Let us consider the digraph in Figure 3. It features 5 strongly connected components, namely, $C_1 = \{0\}$, $C_2 = \{4, 3, 2, 1\}$, $C_3 = \{5\}$, $C_4 = \{6\}$, and $C_5 = \{7\}$. The DAG of its strongly connected components consists of one tree, rooted at C_1, with edges (C_1, C_2), (C_2, C_3), (C_3, C_4), and (C_4, C_5) which are replaced by edges $(0, 4)$, $(4, 5)$, $(5, 6)$, and $(6, 7)$, respectively, at step 3. Then, a tree T_2, rooted at node 4, spanning C_2 is computed, which contains the edges $(4, 3)$, $(3, 2)$, and $(2, 1)$. All the other trees, namely, T_1, T_3, T_4, and T_5, consist of a single node. The maximum density spanning forest returned by Maximum_Density consists of a single tree, whose sum of depth is 19. As we already pointed out, the maximum depth spanning forest for the digraph in Figure 3 consists of 2 trees and has depth 21. The situation is different if we restrict our attention to DAGs.

Theorem 5 *Let G be a DAG. The algorithm Maximum_Density computes a maximum depth spanning forest for G in linear time.*

Proof Since G is a DAG, each strongly connected component of G consists of a single node and thus G is isomorphic to H. Hence, from Lemma 2, it immediately follows that Maximum_Density computes a maximum depth spanning forest for G. □

To make the translation algorithm more flexible, we introduce a constrained variant of the considered problems that gives the designer the possibility to impose the application of the preferred translation rule to some relationships, e.g., those involved in frequently asked/dominant queries (see Section 6). Formally, this amounts to force the maintenance of some edges of the original digraph. The constrained variants of the maximum density and maximum depth problems are defined as follows. Given a digraph G and a set of its edges C, find a spanning forest, containing all edges in C, with the maximum number of edges (*constrained maximum density problem*), and find a spanning forest, containing all edges in C, with the maximum sum of node depths (*constrained maximum depth problem*). Obviously, the constrained versions of the problems may lack a solution. As an example, if the edges in C form a loop, then there is not a solution. Moreover, the solution of the constrained version does not necessarily coincide with that of the original problem, as shown in Figure 5.

As a preliminary step, we identify the conditions which ensure the existence of a solution. Let C be a set of edges, a *confluence* in C is a pair of edges of C with the same target node, that is, a pair of edges (u, v) and (w, v), for some u, v, w in G.

Lemma 3 *Let $G = (V, E)$ be a digraph and $C \subseteq E$. The constrained maximum density (resp., depth) prob-*

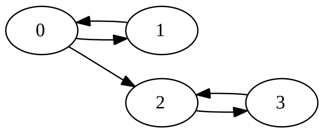

Fig. 5 Different solutions to the maximum density problem for the given digraph exist, each one consisting of 1 tree with 3 edges. None of them contains the edge (3,2). A maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each.

lem has a solution if and only if neither loops nor confluences occur in C.

Proof On the one hand, it is easy to check that if C contains a loop or a confluence, then there exists no forest F including all edges in C. On the other hand, if neither loops nor confluences occur in C, the digraph $G' = (V, C)$ is a spanning forest for G, and thus the problems have a solution. □

The complexity of the constrained maximum depth problem is the same of its unconstrained version.

Theorem 6 *Let $G = (V, E)$ be a digraph and $C \subseteq E$. The constrained maximum depth problem for G and C is NP-complete. Moreover, unless $P = NP$, there is no a constant ratio approximation algorithm for it.*

Proof NP-hardness immediately follows from Theorem 1 (take $C = \emptyset$). To show that it is in NP, consider the problem of deciding whether a digraph has a spanning forest of depth k containing all edges in C. Such a problem is in NP, since given a spanning forest F, both computing its depth and checking that it contains all edges in C can be done in polynomial time. Hence, since the depth of F has an upper bound which is polynomial in the size of the graph size (see the proof of Theorem 1), it follows that the corresponding optimization problem, that is, the constrained maximum depth problem, is in NP. The last part of the thesis is an immediate consequence of Theorem 2 (take $C = \emptyset$). □

We now show that the constrained maximum density problem can be effectively reduced to the maximum density one. Let Constrained_Maximum_Density be the following algorithm, which takes a digraph $G = (V, E)$ and a set $C \subseteq E$ as input:

1. check that C contains neither loops nor confluences; otherwise, stop with failure (it has no solution);
2. compute the set of target nodes $T = \{v | \exists (u, v) \in C\}$ in C;
3. compute the graph $\overline{G} = (V, \overline{E})$ such that $(u, v) \in \overline{E}$ iff $(u, v) \in C \vee (v \notin T \wedge (u, v) \in E)$;
4. apply Maximum_Density to $\overline{G}$ (let $\overline{F}$ be the output it produces);
5. for each edge $(u, v) \in C$, if $(u, v) \notin \overline{F}$, then let (r, s) be an edge on the path from v to u in $\overline{F}$ such that $(r, s) \notin C$. Replace (r, s) by (u, v) in $\overline{F}$;
6. output the forest $\overline{F}$.

Theorem 7 *Let $G = (V, E)$ be a digraph and let $C \subseteq E$. Constrained_Maximum_Density solves the constrained maximum density problem for G and C in linear time.*

Proof First, by Lemma 3, we have that the algorithm terminates before step 4 if and only if the problem has no solution. Second, since $\overline{G}$ is a subgraph of G with the same set of nodes as G, the spanning forest $\overline{F}$ computed by step 4 is a spanning forest for G as well. Third, step 5 does not modify the number of edges.

Let $\overline{F}$ be the output of the algorithm. We show that (i) $\overline{F}$ is a spanning forest for G, and (ii) it is a maximum density spanning forest for G under the constraint that it must include all edges in C.

As far as item (i) is concerned, let $\overline{F}$ be the spanning forest computed by step 4 and let $(u, v) \in C$ be such that $(u, v) \notin \overline{F}$. We prove that v is a root of $\overline{F}$ and u belongs to the tree rooted at v. By contradiction, suppose that v is not a root of $\overline{F}$. Hence, v has a predecessor in $\overline{F}$. Since (u, v) is the only edge in $\overline{G}$ entering v, the predecessor of v in $\overline{F}$ must be u, against the hypothesis that $(u, v) \notin \overline{F}$ (contradiction). Now, again by contradiction, suppose that u does not belong to the tree rooted at v. Let $\overline{F}'$ be $\overline{F} \cup \{(u, v)\}$. The addition of (u, v) to $\overline{F}$ introduces neither confluences (v has no predecessors in $\overline{G}$, and thus in $\overline{F}$, different from u) nor cycles (there is not a path from v to u in $\overline{F}$), and thus $\overline{F}'$ is a spanning forest for $\overline{G}$ with more edges than $\overline{F}$, which is a maximum density spanning forest for $\overline{G}$ (contradiction). The existence of an edge $(r, s) \notin C$ in the path from v to u immediately follows from the fact that we execute step 5 only if C has no cycles. Hence, each iteration of the for-loop in step 5 replace a spanning forest for G by another one with the same number of edges.

As far as item (ii) is concerned, suppose, by contradiction, that there exists a spanning forest F' for G containing all edges in C with a number of edges greater than the spanning forest $\overline{F}$ returned by the algorithm. Since F' contains all edges in C, F' is also a spanning forest for $\overline{G}$. Hence, there exists a spanning forest for $\overline{G}$ with a number of edges greater than the number of edges of the spanning forest produced by step 4 (contradiction).

As for the complexity, Maximum_Density works in linear time and all the other steps have linear time complexity. Hence, Constrained_Maximum_Density has linear time complexity. □

We conclude the section by showing that in the case of DAGs, Constrained_Maximum_Density also computes a constrained maximum depth spanning forest.

Lemma 4 *Let $G = (V, E)$ be a DAG and $C \subseteq E$ which does not contain confluences. Let $T = \{v \mid \exists(u, v) \in C\}$ and $\overline{G} = (V, \overline{E})$ be such that $(u, v) \in \overline{E}$ iff $(u, v) \in C \vee (v \notin T \wedge (u, v) \in E)$. If $\overline{F}$ is a solution of the maximum depth problem for $\overline{G}$, then $\overline{F}$ is also a solution of both the constrained maximum depth problem and the constraint maximum density problem for G and C.*

Proof Since G is a DAG, $\overline{G}$ is a DAG. Moreover, each spanning forest for $\overline{G}$ is a spanning forest for G. We show that all edges $(u, v) \in C$ belong to $\overline{F}$. By contradiction, suppose that there exists $(u, v) \in C$ such that $(u, v) \notin \overline{F}$. Since (u, v) is the only edge entering v in $\overline{G}$ and $(u, v) \notin \overline{F}$, v is a root of $\overline{F}$. Now, let T_v be the tree of $\overline{F}$ rooted at v. Since $(u, v) \in \overline{E}$ and $\overline{G}$ is a DAG, u does not belong to T_v. Hence, $\overline{F}' = \overline{F} \cup \{(u, v)\}$ is a spanning forest for $\overline{G}$ and its depth is greater than that of $\overline{F}$, against the hypothesis that $\overline{F}$ is a maximum depth spanning forest for $\overline{G}$ (contradiction).

We prove now that $\overline{F}$ has maximum depth over all spanning forests for G containing all edges in C. By contradiction, suppose that there exists a spanning forest F for G, containing all edges in C, whose depth is greater than that of $\overline{F}$. F is also a spanning forest for $\overline{G}$, against the hypothesis that $\overline{F}$ is a maximum depth spanning forest for $\overline{G}$ (contradiction).

Finally, we show that $\overline{F}$ has maximum density over all spanning forests for G containing all edges in C. By contradiction, suppose that there exists a spanning forest F for G containing all edges in C, whose density is greater than that of $\overline{F}$. F is also a spanning forest for $\overline{G}$. However, since $\overline{F}$ is a maximum depth spanning forest for $\overline{G}$, by Theorem 3, $\overline{F}$ is also a maximum density spanning forest for $\overline{G}$ (contradiction). □

Theorem 8 *Let $G = (V, E)$ be a DAG and let $C \subseteq E$. Constrained_Maximum_Density solves the constrained maximum depth problem for G and C in linear time.*

Proof If G is a DAG, then also $\overline{G}$ is a DAG. Hence, by Theorem 5, Maximum_Density computes a maximum depth spanning forest for $\overline{G}$. By Lemma 4, the output of step 4 is also a solution to the constrained maximum depth problem for G and C (it contains all edges in C). Hence, step 5 does nothing and the output of Constrained_Maximum_Density is also a maximum depth spanning forest for G. □

5 Implementation and experimental evaluation

We have developed two implementation modules in the Java programming language. The first module, the translator, implements the mapping from ER to XML Schema. It takes the (restructured) ER schema and the associated maximum density spanning forest as input and it returns the corresponding XML Schema document. Each ER construct is associated with a Java class containing a method translating the conceptual construct to a corresponding schema fragment. Those (few) integrity constraints that are not captured within XML Schema, due to lack of expressiveness of the schema language, are annotated using the *appinfo* element of XML Schema. The translator takes advantage of JDOM package to create the schema elements and to serialize the schema document.

The second module is the validator. The validation is split in two parts: the check of those constraints that are expressible in XML Schema, which is performed using the validation package provided in Java API for XML Processing (JAXP, the standard API for XML included in the Java platform), and the verification of additional constraints not expressible in XML Schema, which is implemented by retrieving the annotated constraints with the aid of JDOM and by checking them with the additional code that we implemented.

The two modules have been integrated into ChronoGeoGraph (CGG) [16], a software framework for the conceptual and logical design of spatio-temporal databases. The core of the framework is the CGG model, a conceptual model that extends the ER model with additional constructs for spatio-temporal information [17, 18]. The CGG tool has a graphical interface that allows one to draw CGG conceptual schemas. Moreover, it has a working module that translates a CGG conceptual schema into a relational logical schema and a separate module, on which the devised translation has been embedded, that maps the CGG conceptual schema into XML Schema.

In the following, we present and discuss the outcomes of the experimentation of (the proposed extension of) CGG on a typical XML benchmark. As expected, we have that both validation and query processing are (often significantly) more efficient on nested designs than on corresponding flat schemas. Ultimately, our experiments empirically prove that there exists a real advantage in using the semistructured and hierarchical data model of XML, instead of the structured and flat relational data model.

The experimental setup is the following. We take advantage of the well-known XML benchmark XMark [3]. XMark models an Internet auction web site in which

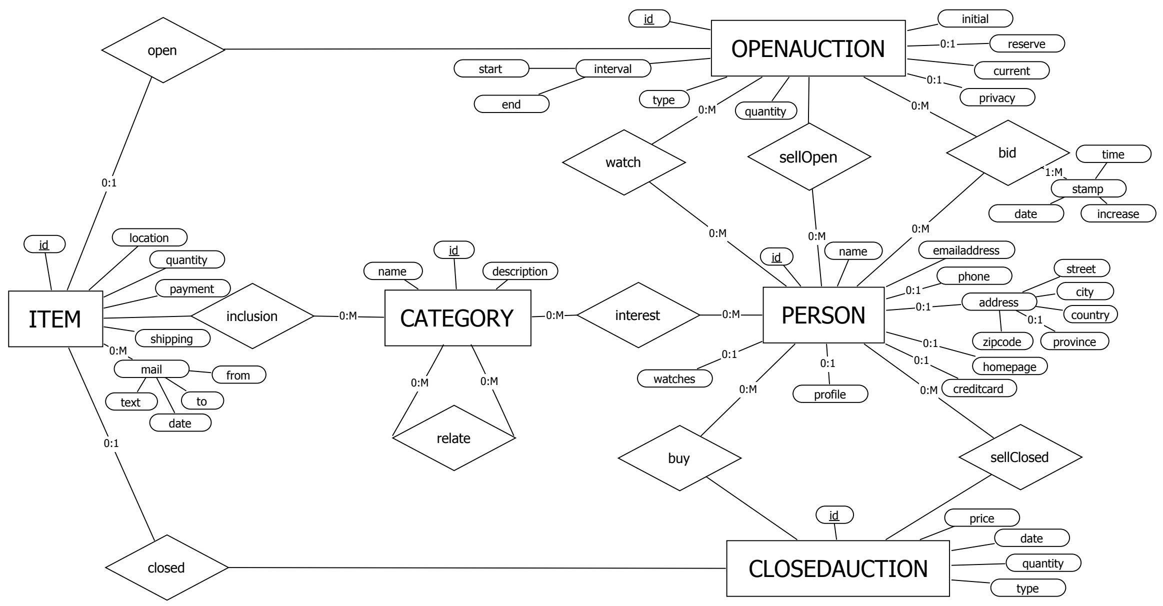

Fig. 6 The XMark conceptual schema.

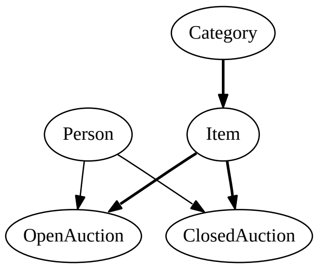

Fig. 7 The XMark nesting graph. The maximum depth nesting forest is highlighted in bold.

people watch ongoing open auctions, biding for the items on sale. People may also sell and buy items, and they may declare their interests with respect to categories of items. When an item is sold in an auction, the auction is declared closed with a given price. A (simplified) ER schema for XMark is depicted in Figure 6, and the corresponding nesting graph is shown in Figure 7 (as a matter of fact, our simplified ER schema includes all meaningful entities and relationships of the original XMark design).

Starting from the XMark conceptual design, we obtained two schemas: a nested, hierarchical-style XMark schema (Figure 8), which exploits the nesting of XML elements as much as possible (we used the maximum depth nesting forest depicted in Figure 7), and a flat, relational-like XMark schema (Figure 9), in which each entity of the conceptual design is encoded at the same level of the XML hierarchy and conceptual relationships are mapped using the key and foreign key mechanisms

```
// element definitions
site((Category | Person)*)
Category(id, inclusion*, relate*)
  relate(categoryref)
  inclusion(Item)
    Item(id, open?, closed?)
      open(OpenAuction)
        OpenAuction(id, sellOpen, bid*)
          sellOpen(personref)
          bid(personref, stamp)
            stamp(date, time, increase)
      closed(ClosedAuction)
        ClosedAuction(id, buy, sellClosed)
          buy(personref)
          sellClosed(personref)
Person(id, interest*, watch*)
  interest(categoryref)
  watch(openauctionref)
// key constraints
KEY(Category.id)
KEY(Item.id)
KEY(OpenAuction.id)
KEY(ClosedAuction.id)
KEY(Person.id)
// foreign key constraints
KEYREF(sellOpen.personref --> Person.id)
KEYREF(bid.personref --> Person.id)
KEYREF(buy.personref --> Person.id)
KEYREF(sellClosed.personref --> Person.id)
KEYREF(interest.categoryref --> Category.id)
KEYREF(watch.openauctionref --> OpenAuction.id)
KEYREF(relate.categoryref --> Category.id)
```

Fig. 8 A nested XMark schema in XSN.

```
// element definitions
site((Category|Item|Person|OpenAuction|ClosedAuction)*)
OpenAuction(id, open, sell, bid*)
 open(itemref)
 sellOpen(personref)
 bid(personref, stamp)
   stamp(date, time, increase)
ClosedAuction(id, closed, buy, sell)
 closed(itemref)
 buy(personref)
 sellClosed(personref)
Item(id, inclusion)
 inclusion(categoryref)
Category(id, relate*)
  relate(categoryref)
Person(id, interest*, watch*)
  interest(categoryref)
  watch(openauctionref)
// key constraints
KEY(OpenAuction.id)
KEY(ClosedAuction.id)
KEY(open.itemref)
KEY(closed.itemref)
KEY(Item.id)
KEY(Category.id)
KEY(Person.id)
// foreign key constraints
KEYREF(open.itemref --> Item.id)
KEYREF(closed.itemref --> Item.id)
KEYREF(inclusion.categoryref --> Category.id)
KEYREF(relate.categoryref --> Category.id)
KEYREF(interest.categoryref --> Category.id)
KEYREF(watch.openauctionref --> OpenAuction.id)
KEYREF(sellOpen.personref --> Person.id)
KEYREF(bid.personref --> Person.id)
KEYREF(buy.personref --> Person.id)
KEYREF(sellClosed.personref --> Person.id)
```

Fig. 9 A flat XMark schema in XSN.

(the resulting XMark flat schema is very close to the original DTD for XMark). Both schemas contain the same information and capture all conceptual integrity constraints.

The XMark benchmark includes a scalable data generator that produces well-formed, meaningful XML documents that are valid with respect the XMark schema. The user can control the size of the generated document using a scaling parameter, where scale 1 corresponds to a document of 100 MB. We took advantage of the data generator to produce XML instances of increasing size, starting from scaling factor 0.001 (100 KB) up to 1 (100 MB). We mapped these XML instances into corresponding instances for the nested and flat designs, using Java classes that we coded.

We ran all experiments on a 2.53 GHz machine with 2.9 GB of main memory running Ubuntu 9.10 operating system. During the early elaborations of our tests, we took advantage of the benchmarking platform XCheck-Java [19].

As for validation, we measured the validation time of the generated XML instances with respect to both the flat and nested XMark schemas, using the validation package included in the Java API for XML Processing. All documents have been parsed using the event-based SAX method. We expressed the selector XPath query of the key and keyref constraints of XML Schema either using the child axis (/) or using the descendant axis (//). The resulting elapsed times, expressed in seconds, are shown in Table 1. Validating nested designs is more efficient than validating flat schemas. This was expected, since, thanks to the hierarchical structure, the nested design has fewer constraints (5 keys and 7 foreign keys) compared to the flat schema (7 keys and 10 foreign keys). Furthermore, expressing constraints using the child axis instead of the blind descendant modality makes validation faster on hierarchical schemas, while the effect is negligible on flat designs. This might be a useful guidance for database designers.

As for query performance, we devised a benchmark comprising four significant queries in the XQuery language. Each query is encoded in two instances, a flat version for the flat XMark schema, and a nested version, that works over the nested XMark schema. The benchmark queries are as follows:

Q1. *Categories and the items they contain.* The flat version of this query, shown below, performs a join operation between items and categories:

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
for $item in $doc/site/Item
where $item/inclusion/categoryref = $category/id
return
<result>
 {$category/id}
 {$item/id}
</result>
```

On the other hand, the nested version fluently traverses the XML tree without joins, exploiting the fact that, in nested XML instances, items are embedded inside categories, ready for use:

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
for $item in $category/inclusion/Item
return
<result>
 {$category/id}
 {$item/id}
</result>
```

Q2. *Categories and the open auctions bidding items belonging to these categories.* The flat version of this query performs two joins: a first join between categories and items, and a second one between items and open auctions:

scale	flat//	nest//	flat	nest
0.001	0.41	0.36	0.39	0.38
0.005	0.65	0.57	0.63	0.69
0.010	0.96	0.90	0.86	0.90
0.050	1.64	1.61	1.59	1.45
0.100	2.53	2.15	2.31	2.27
0.500	25.01	21.99	24.77	19.66
1.000	83.09	73.22	82.62	67.46

Table 1 Validation performance. The columns labeled with flat (nest) refer to validation against the flat (nested) XMark schema; the suffix // means the the key and foreign key constrains are expressed using the descendant axis instead of the child one.

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
let $item := for $i in $doc/site/Item
         where $i/inclusion/categoryref=$category/id
         return $i
for $auction in $doc/site/OpenAuction
where $auction/open/itemref = $item/id
return
<result>
 {$category/id}
 {$auction/id}
</result>
```

The nested version fully exploits the hierarchical structure of the nested instances in which open auctions are embedded inside items, which are in turn nested inside categories:

```
let $doc := doc("xmark.xml")
for $category in $doc/site/Category
for $auction in
    $category/inclusion/Item/open/OpenAuction
return
<result>
 {$category/id}
 {$auction/id}
</result>
```

Q3. *The open and corresponding closed auctions.* The flat version joins open and closed auctions as follows:

```
let $doc := doc("xmark.xml")
for $open in $doc/site/OpenAuction
for $closed in $doc/site/ClosedAuction
where $closed/closed/itemref = $open/open/itemref
return
<result>
 {$open/id}
 {$closed/id}
</result>
```

The nested version combines descendant and ancestor axes to draw a seamless path in the nested instance: the query first descends to open auctions, backtracks up to the corresponding items, and finally falls down toward the associated closed auctions. The query exploits the fact that both open and closed auctions are nested inside items:

```
let $doc := doc("xmark.xml")
for $open in $doc//OpenAuction
for $closed in $open/ancestor::Item//ClosedAuction
return
<result>
 {$open/id}
 {$closed/id}
</result>
```

Q4. *People and the closed auctions bidding items bought by these people.* The flat instance of the query makes a join between people and closed auctions, which are both unnested, top-level entities:

```
let $doc := doc("xmark.xml")
for $people in $doc/site/Person
for $auction in $doc/site/ClosedAuction
where $auction/buy/personref = $people/id
return
<result>
 {$people/id}
 {$auction/id}
</result>
```

The nested version makes a similar join between people and closed auctions. While person is a top-level entity in the nested schema, closed auctions are nested inside items and inside categories, hence they must be located in the hierarchy before the join operation can start. Hence, this query disadvantages, in principle, the nested version of the schema.

```
let $doc := doc("xmark.xml")
for $people in $doc//Person
for $auction in $doc//ClosedAuction
where $auction/buy/personref = $people/id
return
<result>
 {$people/id}
 {$auction/id}
</result>
```

We tested the devised benchmark on three open-source XML query engines: BaseX (version 6) [6], Saxon (release B 9.1.0.8 for Java) [20], and MonetDB/XQuery (release 4) [21]. BaseX is a native XML database, Saxon is a native processor for XSLT and XQuery, and MonetDB/XQuery is a XML-enabled database which maps XML into the relational data model. It is worth stressing that our goal here is to compare query performance

BaseX	**Q1**		**Q2**		**Q3**		**Q4**	
scale	**nest**	**flat**	**nest**	**flat**	**nest**	**flat**	**nest**	**flat**
0.001	0.00	0.01	0.00	0.01	0.00	0.00	0.02	0.01
0.005	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01
0.010	0.01	0.01	0.01	0.02	0.01	0.01	0.02	0.02
0.050	0.04	0.04	0.03	0.06	0.01	0.02	0.06	0.05
0.100	0.07	0.06	0.05	0.11	0.02	0.02	0.09	0.10
0.500	0.28	0.29	0.19	0.80	0.08	0.22	0.70	0.83
1.000	0.55	0.58	0.37	1.81	0.15	0.56	1.71	1.88

Table 2 Query evaluation performance: BaseX

Saxon	**Q1**		**Q2**		**Q3**		**Q4**	
scale	**nest**	**flat**	**nest**	**flat**	**nest**	**flat**	**nest**	**flat**
0.001	0.89	0.91	0.88	0.89	0.87	0.86	0.91	0.86
0.005	1.03	1.05	1.06	1.10	1.00	1.16	1.18	1.22
0.010	1.15	1.32	1.14	1.41	1.15	1.49	1.59	1.56
0.050	1.61	1.90	1.63	2.08	1.54	2.07	2.30	2.29
0.100	1.64	2.21	1.87	2.58	1.81	2.69	3.75	3.59
0.500	2.54	7.07	2.88	14.52	2.80	22.86	65.85	47.70
1.000	3.50	19.99	4.10	46.23	3.90	86.15	264.34	173.95

Table 3 Query evaluation performance: Saxon

MDB/XQ	**Q1**		**Q2**		**Q3**		**Q4**	
scale	**nest**	**flat**	**nest**	**flat**	**nest**	**flat**	**nest**	**flat**
0.001	0.05	0.08	0.06	0.12	0.04	0.08	0.03	0.03
0.005	0.04	0.08	0.06	0.12	0.04	0.08	0.03	0.03
0.010	0.04	0.08	0.06	0.12	0.06	0.09	0.04	0.04
0.050	0.04	0.09	0.06	0.13	0.06	0.10	0.04	0.05
0.100	0.05	0.09	0.07	0.14	0.07	0.11	0.05	0.05
0.500	0.05	0.18	0.10	0.22	0.13	0.16	0.10	0.09
1.000	0.05	0.25	0.15	0.34	0.18	0.23	0.16	0.15

Table 4 Query evaluation performance: MonetDB/XQuery

on different designs, and not query performance on different engines. We are benchmarking schema designs, not query processors. Tables 2, 3, and 4 show the query evaluation performance, expressed in seconds of elapsed time, for the three benchmarked engines, respectively. We can draw the following conclusions.

1. Queries Q1, Q2, and Q3 are significantly more efficient in their nested version on hierarchical documents. For instance, the time performance ratio for query Q2 between the flat and the nested largest XML instance is 4.9 for BaseX, 11.3 for Saxon, and 2.3 for MonetDB/XQuery. These queries use an *unfolding* technique to join related information: the linked information is just few steps away from in the XML tree, and hence it can be retrieved by unfolding few XML elements and directly accessing their content. On the other hand, the flat versions of these queries interpreted on the unnested instances use a *product* approach to combine information that is far away in the XML tree, alighted on different immediate sub-trees of the database element, in a relational-style manner.
2. On the other hand, it is not clear which schema structure, flat or nested, is more efficient for query Q4. Recall that this query is designed to favour the flat schema structure, because the entities it links are immediately accessible top-level elements in flat instances, while they must be located down in the hierarchical structure in nested documents. MonetDB/XQuery makes no difference between the two structures, BaseX performs slightly better on the nested structure, while Saxon is significantly faster on the flat schema.

6 Related work

There is a vast literature on the relationships between XML and relational databases, which ranges from expressiveness issues to performance comparison, going through data import/export from/to XML/relational databases. In particular, a general comparison of XML and relational constructs and an analysis of the basic kinds of mapping between them can be found in [22]. Unfortunately, this literature is partly redundant (you find very similar analyses in a number of contributions) and not well linked (the corresponding citation network is made of disconnected components). We identified three main research themes related to our work: (i) the encoding of (standard) conceptual schemas into some XML schema language, (ii) the mapping of relational schemas into XML schemas, and (iii) the development of new conceptual models tailored to XML databases.

As for the first theme, even though some work has been done to provide an XML encoding of other formalisms for conceptual modeling, such as, for instance, UML [23] and ORM [24], most contributions refer to the ER conceptual model. For this reason, we restrict our attention to the XML encoding of the (enhanced) ER model. The various proposals differ in a number of respects, including the choice of the target XML schema language (DTD or XML Schema), the set of constraints encoded by the conceptual schema they cope with (cardinality constraints on the participation of entities in relations, constraints on specializations) as well as the way in which the constraints they encode are dealt with, and the correspondence they establish between the constructs of the conceptual model and those of the XML target language (XML elements vs. attributes, XML ID/IDREF vs. KEY/KEYREF).

The problem of mapping relational schemas into XML ones has been extensively and successfully addressed in the literature, e.g., [25–27]. From the point of view of information preservation, converting a relational (logical) schema into an XML one is easier than converting an ER (conceptual) schema into it. The set of constructs of the ER model, in particular, the integrity constraints that it allows one to express, is indeed a superset of those of the relational model. Moreover, the generated XML schemas usually mirror the flat schemas of the source relational databases as opposed to what happens with the conceptual design. The problem of transforming relational schemas into highly nested XML schemas has been addressed by a few contributions, e.g., [8, 28]. All proposed solutions are based on a direct or indirect classification of database relations. As an example, Zhou et al. [28] partitions the set of relations in base, single-related, and multi-related relations on the basis of their foreign key constraints: base relations are devoid of foreign keys, single-related ones feature exactly one foreign key, and multi-related ones have two or more foreign keys [28]. One of the most elaborated solutions, which aims at providing a highly-nested XML counterpart of relational schemas, is the one proposed by Liu et al. [8]. They preliminarily map the relational schema into a graph, whose nodes and edges correspond to the tables and the referential integrity constraints of the original schema, respectively. The resulting graph provides a clean characterization of critical points such as nesting confluences and nesting cycles. However, the authors provide no general solution, leaving any design decision to the user, who is supposed to be guided by information about dominant queries. A different approach is followed by Duta et al. [29] and by Fong and Cheung in [30]. They first map relational schemas back to ER schemas, and then they translate the generated conceptual schemas into XML schemas, taking advantage of existing mapping rules. As an example, Duta et al. [29] propose two algorithms to encode relational schemas in XML Schema preserving the source relationships and their structural constraints. The first one, called ConvRel, translates each relationship individually into a nested XML structure; the second one, called Conv2XML, identifies complex nested structures able to capture all possible relationships in a relational database in a uniform way. They also provide an ordered list of parameters to measure the relative quality of mappings from relational schemas to XML ones, namely, constraint-preservation, nested structure, compact structure, length of generated XML file, similarity to the relational structure (the first one being the most significant one).

The last related theme is the development of ad-hoc conceptual models, tailored to XML databases, that allow the developer to express at conceptual level conditions that are usually dealt with by XML models only, such as, for instance, ordering conditions on attributes and alternative attributes. Most proposals essentially extend standard conceptual models with XML features. This is the case, for instance, with the ERX model [31] and the XSEM model [32], that extend the ER model (the proposal of the XSEM model is based on the comparative analysis of existing models given in [33]), and with the UXS model, which is based on UML [34]. Other proposals define new hierarchical conceptual models, whose constructs closely resemble those of XML schema languages. This is the case, for instance, with the ORA-SS model [35] and with the XML tree model [36]. Most contributions pair the proposal of a new conceptual model with a translation algorithm that maps the conceptual schema into the corresponding XML schema.

Our work presents significant intersections with research about relational-to-XML mappings and XML-inspired conceptual models. However, the closest research theme is definitely that about the XML encoding of (standard) conceptual schemas. For this reason, in the following we restrict our attention to it.

6.1 Encoding of conceptual schemas in XML

The first mappings from conceptual models to XML proposed in the literature take DTD as their target language [23,37]. Despite its simplicity and conciseness, DTD is not expressive enough to capture some relevant database integrity constraints, such as, for instance, domain constraints, composite keys, and arbitrary concept cardinalities [22]. To get rid of these drawbacks, most recent encodings replace DTD by XML Schema.

Regardless of the choice of DTD or XML Schema as the target language, existing proposals for an XML encoding of conceptual schemas can be evaluated with respect to different parameters. The basic and most important ones are preservation of information (to what extent concepts and constraints expressed by means of conceptual schemas are preserved by the corresponding XML schemas) and nesting degree of the resulting XML schemas (the length of nested element chains). Additional quality parameters have been proposed in the literature [38], including (absence of) redundancy, conformity with applications, and design reversibility. In fact, they are not independent from the basic ones, and thus we will not discuss them separately. The most interesting one is *design reversibility.* An XML schema satisfies such a condition if it allows one to reconstruct the original conceptual schema. Design reversibility turns out to be useful in data integration and reverse engineering. In general, full design reversibility cannot be guaranteed without annotations, as there are constraints encoded by conceptual schemas that cannot be preserved by XML ones. The use of annotations for design reversibility has been analyzed in some detail in [37]. In the following, we first focus our attention on the preservation of information (Section 6.1.1) and then on the nesting degree of the resulting XML schemas (Section 6.1.2).

6.1.1 Preservation of information

A large variety of XML encodings of conceptual schemas can be found in the literature. No relevant differences can be found in their treatment of basic constructs. On the contrary, there is not a consensus mapping for advanced constructs, e.g., specialization, and constraints, e.g., cardinality constraints on relations. Furthermore, many proposals disregard some important constructs and constraints (for instance, many-to-many relationships with total participation of both entities). The translation we presented in Section 3 can be viewed as the merge of a number of existing proposals, coping with all distinctive features of the ER model in a systematic and detailed way. In the following, we briefly survey different translations of the most problematic constructs and constraints, namely, cardinality constraints on the participation of entities in relations, weak entity types, and constraints on specializations.

Relationship. The translation of relationships is more complex than that of entities as one must take into account all their distinctive features (one-to-one, one-to-many, or many-to-many binary relationships, binary relationships or relationships with degree greater than 2, total or partial participation of entities in relationships, and so on). In addition, choices that influence the nesting of the corresponding XML elements have a considerable impact on validation and query processing. As a result, different mappings of relationships have been proposed in the literature.

The translation of binary functional relationships R is quite standard: the element corresponding to (one of) the entity with participation constraint $(1, 1)$, say A, is nested in the element corresponding to R, which, in its turn, is nested in the element corresponding to the other entity, say B [37,29,39]. Minor variants to such a translation schema have been proposed in [38], where the relative positions of R and A are exchanged, and in [40], where there is not an element for R, thus loosing design reversibility. In the specific case of one-to-one relationships, some translations map them into a unique element (merge) [37,40].

Non-functional (binary) one-to-many relationships R between two entities A and B are dealt with by adding a reference to the element corresponding to the entity with participation constraint $(0, 1)$, say A (and not directly the element, as in the functional case), in the element corresponding to R, which, in its turn, is nested in the element corresponding to the entity with participation constraint $(\_, N)$, say B. The same solution is often applied to the case of non-functional (binary) many-to-many relationships. As both entities participate in the relationship with maximum cardinality equal to N, there is the problem of establishing which one must be mapped into the B-element (resp., A-element). Different criteria have been proposed in the literature. In [39], the B-element corresponds to an entity with participation constraint $(1, N)$ (if any). Such a solution aims at minimizing the number of additional (external) constraints. In [38], the choice of the B-element is based on the notion of dominant entity, where a dominant entity is the entity from which most accesses to R start. An equivalent solution, based on the notion of general access frequency, is provided in [40]. The dominance/frequency approach aims at increasing the performance of query evaluation, and it may possibly yield to the loss of some constraints. An alternative mapping of many-to-many relationships has been proposed in [37], where the element corresponding to the relationship includes references to the elements corresponding to the participating entities. A non-trivial limitation of such a solution is that it cannot enforce total participation of entities in relationships. Finally, a mixed solution can be found in [29]. The mapping of many-to-many relationships with partial participation of entities is the same as in [37], while the treatment of the other cases is the same as in [39], apart from the replacement of references by elements. Such a replacement trades the absence of redundancy and the enforcement of key constraints for the achievement of nested and compact structures, that is, the authors accept the presence of some redundancies and the lack of some key constraints in order to increase the nesting degree and the compactness of the resulting structures.

As we already pointed out in Section 3, the only case in which there is no way of providing a direct XML encoding of all the constraints, whatever translation one adopts, is that of many-to-many relationships with total participation of both entities. Some papers recognize the existence of such a problem, but provide no solution; most papers ignore it. Complications inherent to the management of many-to-many relationships in XML are discussed in a survey paper by Link and Trinh on the treatment of cardinality constraints in XML [41]. They analyze a variety of alternative mappings of many-to-many relationships, pointing out their advantages and disadvantages. In particular, they explicitly argue that it is not possible to provide an information-preserving and redundancy-free mapping of many-to-many relationships with total participation of both entities (to cope with this case, some form of existence constraint would be necessary).

Translations of binary (functional or non-functional) relationships can be generalized to relationships of higher degree, as shown in Section 3. As a matter of fact, most proposals in the literature do not explicitly consider higher-degree relationships.

Weak entity. A special case of functional (binary) relationships is that of identifying relationships of weak entities. Let A be a weak entity, R be the identifying relationship, and B be the owner entity. As A has participation constraint equal to $(1, 1)$, we nest the element for A in the element for R, which, in its turn, is nested in the element for B. In addition, the key of A consists of the union of the partial key of A and the key of B. As shown in Section 3, this can be done by duplicating the key of B in the element for A and forcing the uniqueness of the value of the two occurrences of B by means of an external constraint. Most translations proposed in the literature do not deal with weak entities. This is the case, for instance, with [40]. The few exceptions assimilate identifying relationships to functional relationships, neglecting the problem of key definition. This is the case, for instance, with [39]. In [37], Kleiner and Lipeck correctly pointed out the problem with key definition. However, their choice of DTD as the target XML language prevents them from defining the key of the element for the weak entity in a compositional way (as usual). Finally, the straightforward ‘solution’ proposed in [38] does not work, as we already argued in Section 3.

Specialization. Two different approaches to the XML mapping of specializations can be found in the literature. The first one makes use of the construct *extension* (in fact, such a construct is featured by XML Schema, not by DTD) [38]. Given a specialization of an entity A in two entities B and C, the types of the elements for B and C are defined as extensions of the type of the element for A. Such a solution suffers from various weaknesses. First, it does not allow one to express constraints on specializations consisting of one parent entity and two or more children. Moreover, it cannot manage multiple specializations of the same entity (for instance, there is no way to identify the specialization a given child belongs to). The second approach embeds the elements for the children in the element for the parent and it deals with constraints on specializations (total/partial, disjoinct/overlapping) by using the XML constructs *sequence* and *choice* in combination with occurrence constraints [23,37,39]. Despite the reservations formulated in [42], such an approach makes it possible to capture all constraints on specializations, including the total/overlapping constraints, as shown in [23] for binary specializations (in Section 3, we showed that such a solution can be easily generalized to n-ary specializations). An alternative solution is outlined in [40], where the authors suggest to first restructure the ER schema, by replacing specializations by standard relationships, and then to apply the standard translation rules for relationships. In addition, they consider the case in which the parent entity is removed (resp., the children are removed) and information about it (resp., them) is moved to the children (resp., to the parent entity). However, the removal of the parent (resp., children) may introduce redundancy and it does not preserve design reversibility.

6.1.2 Structure of the resulting XML schemas

The XML nesting problem has not been systematically dealt with in the literature. There exist, indeed, various contributions that underline its importance, e.g., [37,39,38]. However, besides recognizing the influence of functional relationships and specializations on the nesting degree of the resulting XML structure, most papers limit themselves to the definition of a translation algorithm, that guarantees neither maximal density nor maximal depth. In particular, they do not explicitly address the problems of nesting confluences and loops.

The translation algorithm outlined in [39] preliminarily identifies the set of first-level entities, namely, those entities whose corresponding elements cannot be nested into other elements without introducing some form of redundancy (for instance, entities whose participation in relationships is always partial), and put them as direct subelements of the root. Then, for every such element, the algorithm generates the XML subtree rooted at it. To this end, it navigates in the ER schema, starting from the corresponding first-level entity, until there are no more reachable entities or relationships whose corresponding elements can be added to the considered subtree. Finally, the algorithm executes the same steps for those (strong) entities, that do not participate in a specialization relation as children, which have not been considered yet (if any). The way in which nesting confluences and loops are dealt with depends on the ordering according to which entities and relationships are taken into consideration (such an ordering is to a large extent arbitrary, e.g., the authors mention as a possible ordering the alphabetical ordering of entity and relationship names). In some critical situations, e.g., loops involving non-first-level entities only, this may prevent the algorithm from achieving the highest possible nesting degree.

A similar translation algorithm is given in [38]. As a preliminary step, the algorithm generates an element for every entity, by distinguishing between strong and weak entities. Then, it processes relationships according to a fixed order: first, it considers relationships with degree greater than 2; then, it copes with recursive relationships; finally, it deals with (non-recursive) binary relationships. As for binary relationships, it starts with many-to-many relationships; then, it moves to one-to-many ones; finally, it considers one-to-one relationships. The final position of the elements for the entities within the resulting XML structure is determined by the relationships they participate in. The authors claim that the order according to which relationships are processed guarantees that the nesting of any pair of elements corresponding to related entities can be fixed once and for all. Unfortunately, it seems that the algorithm makes no provision for the treatment of nesting confluences and loops (as a matter of fact, the informal description of the algorithm makes it difficult to completely check its correctness and the proposed examples are useless, as they avoid all critical cases).

A translation algorithm that takes into account data and query workload of the expected XML applications has been proposed in [40]. Besides the ER schema, the input to such an algorithm includes information about data volumes and types and frequencies of estimated operations. The algorithm consists of a sequence of three main steps: (i) *generalization conversion*; (ii) *relationship conversion*; (iii) *integration*. Both in step (i) and in step (ii), the order according to which specializations (resp., relationships) are considered as well as the choice among the possible alternative (rewritings and) translations are based on information about general access frequencies of the involved relationships and entities. Step (iii) defines the root element of the schema, that can be either the only element devoid of a parent in the current structure or an additional element. The problems of nesting confluences and loops are not explicitly addressed; however, they are indirectly solved by the execution of steps (i) and (ii). Once more, there is no guarantee that the achieved solutions maximize density and/or depth. The effectiveness of the proposed solution has been checked by executing a suitable sets of queries in XQuery on the trial version of the native XML database Tamino and comparing the outcomes with those obtained by applying the same queries to alternative translations given in the literature.

This paper refines and widely extends the work described in [4]. The additional material includes: (i) an introduction to the addressed problem in its generality, emphasizing the distinctive features of native XML databases (Section 2); (ii) a description of the rules of the translation, where all details are worked out (Section 3); (iii) a comprehensive complexity analysis, that takes into account some additional cases of special interest (apart from Theorems 1 and 4, all results in Section 4 are original); (iv) a refined implementation and an extensive experimental evaluation of the proposed mappings (Section 5); (v) a systematic analysis of related work (Section 6).

7 Conclusion

In this paper, we devised an original graph-theoretic approach to the problem of mapping ER schemas into highly-nested XML schema documents. The paper pairs the formal analysis of the computational complexity of the proposed graph algorithms with an empirical evaluation of their implementation. We are thinking of a refinement of the algorithms for the maximum density/depth problems based on weighted graphs, where higher (resp., lower) values are assigned to edges that should be maintained (resp., can be removed). In addition, we are working at an extension of the proposed translation to spatio-temporal conceptual models.

References

1. Gubiani, D., Montanari, A.: ChronoGeoGraph: an expressive spatio-temporal conceptual model. In: SEBD. (2007) 160–171
2. Gubiani, D., Montanari, A.: A tool for the visual synthesis and the logical translation of spatio-temporal conceptual schemas. In: SEBD. (2007) 495–498
3. Schmidt, A., Waas, F., Kersten, M.L., Carey, M.J., Manolescu, I., Busse, R.: XMark: A benchmark for XML data management. In: VLDB. (2002) 974–985 Available at `http://www.xml-benchmark.org`.
4. Franceschet, M., Gubiani, D., Montanari, A., Piazza, C.: From entity relationship to XML Schema: a graph-theoretic approach. In: XSym. Volume 5679 of LNCS. (2009) 145–159
5. Harold, E.R., Means, W.S.: XML in a Nutshell. 3rd edn. O'Reilly (2004)
6. DBIS Research Group: BaseX – Processing and visualizing XML with a native XML database. Available at `http://www.inf.uni-konstanz.de/dbis/basex/`
7. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 6th edn. Addison-Wesley (2010)
8. Liu, C., Vincent, M.W., Liu, J.: Constraint preserving transformation from relational schema to XML Schema. World Wide Web **9**(1) (2006) 93–110
9. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Longman (1995)
10. Krishnan, R., Raghavachari, B.: The Directed Minimum-Degree Spanning Tree Problem. In Hariharan, R., Mukund, M., Vinay, V., eds.: FSTTCS. Volume 2245 of LNCS., Springer (2001) 232–243
11. Yao, G., Zhu, D., Li, H., Ma, S.: A polynomial algorithm to compute the minimum degree spanning trees of directed acyclic graphs with applications to the broadcast problem. Discrete Mathematics **308**(17) (2008) 3951–3959
12. Karger, D.R., Motwani, R., Ramkumar, G.D.S.: On Approximating the Longest Path in a Graph. Algorithmica **18**(1) (1997) 82–98
13. Bazgan, C., Santha, M., Tuza, Z.: On the Approximation of Finding A(nother) Hamiltonian Cycle in Cubic Hamiltonian Graphs. Journal of Algorithms **31**(1) (1999) 249–268
14. Galbiati, G., Morzenti, A., Maffioli, F.: On the approximability of some Maximum Spanning Tree Problems. Theoretical Computer Science **181** (1997) 107–118
15. Dovier, A., Piazza, C., Policriti, A.: An Efficient Algorithm for Computing Bisimulation Equivalence. Theoretical Computer Science **311**(1–3) (2004) 221–256
16. Gubiani, D., Montanari, A.: ChronoGeoGraph. Available at `http://dbms.dimi.uniud.it/cgg/`
17. Gubiani, D., Montanari, A.: A conceptual spatial model supporting topologically-consistent multiple representations. In: GIS. (2008) 57–66
18. Gubiani, D., Montanari, A.: A relational encoding of a conceptual model with multiple temporal dimensions. In: DEXA. (2009) 792–806
19. Afanasiev, L., Franceschet, M., Marx, M., Zimuel, E.: XCheck: A platform for benchmarking XQuery engines (demonstration). In: VLDB. (2006) 1247–1250 Available at `http://www.dimi.uniud.it/∼francesc/xcheck-java`.
20. Kay, M.: Saxon. The XSLT and XQuery Processor. Available at `http://saxon.sourceforge.net`
21. Boncz, P., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: MonetDB/XQuery. MonetDB database system with XQuery front-end. Available at `http://monetdb.cwi.nl/XQuery/index.html`
22. Kappel, G., Kapsammer, E., Retschitzegger, W.: Integrating XML and relational database systems. World Wide Web **7**(4) (2004) 343–384
23. Conrad, R., Scheffner, D., Freytag, J.C.: XML conceptual modeling using UML. In: ER. (2000) 558–571
24. Bird, L., Goodchild, A., Halpin, T.A.: Object role modelling and XML-Schema. In: ER. (2000) 309–322
25. Lee, D., Mani, M., Chiu, F., Chu, W.W.: NeT & CoT: translating relational schemas to XML schemas using semantic constraints. In: CIKM. (2002) 282–291
26. Lv, T., Yan, P.: Mapping relational schemas to XML DTDs with constraints. In: IMSCCS. (2006) 528–533
27. Fong, J., Fong, A., Wong, H.K., Yu, P.: Translating relational schema with constraints into XML schema. International Journal of Software Engineering and Knowledge Engineering **16**(2) (2006) 201–244
28. Zhou, R., Liu, C., Li, J.: Holistic constraint-preserving transformation from relational schema into XML Schema. In: DASFAA. (2008) 4–18
29. Duta, A.C., Barker, K., Alhajj, R.: Converting relationships to XML nested structures. Journal of Information and Organizational Sciences **28**(1-2) (2004) 15–29
30. Fong, J., Cheung, S.K.: Translating relational schema into XML schema definition with data semantic preservation and XSD graph. Information & Software Technology **47**(7) (2005) 437–462
31. Psaila, G.: ERX: A conceptual model for XML documents. In: SAC. (2000) 898–903
32. Necasky, M.: XSEM - A Conceptual Model for XML. In: APCCM. (2007) 37–48
33. Necasky, M.: Conceptual modeling for XML: A survey. In: DATESO. (2006) 40–53
34. Combi, C., Oliboni, B.: Conceptual modeling of XML data. In: SAC. (2006) 467–473
35. Dobbie, G., Xiaoying, W., Ling, T., Lee, M.: Designing semistructured databases using ORA-SS model. In: WISE, IEEE Computer Society (2001) 171–180
36. Fong, J., Cheung, S.K., Shiu, H.: The XML tree model - toward an XML conceptula schema reversed from XML Schema Definition. Data & Knowledge Engineering **64** (2008) 624–661
37. Kleiner, C., Lipeck, U.W.: Automatic generation of XML DTDs from conceptual database schemas. In: GI Jahrestagung (1). (2001) 396–405
38. Liu, C., Li, J.: Designing quality XML schemas from E-R diagrams. In: WAIM. (2006) 508–519
39. Pigozzo, P., Quintarelli, E.: An algorithm for generating XML schemas from ER schemas. In: SEBD. (2005) 192–199
40. Schroeder, R., dos Santos Mello, R.: Designing XML documents from conceptual schemas and workload information. Multimedia Tools and Applications **43**(3) (2009) 303–326
41. Link, S., Trinh, T.: Know your limits: Enhanced XML modeling with cardinality constraints. In: ER. (2007) 19–30
42. Schroeder, R., dos Santos Mello, R.: Conversion of generalization hierarchies and union types from extended entity-relationship model to an XML logical model. In: SAC. (2008) 1036–1037