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

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

MASSIMO FRANCESCHET, University of Udine
DONATELLA GUBIANI, University of Udine
ANGELO MONTANARI, University of Udine
CARLA PIAZZA, University of Udine

We propose a mapping from a database conceptual design to a schema for XML that produces highly connected and nested XML structures. We first introduce two alternative definitions of the mapping, one modeling entities as global XML elements and expressing relationships among them in terms of keys and key references (flat design), the other one encoding relationships by properly including the elements for some entities into the elements for other entities (nest design). Then, we provide a benchmark evaluation of the two solutions showing that the nest approach, compared to the flat one, leads to improvements in both query and validation performances. This motivates us to systematically investigate the best way to nest XML structures. We identify two different nesting solutions: 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, and a maximum *density* nesting, that minimizes the number of schema constraints used in the mapping of the conceptual schema, thus reducing the validation overhead. On the one hand, the problem of finding a maximum depth nesting turns out to be NP-complete and, moreover, it admits no constant ratio approximation algorithm. On the other hand, we devise a graph-theoretic algorithm, NiduX, that solves the maximum density problem in linear time. Interestingly, NiduX finds the optimal solution for the harder maximum depth problem whenever the conceptual design graph is either acyclic or complete. In randomly generated intermediate cases of the graph topology, we experimentally show that NiduX finds a good approximation of the optimal solution.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design

General Terms: Conceptual Modeling, Logical Design, XML Schema

Additional Key Words and Phrases: Entity-Relationship model, graph theory, computational complexity

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. Semistructured data has a loose structure (schema): a core of attributes are shared by all objects, but many individual variants are possible. Hierarchical information is structured as complex entities which, recursively, might embed other complex entities. There is no limit to the nesting level of information. 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. Specifically, we propose a mapping from conceptual designs to logical schemas for native XML databases. We adopt the well-known *Entity-Relationship* (ER) model extended with specialization (specialization is particularly relevant in the design of semistruc-

tured data), as the conceptual model for native XML databases, and we opt for *W3C XML Schema Language* (XML Schema for short) as the schema language for XML. The main alternative to XML Schema is Document Type Definition (DTD) language, but the latter is strictly less expressive than the former. In particular, it lacks expressive means to specify integrity constraints, which are fundamental in database design.

In general, the same ER conceptual schema can be mapped in different XML schemas. We take into consideration two alternative ways of mapping ER conceptual schemas into XML: a flat relational-style design methodology and a nesting approach. The former never nests (the element for) an entity into (the element for) another entity (in analogy to relational databases that feature a distinct table for each entity); the latter nests entities as much as possible. Making use of the well-known XML benchmark XMark running on both native and XML-enabled databases, we provide an experimental evaluation and comparison of these two mapping mechanisms over large datasets. The results of such an experimentation show that both validation and query processing are globally more efficient with nested schemas than with flat ones. This outcome motivates the theoretical problem of finding the best possible nesting for the design. In the search for a solution to this problem, we must take into account at least two different criteria: highly connected XML schemas minimize the number of schema integrity constraints, thus reducing the validation overhead, while highly nested XML schemas reduce the number of expensive join operations that are necessary to reconstruct information at query time, thus decreasing the query evaluation time.

The main contribution of the paper is an original graph-theoretic approach to the problems of finding highly connected and highly nested XML schemas corresponding to entities and relationships of the ER schema. First, we define a conceptual design graph whose nodes are the conceptual entities of the ER schema and whose edges represent functional relationships among these entities. Then, given a conceptual design graph, we study the problems of finding the maximum density spanning forest, that is, the structure with the highest level of connectedness, which corresponds to the nesting that optimizes the validation process, and the maximum depth spanning forest, that is, the structure with the highest level of nesting, which represents the nesting that optimizes the query evaluation task. It turns out that the maximum depth problem is computationally hard, while the maximum density problem is tractable. We thus propose NiduX, an efficient algorithm that solves the maximum density problem in its generality as well as the maximum depth problem whenever the conceptual design graph is at two extremes of the graph topology: either acyclic or complete. Moreover, we show that, on randomly generated graphs lying at intermediate cases with respect to these extremes, NiduX performs surprisingly well, finding approximated solutions for the maximum depth problem that are worth, on average, four-fifth of the optimal solution.

The paper is organized as follows. Section 2 illustrates different ways of mapping conceptual designs into XML schemas, which are evaluated in Section 3 using a benchmark approach. In Section 4 we theoretically investigate the problems of finding the XML schemas with the highest level of connectedness and with the highest level of nesting, and we describe the embedding algorithm NiduX. Section 5 evaluates the effectiveness of NiduX with respect to the hard problem of finding the structure with the highest nesting level. Related work is extensively discussed in Section 6, while conclusions are drawn in Section 7.

2. 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 shall deeply exploit the hierarchical and semistructured nature of the XML data model.

2.1. XML schema notation

Representing an XML schema in XML Schema 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 between DTD and XML Schema[1]. Moreover, for the sake of simplicity, all pieces of information are encoded using XML elements and the use of XML attributes is avoided.

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)`.

XSN extends DTD with the following three constructs (it can be easily shown that each of them can be encoded in XML Schema):

— *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. Consider, for instance, the case of a national soccer team participating in an international competition like the European soccer championship. Each team has a trainer, a physician, one or more physical therapists, and at least 11 and at most 23 players. This last constraint can be expressed as `player[11,23]`. We shall use the following shortcuts borrowed from DTD:
  1. when both x and y are equal to 1, the occurrence constraint may be omitted, that is, the definition `item` equals `item[1,1]`;
  2. when x = 0 and y = 1, the occurrence constraint may be abbreviated as ?, that is, the definition `item?` equals `item[0,1]`;
  3. when x = 0 and y = N, the occurrence constraint may be abbreviated as *, that is, the definition `item*` equals `item[0,N]`;
  4. 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 of A, then the notation `KEY(A.KA)` means that KA is a key for A. Keys consisting of more than one element 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.

[1]We would like to make it clear that we are not interested in the notation XSN per se. In particular, we are not interested in providing a precise characterization of its expressive power. We view XSN as a convenient notation to describe in a succinct way the XML Schema constructs that are exploited in the translation of ER schemas.

— *Foreign key constraints*. If A is an element with key KA, B is an element, and FKA is a child element 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.

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 not expressive enough for our needs. 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.

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 and title are the keys for elements author and paper, respectively, 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)
```

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.

2.2. Mapping ER to XSN

An ER schema basically consists of entities and relationships between them [Elmasri and Navathe 2010]. 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, and 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 problem of mapping ER conceptual schemas into XML ones is definitely not a new one (a detailed analysis of related work is given in Section 6). Nevertheless, a fully satisfactory solution has not been achieved yet. In this paper, we address and work out the main difficulties of the translation problem. In the present section, we introduce a set of rules for the translation of ER constructs into XSN. The mapping rules for some constructs of the ER model, namely, entities and their attributes, are straightforward and well-established; those for other constructs, namely, relationships and specializations, are more complex and controversial. We shall provide alternative definitions

of the mapping rules for ER relationships and specializations, that share their basic features, but differ from each other in some significant respects. As an example, all mapping rules for binary relationships include (the element for) the relationship in (the element for) one of the participating entities, but they differ for the inclusion or not of (the element for) the other entity into (the element for) the relationship. In the former case, the mapping rules produce a "nested" schema; in latter one, they generate a "flat" schema. In the following, we shall systematically analyze and compare these two approaches.

2.2.1. 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.

2.2.2. 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.

2.2.3. Relationships. Each binary relationship has two cardinality constraints of the form (x, y), where x is a natural number, that specifies the minimum cardinality (or 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 cardinality (or cardinality ratio) constraint. In the following, we shall restrict our attention to the cases in which x is either 0 or 1, and y is either 1 or N ($2^4 = 16$ possible combinations), as cardinality constraints of the form (x, y), with $x > 1$, can be basically dealt with as the cardinality constraint $(1, N)$ (later, we shall briefly explain what changes are needed).

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 cardinality constraint (x_1, y_1) and right cardinality constraint (x_2, y_2). We denote such a case with the notation $A \overset{(x_1,y_1)}{\longleftrightarrow} R \overset{(x_2,y_2)}{\longleftrightarrow} B$. We define the mapping of binary relationships case-by-case according to the following order: first one-to-one relationships (Table I); then, one-to-many ones (Table II); finally, many-to-many ones (Table III). We distinguish between flat and nest encodings, and, for each of them, between the mapping that includes R into A and the one that includes R into B.

Table I. In case 1, the choice of including R into A and that of including R into B are equivalent in terms of number of used constraints. Moreover, since no one of the two entities is constrained to participate in the relationship (both of them have minimum cardinality constraint equal to 0), including B into A (resp., A into B) would lead to the loss of all B (resp., A) elements which are not associated with any A (resp., B) element. To prevent it, one must not include one entity into the other, and thus flat and nest mappings coincide. Notice that the constraint `KEY(R.KB)` in the left mapping captures the right maximum cardinality constraint 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 mapping. Attributes of the relationship R (if any) are included in the element R that represents the relationship. In case 2, maximum nesting is achieved by embedding element B into element A. In such a case (nest mapping on the left), no foreign key constraint is necessary and the right minimum cardinality constraint holds by construction. The right max-

Table I. Mapping of one-to-one relationships.

n.	*relationships*		*R into A*	*R into B*

1	$A \overset{(0,1)}{\longleftrightarrow} R \overset{(0,1)}{\longleftrightarrow} B$	flat	`A(KA,R?)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB),KEY(R.KB)`
			`KEYREF(R.KB-->B.KB)`
				`B(KB,R?)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`

		nest	The same as in the flat case.	The same as in the flat case.

2	$A \overset{(0,1)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$	flat	`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)`
				`B(KB,R)`
				`  R(KA)`
				`A(KA)`
				`KEY(A.KA),KEY(B.KB),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`

		nest	`A(KA,R?)`
			`  R(B)`
			`  B(KB)`
			`KEY(A.KA),KEY(B.KB)`
				The same as in the flat case.

3	$A \overset{(1,1)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$	flat	`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)`
				`B(KB,R)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`
				`KEYREF(A.KA-->R.KA)`

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

4	$A \overset{(1,1)}{\longleftrightarrow} R \overset{(0,1)}{\longleftrightarrow} B$	inverse of 2		

imum cardinality constraint is captured by `KEY(B.KB)`. In case we include R into B, the embedding of A into B is not possible, as the left minimum cardinality constraint is 0, and thus flat and nest mappings coincide. An additional key constraint, to capture the left maximum cardinality constraint, and an extra foreign key constraint are needed. In case 3, the two nest (resp., flat) solutions are fully symmetric. In the nest case, the element for one entity can be fully embedded in the element for the other one. In the flat case, besides the standard key constraints for entities A and B, three additional constraints are needed. Consider the flat mapping on the left. The foreign key constraint `KEYREF(R.KB --> B.KB)` is used to restrict participation in R to B elements only, as usual. Moreover, the right maximum cardinality constraint is coded with `KEY(R.KB)`. Finally, the foreign key constraint `KEYREF(B.KB --> R.KB)` captures the right minimum cardinality constraint, which forces each B element to appear inside an R element of A, that is, each B element must be associated with at least one A element. Notice that such a foreign key constraint can be forced since R.KB is a key (in XML Schema, a foreign key cannot refer to something that is not a key). Case 4 is the inverse of case 2, that is, the mappings for case 4 are exactly the same as those for case 2, provided that we substitute B for A, and vice versa.

Table II. As in case 1 (both minimum cardinality constraints are equal to 0), in case 5 there is no way to include one entity into the other, and thus flat and nest mappings coincide. The mappings on the right use the extra constraint `KEY(R.KA)` to capture the left maximum cardinality constraint. Case 6 is the inverse of case 5. Once more, there is no way to include one entity into the other in case 7. On the one hand, A cannot be included into B as this option would lead to the loss of all A elements which are not associated with any B; on the other hand, B cannot be included into A as this option would introduce data redundancy and would violate the key constraint `KEY(B.KB)` (in

Table II. Mapping of one-to-many relationships.

num.	*relationships*		*R into A*	*R into B*

5	$A \overset{(0,1)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$	flat	`A(KA,R?)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
				`B(KB,R*)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`

		nest	The same as in the flat case.	The same as in the flat case.
6	$A \overset{(0,N)}{\longleftrightarrow} R \overset{(0,1)}{\longleftrightarrow} B$	inverse of 5		

7	$A \overset{(0,1)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$	flat	`A(KA,R?)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
			`CHECK(B.KB --> R.KB)`
				`B(KB,R+)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`

		nest	The same as in the flat case.	The same as in the flat case.
8	$A \overset{(1,N)}{\longleftrightarrow} R \overset{(0,1)}{\longleftrightarrow} B$	inverse of 7		

9	$A \overset{(1,1)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$	flat	`A(KA,R)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
				`B(KB,R*)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`
				`KEYREF(A.KA-->R.KA)`

		nest	The same as in the flat case.	`B(KB,R*)`
				`  R(A)`
				`  A(KA)`
				`KEY(B.KB),KEY(A.KA)`

10	$A \overset{(0,N)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$	inverse of 9		

11	$A \overset{(1,1)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$	flat	`A(KA,R)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
			`CHECK(B.KB --> R.KB)`
				`B(KB,R+)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA),KEY(R.KA)`
				`KEYREF(R.KA-->A.KA)`
				`KEYREF(A.KA-->R.KA)`

		nest	The same as in the flat case.	`B(KB,R+)`
				`  R(A)`
				`  A(KA)`
				`KEY(B.KB),KEY(A.KA)`

12	$A \overset{(1,N)}{\longleftrightarrow} R \overset{(1,1)}{\longleftrightarrow} B$	inverse of 11		

general, no embedding of B into A is possible whenever the right maximum cardinality constraint is greater than 1). Flat and nest mappings thus coincide also in this case. The mappings on the right capture the left maximum cardinality constraint by means of the constraint `KEY(R.KA)` (as it happens in the right mappings of case 5). The left mappings do not capture the *right minimum cardinality constraint* `B.KB --> R.KB`, that requires each B instance to be associated with at least one A instance. Such a constraint must be dealt with by an ad hoc validation procedure. The translation rule keeps track of these constraints by properly annotating the generated XML schema and it delegates their verification to an external validation library. The annotation consists of a clause of the form `CHECK(constraint)`, which specifies the constraint to be externally checked. In the considered case, it takes the form `CHECK(B.KB --> R.KB)`. Notice that, to force such a missing constraint, one may be tempted to add the foreign key `KEYREF(B.KB --> R.KB)`. Unfortunately, as we already pointed out, 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. The generalization of the mappings on the right to the case of cardinality constraints (x_2, y_2), with $x_2 > 1$ is straightforward: if suffices to replace `R+` by `R[x2,y2]`. As for the mapping on the left, we must distinguish between $y_2 (\geq x_2) \in \mathbb{N}$ and $y_2 = N$. In the first case, we must pair `CHECK(B.KB-->R.KB)` with `CHECK(A.KA-->R.KA)`, while in the other case, the mappings remain unchanged. Case 8 is the inverse of case 7. In case 9, the nest mapping on the right exploits the full nesting of elements. On the contrary, the nest mapping on the left cannot include B into R, and thus it coincides with the flat mapping, which makes use of an extra keyref constraint. Case 10 is the inverse of case 9. Finally, in case 11, the mappings on the right are exactly the same as those for case 9, apart from the replacement of `B(KB,R*)` by `B(KB,R+)`, and the mappings on the left are exactly the same as those for case 7, apart from the replacement of `A(KA,R?)` by `A(KA,R)`. Case 12 is the inverse of case 11.

Table III. Nesting of the element for one entity into the element for the other one is never possible in the mappings of many-to-many relationships, and thus nest and flat mappings coincide in cases 13-16. Two symmetric mappings are defined in case 13. Since both minimum cardinality constraints are equal to 0, no external checks are needed. In case 14, the left mapping needs an ad hoc validation procedure to check the right minimum cardinality constraint. Case 15 is the inverse of case 14. Finally, case 16 is the only case in the mapping of binary relationships where, whatever solution we adopt, we have to resort to external validation procedure to check one of the two minimum cardinality constraints. 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 constraint")
```

Such a solution captures all integrity constraints specified at conceptual level. However, it imposes 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. Inverse relationship constraints can be formalized in hybrid logic as circular backward constraints (see [Franceschet and de Rijke 2006], page 301). Since this is not a solution we actually take into consideration, we do not believe it necessary to explicitly report its formalization in hybrid logic.

A general translation pattern for binary relationships, that minimizes the number of constraints to be imposed on the resulting XML schemas, can be drawn from the above nest mappings. It 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)`, while an ad hoc validation procedure must be used to check the minimum of the cardinality constraint $(1, N)$ (we keep track of these constraints to be externally checked by means of the clauses `CHECK(B.KB --> R.KB)` or `CHECK(A.KA-->R.KA)`).

Table III. Mapping of many-to-many relationships.

num.	*relationships*		*R into A*	*R into B*

13	$A \overset{(0,N)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$	flat	`A(KA,R*)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
				`B(KB,R*)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA)`
				`KEYREF(R.KA-->A.KA)`

		nest	The same as in the flat case.	The same as in the flat case.

14	$A \overset{(0,N)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$	flat	`A(KA,R*)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
			`CHECK("B.KB --> R.KB")`
				`B(KB,R+)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA)`
				`KEYREF(R.KA-->A.KA)`

		nest	The same as in the flat case.	The same as in the flat case.
15	$A \overset{(1,N)}{\longleftrightarrow} R \overset{(0,N)}{\longleftrightarrow} B$		inverse of 14	

16	$A \overset{(1,N)}{\longleftrightarrow} R \overset{(1,N)}{\longleftrightarrow} B$	flat	`A(KA,R+)`
			`  R(KB)`
			`B(KB)`
			`KEY(A.KA),KEY(B.KB)`
			`KEYREF(R.KB-->B.KB)`
			`CHECK(B.KB --> R.KB)`
				`B(KB,R+)`
				`  R(KA)`
				`A(KA)`
				`KEY(B.KB),KEY(A.KA)`
				`KEYREF(R.KA-->A.KA)`
				`CHECK(A.KA-->R.KA)`

		nest	The same as in the flat case.	The same as in the flat case.

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 shall see later on, the same happens with specializations [Elmasri and Navathe 2010]).

The rules to translate binary relationships can be generalized to relationships of higher degree. For instance, let R be a ternary relationship among A, B, and C, where A participates in R with constraint $(1, N)$, B with constraint $(0, 1)$, and C with constraint $(1, 1)$. A mapping that minimizes the number of constraints is as follows:

```
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 $n > 2$, that minimizes the number of constraints to be imposed on the resulting XML schemas, 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 even such an entity does not exist, we choose one that participates with cardinality constraint $(0, N)$. If all entities participate with cardinality constraint $(1, 1)$, we shall 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.

Table IV. Mapping of different types of specialization.

type of specialization	*nest mapping*	*flat mapping*
partial overlapping	A(KA,attA,B?,C?) B(attB) C(attC) KEY(A.KA)	A(KA,attA) B(KA,attB) C(KA,attC) KEY(A.KA),KEY(B.KA),KEY(C.KA) KEYREF(B.KA-->A.KA) KEYREF(C.KA-->A.KA)
partial disjoint	A(KA,attA, (B|C)?) B(attB) C(attC) KEY(A.KA)	A(KA,attA) B(KA,attB) C(KA,attC) KEY(A.KA),KEY(B.KA|C.KA) KEYREF(B.KA-->A.KA) KEYREF(C.KA-->A.KA)
total overlapping	A(KA,attA, ((B,C?) | C)) B(attB) C(attC) KEY(A.KA)	B(KA,attA,attB) C(KA,attA,attC) KEY(B.KA),KEY(C.KA)
total disjoint	A(KA,attA, (B|C)) B(attB) C(attC) KEY(A.KA)	B(KA,attA,attB) C(KA,attA,attC) KEY(B.KA|C.KA)

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.

2.2.4. 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 above cases 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 emphasizing that the use of an external validation procedure to check the constraint `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 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 when the relationship is one-to-many). On the other hand, if we point the selector at the level of 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[2].

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

2.2.5. Specialization. The hierarchical nature of the XML data model can be successfully exploited in the mapping of specializations. We distinguish four cases: partial-overlapping, partial-disjoint, total-overlapping, and total-disjoint specializations. Let us consider first consider simple specializations with a parent entity A, with key KA and attributes attA, and two child entities B and C with attributes attB and attC, respectively. Nest and flat mappings are described in Table IV.

If the specialization is partial, the nest solution, that embeds both element B and element C inside element A, is definitely more compact. Neither key nor foreign key constraints are necessary for B and C. The overlapping constraint is captured by using the occurrence specifiers: A may contain any subset of {B, C}. The disjoint constraint is expressed by means of a choice: A may contain either B or C. In the flat solution, disjointness is forced by the key constraint `KEY(B.KA|C.KA)`: the value for KA must be unique across the union of B and C domains. If the specialization is total, the flat solution, unlike the nest one, discards the entity A. This results in a more compact mapping, but, being attA included in both element B and element C, redundancy arises in case of disjoint specializations.

The generalization to specializations involving $n > 2$ child entities is immediate in all cases except for the nest mapping of total-overlapping specializations. 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:

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

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 relationships.

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 [Elmasri and Navathe 2010]. 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.

[2]Apparently, the authors of [Liu and Li 2006] repeatedly missed this point.

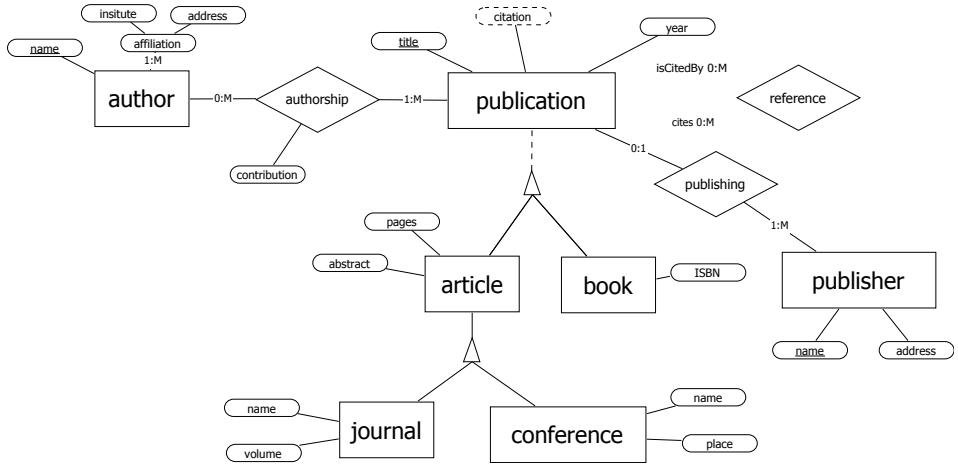

Fig. 1. A citation-enhanced bibliographic database.

```
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.

We conclude the section with a simple example: the mapping of the ER schema given in Figure 1 (we borrow the notation from ChronoGeoGraph, a software framework for the conceptual and logical design of spatio-temporal databases [Gubiani and Montanari 2007]), which describes a citation-enhanced bibliography (a typical semi-structured data instance), is reported in Figure 2.

3. BENCHMARK EVALUATION OF THE MAPPING

In Section 2, we presented different ways to map the same conceptual information into an XML structure. They can be brought back to the following two alternative mechanisms to encode conceptual relationships and specializations:

— *the reference mechanism*, in which the entities involved in relationships and specializations are translated as global XML elements, that is, as immediate children of the database XML element, and the relationships between them are encoded using the

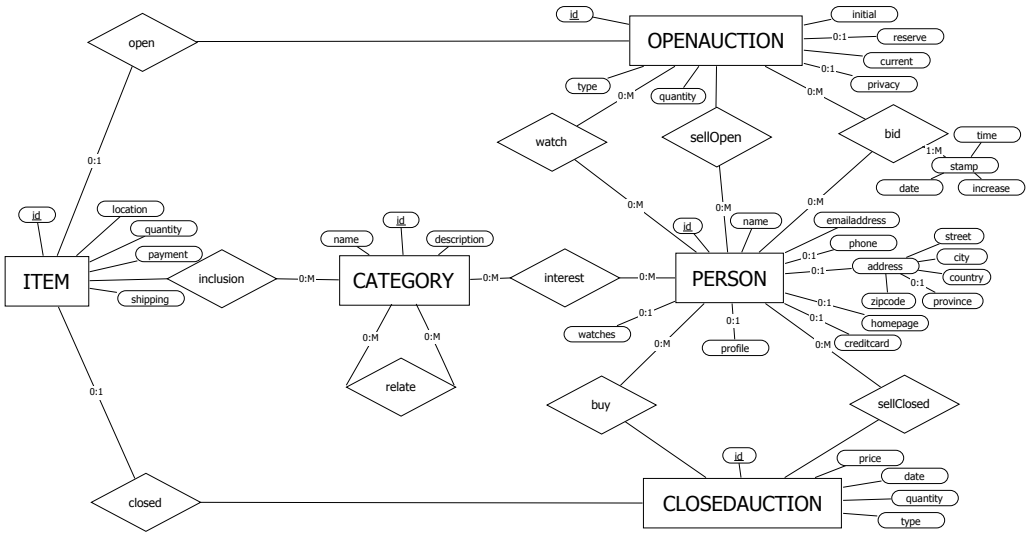

Fig. 3. The XMark conceptual schema.

mechanism of keys and key references (this mechanism is well known to the relational database world);

— *the embedding mechanism*, in which relationships and specializations are encoded by properly including some entities into other ones.

The embedding mechanism produces a hierarchical XML structure, the reference mechanism generates flat structures. Moreover, the latter introduces more integrity constraints (keys and key references) than the former.

The goal of this experimental section is to understand the practical impact of these two translation mechanisms with respect to validation and query processing. The principal hypothesis we are going to test is that both validation and query processing are more efficient on XML structures generated with the embedding mechanism.

The experimental setup is as follows. We ran the experiments on a 2.53 GHz machine with 2.9 GB of main memory running Ubuntu 9.10 operating system. We make use of the well-known XML benchmark XMark [Schmidt et al. 2002]. XMark models an Internet auction web site in which people watch ongoing open auctions, bidding 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 3 (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 4), which exploits the embedding of XML elements as much as possible, and a flat, relational-like XMark schema (Figure 5), in which each entity of the conceptual design is encoded as a global XML element and conceptual relationships are mapped using the key and key reference mechanisms (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 to the XMark schema. The user can control the size of the generated document using a scaling parameter,

```
// 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. 4. A nested XMark schema in XSN.

where scale 1 corresponds to a document of 100 MB. We used the data generator to produce XML instances of increasing size and mapped these XML instances into corresponding instances for the nested and flat designs. We generated XMark documents from 5 MB to 1 GB corresponding to the following scaling parameters: 0.05, 0.1, 0.5, 1, 5, and 10.

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. The resulting elapsed times, expressed in seconds, are shown in Figure 6. Validating nested designs is more efficient than validating flat schemas. The reason is that, 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)[3].

As for query performance, we used the following XQuery benchmark, which is partially borrowed from the XMark benchmark:

— **Q1**: Categories and the items they contain.

[3]Incidentally, we noticed that expressing constraints using the child axis (/) instead of the descendant axis (//) in the selector XPath expression of the key and keyref constraints of the schema makes validation faster on hierarchical schemas, while the effect is negligible on flat designs. This might be a useful guidance for database designers.

```
// 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. 5. A flat XMark schema in XSN.

— **Q2**: Categories and the open auctions bidding items belonging to these categories.
— **Q3**: The open and corresponding closed auctions.
— **Q4**: People and the items they bought.
— **Q5**: The number of pieces of prose present in the database.
— **Q6**: People and the closed auctions bidding items bought by them.
— **Q7**: People and the number of items they bought.
— **Q8**: The number of sold items that cost more than 40.

Notice that queries Q4, Q5, Q7, and Q8 above correspond to queries Q9, Q7, Q8, and Q5 in XMark, respectively. 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. Queries Q1, Q2, and Q3 have been designed to fully exploit the hierarchical structure of the XML documents as much as possible. The nested version of these queries uses 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 version of

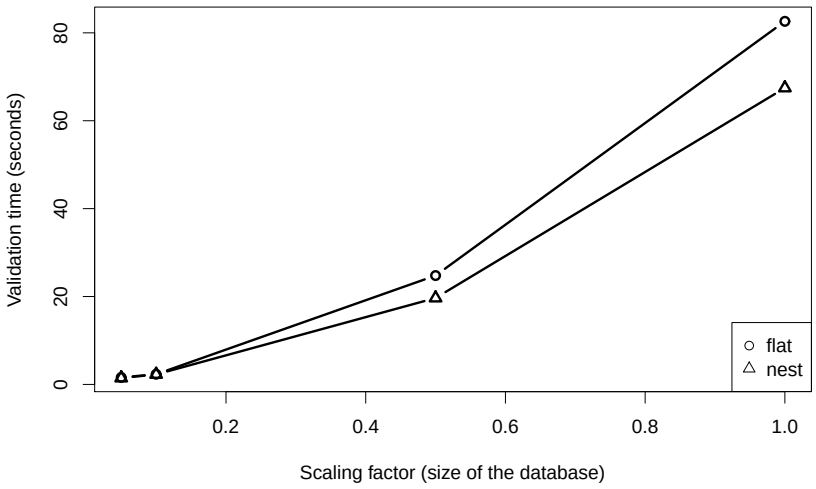

Fig. 6. Validation times of nested and flat XML instances against nested and flat XML schemas.

the queries uses 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. We reasonably expect that the nested version of these queries is evaluated more efficiently than the corresponding flat version.

On the contrary, queries from Q5 to Q8 have been designed to favor flat instances over nested ones. The flat version of these queries accesses elements which are immediate children of the root of the XML document, hence superficial elements that are immediately reachable. On the other hand, the nested version of these queries locates elements that are arbitrarily nested below the root of the XML document. We believe that the flat version of these queries is evaluated more efficiently than the corresponding nested version.

Finally, query Q4 is hybrid. It contains two joins; the nested version uses both the above described unfolding and product techniques to solve them, accessing elements that are significantly nested below the tree root. The flat version uses the product approach for both joins, but it accesses superficial elements that are immediately reachable from the tree root. We refer to queries from Q1 to Q4 as pro-nest queries, while queries from Q5 to Q8 are pro-flat queries. By way of example, we show both the flat and nested versions of queries Q2, Q4, and Q6 of our benchmark.

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:

```
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>
```

Q4. *People and the items they bought*. The flat version of this query performs two joins: a first join between people and closed auctions, and a second one between closed auctions and items:

```
let $auction := doc("xmark.xml")
for $p in $auction/site/Person
let $auc := $auction/site/ClosedAuction[buy/personref = $p/id]
let $item := $auction/site/Item[id = $auc/closed/itemref]
return <person id="{$p/id}">
         <items>{$item}</items>
       </person>
```

The nested version partially exploits the hierarchical structure of the nested instances in which closed auctions are embedded inside items. However, people and closed auctions are on different subtrees of the nested schema, hence a traditional join is necessary to related these two entities:

```
let $auction := doc("xmark.xml")
for $p in $auction/site/Person
let $item := $auction/site/Category/inclusion/Item
             [closed/ClosedAuction/buy/personref = $p/id]
return <person id="{$p/id}">
         <items>{$item}</items>
       </person>
```

Q6. *People and the closed auctions bidding items they bought*. The flat instance of the query makes a join between people and closed auctions, which are both unnested, immediately accessible elements:

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

The nested version makes a similar join between people and closed auctions. The unfolding technique cannot be exploited in this case, and a relational-style join is necessary. Moreover, the elements involved in the query must be located in the hierarchy before the join operation can start.

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

We tested the devised benchmark on three open-source XML query engines: BaseX (version 6.1) [DBIS Research Group 2011], Saxon (home edition 9.4 for Java) [Kay 2011], and MonetDB/XQuery (release 4) [Boncz et al. 2011]. BaseX is a native XML

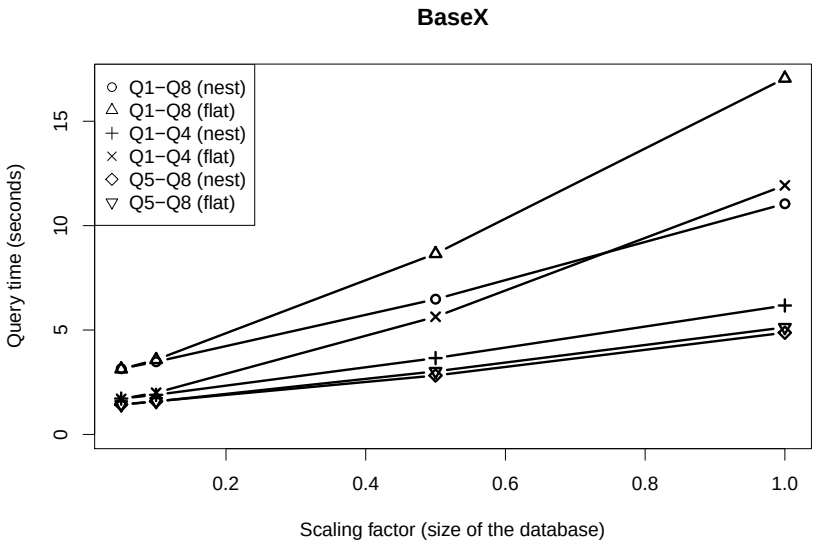

Fig. 7. Query benchmark performance for BaseX. The six lines refer to the nested and flat version of all benchmark queries (Q1-Q8), of pro-nest queries (Q1-Q4), and of pro-flat queries (Q5-Q8).

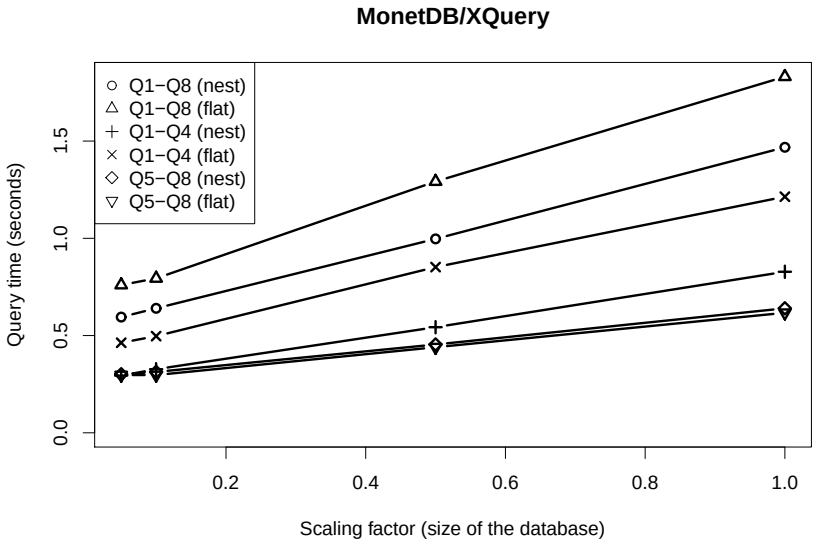

Fig. 8. Query benchmark performance for MonetDB/XQuery. The six lines refer to the nested and flat version of all benchmark queries (Q1-Q8), of pro-nest queries (Q1-Q4), and of pro-flat queries (Q5-Q8).

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 on different designs, and

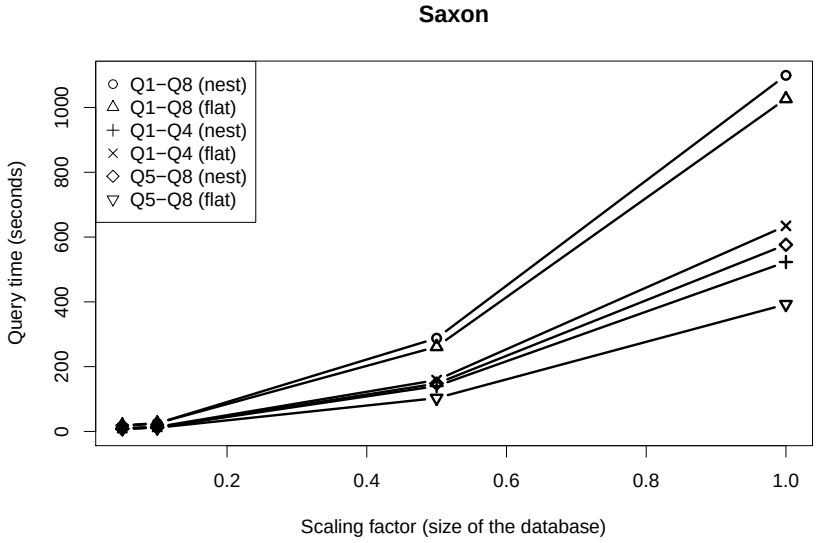

Fig. 9. Query benchmark performance for Saxon. The six lines refer to the nested and flat version of all benchmark queries (Q1-Q8), of pro-nest queries (Q1-Q4), and of pro-flat queries (Q5-Q8).

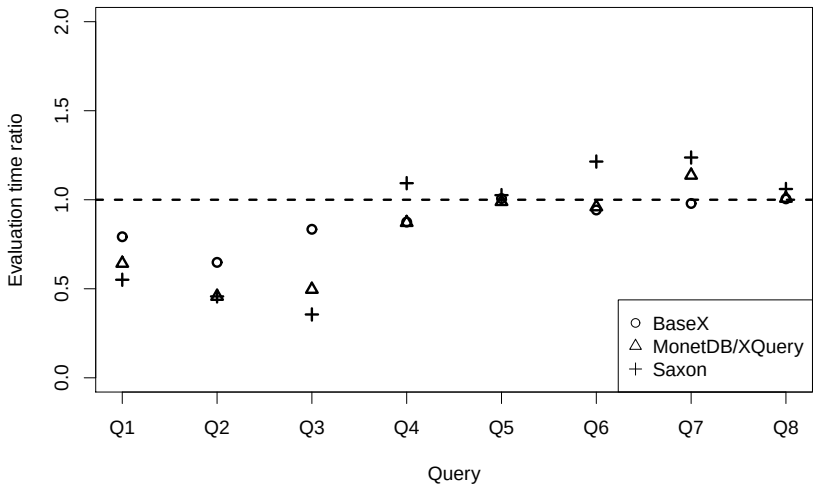

Fig. 10. Query performance ratio between the nested and flat version of the benchmark queries for the three engines.

not query performance on different engines. We are benchmarking schema designs, not query processors. We built no indexes to speed up query processing and we ran all queries with a warm cache.

Figures 7, 8, and 9 show the query benchmark performance (the total time for the evaluation of the benchmark), expressed in seconds of elapsed time, for the three benchmarked engines. BaseX evaluates the nested version of the pro-nest benchmark more efficiently than the corresponding flat version, while there is no significant difference between the two versions of queries in the pro-flat benchmark. Overall, nested queries on nested documents are processed by BaseX faster than flat queries on flat documents. The picture is very similar when MonetDB/XQuery is chosen as query engine. Saxon also evaluates the nested version of pro-nest queries quicker than the corresponding flat version. However, as for pro-flat queries, the flat version is clearly faster than the nested one. Overall, there is a tie between the two versions of queries and documents when Saxon is used as query engine.

We noticed that the query evaluation times computed with Saxon have a high variability (they significantly deviate from the mean evaluation time), while the times for the other query engines are more stable. It follows that, for Saxon, benchmark performance is dominated by few time demanding queries. Hence, we also computed the query evaluation time ratio between the nested and flat version of the benchmark queries. This ratio does not depend on the absolute query evaluation times. Figure 10 shows the query performance ratio between the nested and flat version of the benchmark queries for the three engines. For each query, the ratio is computed as the evaluation time of the nested version of the query divided by the evaluation time of the flat version of the query, averaged over all document sizes. Hence, a ratio below 1 means that the nested version is evaluated more efficiently than the flat one, a ratio above 1 indicates that the flat version is evaluated faster than the nested one, while a ratio close to 1 corresponds to similar evaluation times. Our interpretation of the experimental results is summarized in the following:

(1) in case of pro-nest queries (Q1-Q4), the nested version is evaluated more efficiently than the corresponding flat version. The mean query performance ratio between the nested and flat version of pro-nest queries is 0.63 for BaseX, 0.62 for MonetDB/XQuery, and 0.50 for Saxon. In particular, the genuine pro-nest queries Q1, Q2, and Q3 are well below the unity threshold for all engines. As for the hybrid query Q4, it is slightly below unity for BaseX and MonetDB/XQuery and a little above unity for Saxon;
(2) the nested and flat versions of queries are evaluated approximately in the same time in case of pro-flat queries (Q5-Q8), with the exceptions of Q6 and Q7 for Saxon, for which the flat version is faster than the nested one. The mean query performance ratio between the nested and flat version of pro-flat queries is 1.00 for BaseX, 1.02 for MonetDB/XQuery, and 1.15 for Saxon;
(3) considering the whole benchmark (Q1-Q8), which has been designed to balance between pro-nest and pro-flat queries, the nested version of queries and schemas has still a computational advantage over the flat version. Indeed, the mean query performance ratio between the nested and flat version of all queries is 0.82 for BaseX, 0.82 for MonetDB/XQuery, and 0.83 for Saxon.

In summary, the experimental results tell us that, on the benchmarked queries and engines, the nested version of pro-nest queries is more efficient than the corresponding flat version, while the nested and flat versions of pro-flat queries have similar evaluation times. The message is that hierarchical designs make query evaluation more efficient when the queries exploit the hierarchy of the design, while the penalty is not severe, compared to a corresponding flat design, when the queries cannot make use of this hierarchy. Since validation is also more efficient on hierarchical designs, we conclude that nesting the design in XML schemas (and formulating hierarchy-aware queries) is computationally appealing. This outcome motivates the theoretical problem

of finding the best possible nesting for the design, an issue investigated in the following section.

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-degree 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 according to the embedding approach. 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 embedding translation rule can be applied to one of the relationships only, while for the other relationship we must resort to the alternative (reference) translation rule. As we shall see, in order to select the relationship to which the embedding 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.

According to the rules for the translation of binary relationships given in Section 2, 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 *XML nesting problem*. In the following, we provide a graph-theoretic formalization of such a problem and we propose and contrast possible solutions to it. We introduce two problems over graphs: maximum depth and maximum density. The first one consists of the search for the structure with the highest level of nesting (maximum depth spanning forest), the second one consists of the search for the structure with the highest level of connectedness (maximum density spanning forest). Unfortunately, while the second problem (maximum density) can always be solved in linear time, the first one turns out to be NP-complete and not even approximable in the general case. However, we show that the two problems are strongly related: the linear time algorithm for maximum density can be used (i) to solve maximum depth on acyclic graphs and on complete graphs, and (ii) to approximate it on graphs with strongly connected components of bounded size. In the last part of the section, we generalize such results allowing the translation to force some relationships to be maintained.

4.1. Maximum Density and Maximum Depth Problems

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

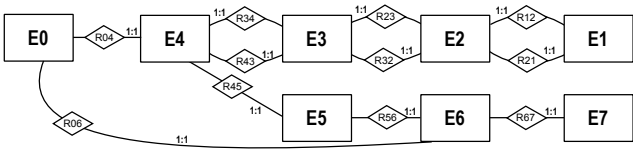

Fig. 11. An ER schema. We assume that each entity E_i has a key attribute k_i and the missing cardinality constraints are different from $(1, 1)$.

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. For instance, the ER schema given in Figure 11 generates the graph depicted in Figure 12. 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 sub-digraph 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 (every root has thus depth 0). The *depth* of a forest is the sum of the depths of its nodes. The nesting problem can be formalized in terms of the following two problems.

Definition 4.1. Given a digraph G the *maximum depth problem* over G consists in finding a spanning forest F of G whose depth is maximum with respect to all the spanning forests of G. Given a digraph G the *maximum density problem* over G consists in finding a spanning forest F of G whose number of edges is maximum with respect to all the spanning forests of G.

Both problems always admit a solution, which is not necessarily unique. The reader might wonder if a spanning forest with maximum depth is also a spanning forest with maximum density. Unfortunately, the answer in negative, as shown by the example depicted in Figure 12.

As we already pointed out in the introduction, maximum depth and maximum density spanning forests are the nestings that optimize the query evaluation task and the validation process, respectively. As shown in the mapping of conceptual schemas into XML schemas (see, for instance, case 2 in Table I), each total functional relationship in the conceptual schema, when translated using the embedding strategy, saves some integrity constrains (keys and foreign keys). Indeed, the topological inclusion of one element into another in the XML document captures, by itself, some cardinality constraints of the conceptual relationship that otherwise need to be specified with KEY and KEYREF mechanisms. Since a total functional relationship corresponds to an edge of the spanning forest, the larger is the number of edges in the spanning forest, the lower is the number of constraints in the resulting schema. Hence, the maximum density spanning forest corresponds to the nesting structure that minimizes the number of key and foreign key constraints of the schema, and hence, ultimately, the validation time. On the other hand, the maximum depth spanning forest minimizes the number of relational-style join operations that are necessary to reconstruct information at query time. Indeed, the deeper a node in the spanning forest is, the larger the number of ancestor nodes of that node is. Combining information between two nodes belonging to the same path (descendant-ancestor nodes) can be done by simply following the path

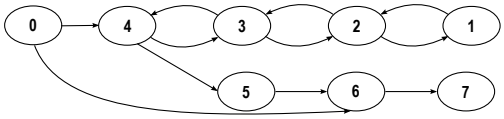

Fig. 12. A maximum density spanning forest for the given digraph is obtained by removing edges (1,2), (2,3), (3,4), and (0,6). It consists of one tree, with 7 edges, and the sum of node depths is 19. A maximum depth spanning forest is the simple path from node 1 to node 7 plus node 0. It consists of 2 trees, with 6 edges in total, and the sum of node depths is 21. Both solutions are unique.

between the two nodes (an operation that is highly optimized in XML databases), and, in particular, it does not require a costly relational-style join operation.

4.2. The Complexity of Maximum Depth

Let us first consider the maximum depth problem. It is not difficult to see that it is close to the Hamiltonian path problem. Given a (directed) 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., [Papadimitriou 1995]). 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 4.2. *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, ..., 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 (2) (by contraposition).
(iii) implies (i). 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., [Krishnan and Raghavachari 2001; Yao et al. 2008]). 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 13.

In [Karger et al. 1997], it has been shown that, given an undirected graph, the problem of finding a longest path is not constant ratio approximable[4] in polynomial time,

[4]We recall that a maximization problem is constant ratio approximable if there exists an algorithm such that, independently of the input, the ratio σ/σ^* between the cost σ of the approximated solution found by the algorithm and the cost σ^* of the optimal solution of the problem is bounded by a positive constant.

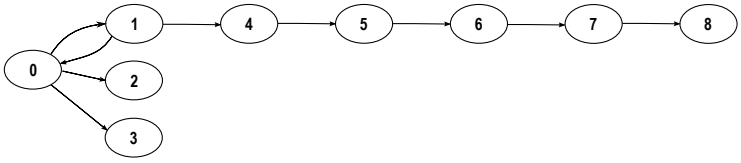

Fig. 13. A maximum depth spanning forest for the given digraph can be obtained by removing edge (1,0). It consists of one tree, whose sum of node depths is 23. Its maximum out-degree is 3. A minimum out-degree spanning forest can be obtained by removing edge (0,1). It consists of one tree whose sum of node depths is 20 and maximum out-degree is 2.

unless $P = NP$. In [Bazgan et al. 1999], such a result is extended to the case of cubic Hamiltonian graphs. In [Galbiati et al. 1997], the above results are exploited to prove that, given an undirected graph, the problem of finding a spanning tree with maximum sum of distances from a specified root is not constant ratio 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 [Galbiati et al. 1997] 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 [Bazgan et al. 1999], 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 ratio approximable in polynomial time, unless $P = NP$. Theorem 4.4 shows that such a result can actually be tailored to digraphs.

As a preliminary step, we prove a property of a subclass of digraphs. The idea is to prove Theorem 4.4 by using the standard mapping from connected undirected graphs to directed ones, which replaces an undirected edge with two directed ones. The digraphs we obtain applying such a mapping are such that each of their maximum depth forests consists of one tree. The *strongly connected components* of a digraph are maximal sets of mutually reachable nodes. In the general case, the number of strongly connected components of a digraph ranges from 1 to $|V|$. A digraph is *strongly connected* if it consists of one strongly connected component.

LEMMA 4.3. *Let $G = (V, E)$ be a strongly connected digraph 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 (i.e., it has a unique root).*

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 4.4. *Unless $P = NP$, there is no polynomial-time 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 4.3, 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 maximum depth and maximum density problems. We start with the case of Directed Acyclic Graphs (DAG). As a matter of fact, the digraph depicted in Figure 4.2, 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 DAG, a maximum depth spanning forest is also a maximum density spanning forest.

THEOREM 4.5. *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. □

4.3. A Linear Time Solution for Maximum Density

In the following, we provide a linear time algorithm, called NiduX, that solves the maximum density problem for digraphs as well as the maximum depth problem for DAGs. As a preliminary step, we introduce a suitable notion of rank that allows us to partition the nodes of a DAG into different strata (see, e.g., [Dovier et al. 2004]).

Definition 4.6. Let $G = (V, E)$ be a DAG. For each $v \in V$, we define $rank_G(v)$ as follows:

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

Let $G = (V, E)$ be a digraph.

(1) compute the graph H of the strongly connected components of G (let $C = \{C_1, \ldots, C_n\}$ 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 (using a depth-first visit);
  (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 (using a depth-first visit);
(5) output the forest $F = (V, E' \cup E_1 \cup E_2 \cup \ldots \cup E_n)$.

Fig. 14. The algorithm NiduX.

According to Definition 4.6, 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.

The algorithm NiduX is provided in Figure 14. Given a digraph G, it first computes a graph H, whose nodes are the strongly connected components of G and whose edges (C_j, C_i) are defined as follows: (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. NiduX operates on it by exploiting the notion of rank.

The main steps of the execution of NiduX on the digraph in Figure 12 are graphically depicted in Figure 15. Step 1 generates the graph H of the strongly connected components consisting of 5 nodes, namely, $C_1 = \{0\}$, $C_2 = \{4, 3, 2, 1\}$, $C_3 = \{5\}$, $C_4 = \{6\}$, and $C_5 = \{7\}$ (we associate with each C_1 the set of nodes of the original digraph it includes). Step 2 computes a maximum density spanning forest for it consisting 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). At step 3, these edges are replaced by edges $(0, 4)$, $(4, 5)$, $(5, 6)$, and $(6, 7)$. Next, at step 4, a tree T_2, rooted at node 4, spanning C_2 is computed, which contains the edges $(4, 3)$, $(3, 2)$, and $(2, 1)$. The trees T_1, T_3, T_4, and T_5 consist of a single node. Step 5 returns a maximum density spanning forest consisting of a single tree, where the sum of node depths is 19. The XSN schema corresponding to the computed maximum density spanning forest is shown in Figure 16 (assuming all missing cardinality constraints in Figure 11 to be of the form `[0,N]`).

LEMMA 4.7. *The spanning forest K generated by step 2 of the algorithm NiduX is a maximum depth spanning forest for H.*

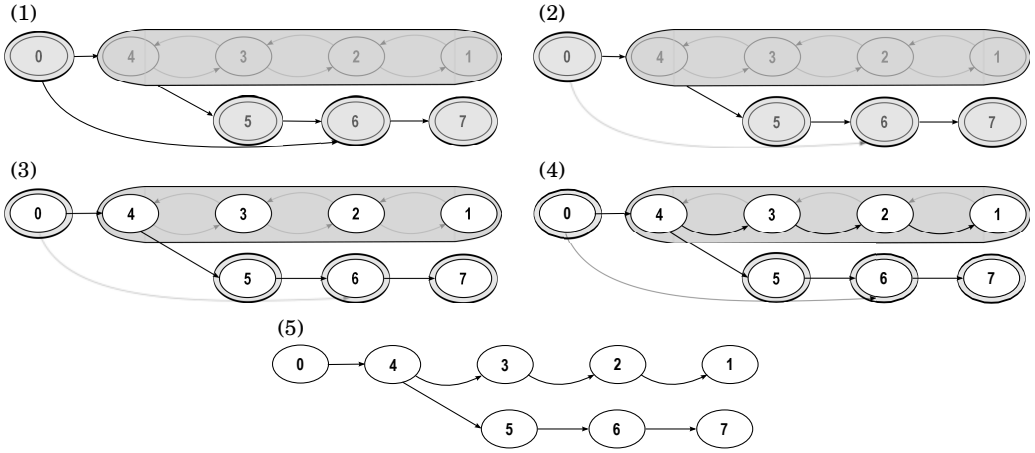

Fig. 15. Execution of NiduX on the graph in Figure 12.

```
SCHEMA(E0*)
  E0(k0,R04*)
    R04(E4)
      E4(k4,R34,R43*,R45*)
        R34(E3ref)
        R43(E3)
          E3(k3,R23,R32*)
            R23(E2ref)
            R32(E2)
              E2(k2,R12,R21*)
                R12(E1ref)
                R21(E1)
                  E1(k1)
        R45(E5)
          E5(k5,R56*)
            R56(E6)
              E6(k6,R06,R67*)
                R06(E0ref)
                R67(E7)
                  E7(k7)
  KEY(E0.k0), KEY(E1.k1), KEY(E2.k2), KEY(E3.k3)
  KEY(E4.k4), KEY(E5.k5), KEY(E6.k6), KEY(E7.k7)
  KEYREF(R34.E3ref-->E3.k3), KEYREF(R23.E2ref-->E2.k2)
  KEYREF(R12.E1ref-->E1.k1), KEYREF(R06.E0ref-->E0.k0)
```

Fig. 16. Mapping of the ER schema represented in Figure 11.

PROOF. Let C_i be a node in C. By Definition 4.6, 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, it 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, NiduX 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 C_i is at depth $h + 1$ in K. □

Since H is a DAG, from Theorem 4.5, it follows that K is also a maximum density spanning forest for H.

THEOREM 4.8. *Let G be a digraph. The algorithm NiduX 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. NiduX picks a node v of G and it computes a tree T rooted at v and spanning G. Hence, NiduX 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 $C_1, \ldots, C_{n+1}$. 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 NiduX 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 NiduX 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 NiduX (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 4.7, 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, NiduX 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, NiduX 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, NiduX picks a node v in C_1 and it computes a tree T_1, rooted at v, which spans C_1. By inductive hypothesis, NiduX 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, NiduX is correct on G.

To prove that NiduX 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 by exploiting a visit over H; (iii) all ranks over H^{-1} can be computed in

Fig. 17. A graph on which NiduX performs increasingly worse with respect to the maximum depth problem as the size of the graph grows.

linear time by a DFS-visit over H^{-1} (see [Dovier et al. 2004]); (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. □

4.4. Applying NiduX to Maximum Depth

By Theorems 4.2 and 4.4, we know that, in general, there is no guarantee about the goodness of the spanning forest computed by NiduX with respect to the maximum depth problem. This is mainly due to (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.

Since the maximum depth problem is not constant ratio approximable in polynomial time, and since NiduX is a polynomial algorithm, there must exist a graph G_n with n nodes such that, as n goes to infinity, the value of the solution computed by NiduX, that is, the depth of the maximum density spanning forest, diverges from the value of the optimal solution for the problem, that is, the depth of the maximum depth spanning forest. An example of such a graph, with $n = 6$, is depicted in Figure 17. The maximum density spanning forest computed by NiduX is the tree $\{(0,1),(1,2),(1,3),(1,4),(1,5)\}$, whose depth, in case of n nodes, is $\sigma_n = 1 + 2(n-2) = O(n)$. On the other hand, the maximum depth spanning forest is the path $\{(5,4),(4,3),(3,2),(2,1)\}$, whose depth, in general, is $\sigma^*_n = (n-2)(n-1)/2 = O(n^2)$. Clearly, as n grows, the approximation ratio σ_n/σ^*_n vanishes.

Nevertheless, in the following we show that NiduX finds the optimal solution if the input graph is acyclic or if it is complete. Moreover, we show that, in general, the goodness of the solution computed by NiduX with respect to the maximum depth problem can be estimated in terms of the size of the largest strongly connected component of the graph.

THEOREM 4.9. *Let G be a DAG. The algorithm NiduX 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 4.7, it immediately follows that NiduX computes a maximum depth spanning forest for G. □

On complete graphs, NiduX performs equally well: it always finds the optimal solution. This because it uses a depth-first search to span the SCCs of the graph (step 4 of the algorithm), that clearly finds an Hamiltonian path of the graph, hence the optimal solution, if the graph is complete.

Furthermore, for the general case, we have the following result about the effectiveness of NiduX as an approximation algorithm for the maximum depth problem.

THEOREM 4.10. *Let $G = (V, E)$ be a digraph, F be the spanning forest computed by NiduX on G, S_F be the depth of F, and S be the maximum depth of a spanning forest*

for G. If each strongly connected component of G contains at most k nodes, then

$$\frac{S}{2k} \leq S_F \leq S$$

PROOF. Let $H = (C, E_H)$ be the graph of the strongly connected components of G, let $\{0, 1, \ldots, h\}$ be the set of ranks in H^{-1}, and let r_0 be the number of strongly connected components of rank 0 in H^{-1}. Moreover, for $i \in \{0, 1, \ldots, h\}$, let n_i be the number of nodes belonging to a strongly connected component of rank i. Finally, let $m_1 = n_1 + n_0 - r_0$ and let $m_i = n_i$, for $i \in \{2, \ldots, h\}$.

We first provide a lower bound to S_F. Let us consider all the nodes belonging to the strongly connected components of rank 0. There are n_0 nodes ranging over r_0 strongly connected components. NiduX puts r_0 nodes at depth 0, while the remaining $n_0 - r_0$ nodes are at least at depth 1. Moreover, NiduX puts the n_1 nodes belonging to the strongly connected components of rank 1 at least at depth 1. Hence, m_1 nodes are at least at depth 1. Similarly, NiduX puts the m_2 $(= n_2)$ nodes belonging to the strongly connected components of rank 2 at depth at least 2. In the general case, NiduX puts the m_i $(= n_i)$ nodes belonging to the strongly connected components of rank i at depth at least i. Hence, we have that

$$S_F \geq \sum_{i=1}^{h} m_i \cdot i$$

Now, we over-approximate S by maximizing the depth of each node. In G, there are r_0 strongly connected components of rank 0. These components have no incoming edges from other strongly connected components, and thus in each spanning forest for G, there is at least one root from each of them, that is, in each spanning forest for G, there are at least r_0 nodes at depth 0. Since each strongly connected component of G has at most k nodes, in each spanning forest for G, each of the remaining $n_0 - r_0$ nodes belonging to a strongly connected component of rank 0 is at most at depth $k - 1$. As far as the nodes belonging to strongly connected components of rank 1 are concerned, in each spanning forest for G, each of them is at most at depth $2 \cdot k - 1$. Each of them, indeed, can be at most at the end of a chain crossing one strongly connected component of rank 0 and one strongly connected component of rank 1, that is, each of them can be at most at the end of a chain involving $2 \cdot k$ nodes. By over-approximating, we can say that, in each spanning forest for G, there are m_1 nodes at depth at most $2 \cdot k$. Similarly, we can say that, in each spanning forest for G, the m_2 $(= n_2)$ nodes belonging to the strongly connected components of rank 2 are at depth at most $3 \cdot k$. In the general case, we can say that, in each spanning forest for G, the m_i $(= n_i)$ nodes belonging to the strongly connected components of rank i are at most at depth $(i+1) \cdot k$. Hence, we have that

$$S \leq \sum_{i=1}^{h} m_i \cdot (i+1) \cdot k \leq 2 \cdot k \cdot \sum_{i=1}^{h} m_i \cdot i$$

Since $\sum_{i=1}^{h} m_i \cdot i \leq S_F$, we can conclude that $S \leq 2 \cdot k \cdot S_F$. □

It follows that the approximation ratio σ/σ^* between the approximated solution computed by NiduX and the optimal solution of the maximum depth problem is always bigger than or equal to $1/2k$, with k the size of the largest strongly connected component of the graph. Hence, the maximum depth problem is constant ratio approximable on the class of graphs for which the size of the largest strongly connected component is bounded by a constant.

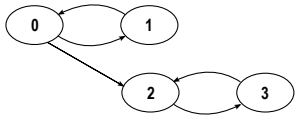

Fig. 18. Different solutions to the maximum density problem for the given digraph exist, each one consisting of 1 tree with 3 edges. None of them contains the edge (3,2). A maximum density spanning forest containing the edge (3,2) necessarily consists of 2 trees with 1 edge each.

4.5. The Constrained Case

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 nesting translation rules 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.

Definition 4.11. Let $G = (V, E)$ be a digraph and $C \subseteq E$ be a set of edges. The *constrained maximum depth problem* over G and C is the problem of finding a spanning forest F of G containing all edges in C and whose depth is maximum with respect to all the spanning forests of G which contain all the edges in C. The *constrained maximum density problem* over G and C is the problem of finding a spanning forest F of G containing all edges in C and whose number of edges is maximum with respect to all the spanning forests of G which contain all the edges in C.

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 18.

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

COROLLARY 4.13. *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 polynomial-time constant ratio approximation algorithm for it.*

PROOF. NP-hardness immediately follows from Theorem 4.2 (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 4.2), it fol-

Let $G = (V, E)$ be a digraph and let $C \subseteq E$.

(1) check that C contains neither loops nor confluences; otherwise, stop with failure (it has no solution);
(2) compute the set of target nodes $T = \{v \mid \exists (u, v) \in C\}$ 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 NiduX 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 $\overline{F}$.

Fig. 19. The algorithm ConstrainedNiduX.

lows 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 4.4 (take $C = \emptyset$). □

We now show that the constrained maximum density problem can be effectively reduced to the maximum density one. Let ConstrainedNiduX be the algorithm in Figure 19, which takes a digraph $G = (V, E)$ and a set $C \subseteq E$ as input.

THEOREM 4.14. *Let $G = (V, E)$ be a digraph and let $C \subseteq E$. ConstrainedNiduX solves the constrained maximum density problem for G and C in linear time.*

PROOF. First, by Lemma 4.12, 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, NiduX works in linear time and all the other steps have linear time complexity. Hence, ConstrainedNiduX has linear time complexity. □

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

LEMMA 4.15. *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 constrained 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 4.5, $\overline{F}$ is also a maximum density spanning forest for $\overline{G}$ (contradiction). □

THEOREM 4.16. *Let $G = (V, E)$ be a DAG and let $C \subseteq E$. The algorithm ConstrainedNiduX 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 4.9, NiduX computes a maximum depth spanning forest for $\overline{G}$. By Lemma 4.15, 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 ConstrainedNiduX is also a maximum depth spanning forest for G. □

We would like to conclude by emphasizing that complexity results and algorithms we provided in this section are original. As for the maximum depth problem, we already pointed out that an undirected rooted version of it has been studied in [Galbiati et al. 1997], where the authors show that it is not constant ratio approximable in polynomial time, unless $P = NP$. However, as witnessed by many important problems in graph theory, small differences in the definition of the problem may lead to important differences in terms of complexities. As an example, *undirected reachability* has been recently proved to be solvable in logarithmic space (L), while *directed reachability* is in non-deterministic logarithmic space (NL), and by proving that it is in L one could conclude that $L = NL$. As far as the maximum density problem is concerned, we defined NiduX exploiting a set theoretical notion of rank. In graph theory, the term *rank* usually refers to the rank of the adjacency matrix. The set theoretical notion of rank

Table V. Evaluation of the approximation of NiduX relative to the maximum depth problem. The meaning of columns is the following: n is the number of nodes of the graph, m is the number of edges, scc is the number of SCCs, k is the size of the largest SCC, r is the inter-component density, σ^* is the exact solution, and σ is the approximated solution computed by the algorithm. The last two columns represent the approximation ratio $\sigma/\sigma^* \in [1/2k, 1]$ and the theoretical lower bound of the approximation ratio $1/2k$.

n	m	scc	k	r	σ^*	σ	σ/σ^*	$1/2k$
7	8	5	2	0.15	6	4	0.67	0.25
7	9	5	3	0.15	10	10	1.00	0.17
8	10	5	2	0.10	7	5	0.71	0.25
10	17	5	3	0.10	17	13	0.76	0.17
12	25	5	4	0.15	24	15	0.62	0.12
14	21	10	2	0.10	15	15	1.00	0.25
15	25	10	2	0.15	38	30	0.79	0.25
17	40	5	4	0.10	70	50	0.71	0.12
19	33	10	3	0.15	28	24	0.86	0.17
19	32	15	2	0.15	26	21	0.81	0.25
21	38	10	3	0.10	63	52	0.83	0.17
21	37	15	2	0.10	34	27	0.79	0.25
27	47	20	2	0.10	36	36	1.00	0.25
30	55	20	2	0.15	69	68	0.99	0.25
35	63	20	3	0.10	71	55	0.77	0.17
200	415	1	200	0	19900	15487	0.78	$2.5 \cdot 10^{-3}$
400	860	1	400	0	79800	69243	0.87	$1.2 \cdot 10^{-3}$
600	983	1	600	0	179700	143379	0.80	$8.3 \cdot 10^{-4}$
800	2826	1	800	0	319600	283568	0.89	$6.2 \cdot 10^{-4}$
1000	3192	1	1000	0	499500	367330	0.74	$5.0 \cdot 10^{-4}$

we used has been exploited in [Dovier et al. 2004] to define efficient bisimulation algorithms. However, the problems we considered in this paper have no correlation with bisimulation. Finally, both the relationships we established between maximum depth and maximum density solutions with respect to the size of the largest strongly connected components and the constrained analysis are not only new, but clearly tailored to our applications.

5. EXPERIMENTAL EVALUATION OF THE NESTING ALGORITHM

In Section 4 we provided a linear time algorithm, called NiduX, that solves the maximum density problem for arbitrary digraphs. Unfortunately, in the general case, the maximum depth problem is computationally hard and hence there exists little hope to discover a polynomial algorithm that solves it. Nevertheless, we have seen that NiduX solves the maximum depth problem for acyclic directed graphs and for complete graphs, and, in general, it computes an approximation σ of the exact solution σ^* for the maximum depth problem such that $\sigma^*/2k \leq \sigma \leq \sigma^*$, where k is the size of the largest strongly connected component (SCC, for short) of the graph. This means that the approximation ratio $\rho = \sigma/\sigma^*$ lies in the interval $[1/2k, 1]$.

In this section, we experimentally investigate the approximation ratio ρ of NiduX relative to the maximum depth problem. We conjecture that ρ is well above the theoretical lower bound of $1/2k$ and possibly close to the upper bound 1. The experimental setup is as follows. We implemented, using the computing environment R, a parameterized generator of *scale-free graphs*. Scale-free graphs have been largely investigated in the network science community since they share many structural features with real-world networks; in particular, scale-free graphs are small-world networks with a node degree distribution following a power law in which most of the nodes (the trivial many) have low degree and a small but significant share of nodes (the vital few) have an extraordinary high degree [Barabási and Albert 1999; Barabási et al. 1999]. This

distribution is the consequence of the popular cumulative advantage (or preferential attachment) phenomenon, effectively summarized by the sentence *the rich get richer*.

Specifically, the graph generator that we realized works as follows. It has three parameters: the number of SCCs of the graph, a vector with the number of nodes for each SCC, and an inter-component edge probability, that is the percentage of edges to draw among two different connected components as a fraction of the maximum number of edges that are possible among the two components. Notice that these parameters are important in the present experimental study, since NiduX works directly on the SCCs of the graph. The generator works in three phases:

(1) a scale-free directed acyclic graph (DAG) is generated with as many nodes as the number of SCCs of the graph to generate;
(2) each node of the DAG is expanded into a strongly connected scale-free graph of prescribed size;
(3) a number of random edges among nodes of different strongly connected components is added according to the given inter-component edge probability.

We made use of the scale-free graph generator offered by the computing environment R to generate scale-free graphs in phases (1) and (2) of our generator. This basic generator works as follows: (i) the initial graph has a single isolated node; (ii) additional nodes are added to the graph one at a time; (iii) each node connects to the existing nodes with a fixed number r of links. The probability that it will choose a given node is proportional to the number of links the chosen node already has. In particular, in phase (1) we set the fixed number of links $r = 1$, obtaining a DAG. In phase (2) we set $r = 2$, and then add random edges until the graph becomes strongly connected.

Furthermore, we coded NiduX in the C programming language. As for the exact solution of the maximum depth problem, we defined a mapping from the problem to integer linear programming. We have implemented the mapping through a Perl script which takes in input the adjacency matrix of a graph and outputs an integer linear programming problem. A solution of the integer linear programming problem denotes a maximum depth spanning forest of the input graph. We have used the solver *lp_solve* to find such a solution. It is worth noticing that finding the exact solution for the maximum depth problem, the spanning forest of maximum depth, is computationally demanding even on relatively small graphs. A potential reason is that the number of possible spanning forests of a graph might be enormous even if the size of the graph is modest.[5]

The first part of Table V (up to $n = 35$) shows the results of experiments on a sample of scale-free graphs generated varying the number of nodes, the number of SCCs, the maximum number of nodes per SCC, and the inter-component density of the graphs. Interestingly, the approximation ratio $\rho = \sigma/\sigma^*$ between the approximated and exact solution is always well above the theoretical lower bound $1/2k$ and often close to the theoretical upper bound 1. The mean approximation ratio is 0.82, which means that, on average, the sub-optimal depth computed by NiduX is 82% of the maximum depth. This is four times larger than the mean theoretical lower bound of the approximation ratio, which amounts to 0.21.

The graphs used so far are relatively small networks characterized by many little SCCs linked together. We also tested the effectiveness of the proposed approximation on larger, strongly connected graphs (graphs with only one SCC) containing a Hamilto-

[5]The Kirchhoff's matrix-tree theorem is an elegant result stating that the total number of spanning trees of an undirected graph is exactly the determinant of any principal minor of the Laplacian matrix of the graph [Kirchhoff 1847]. The Cayley's formula provides a closed form of this number for a complete graph of n nodes: n^{n-2} [Cayley 1889].

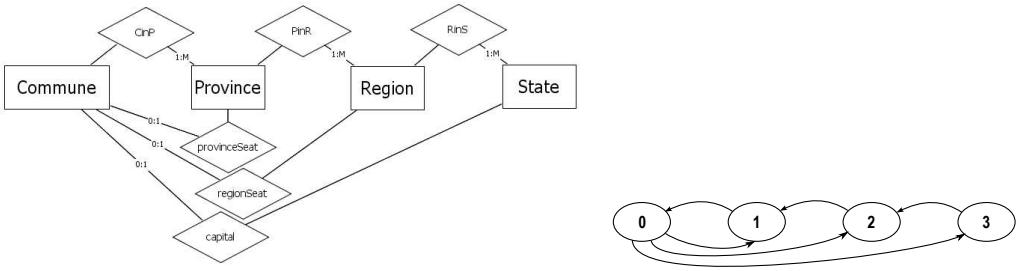

Fig. 20. The Italian administrative organization example.

nian path. These graphs have been generated starting from an Hamiltonian path and randomly adding edges until the graph becomes strongly connected. For these graphs, the maximum depth forest is the Hamiltonian path, whose depth in a graph with n nodes is $n(n-1)/2$. Hence, we are not bound to find the costly exact solution, while the approximated solution terminates very quickly. Results for these graphs are shown in the second part of Table V (from $n = 200$). Once again, the effectiveness of the approximated solution is encouraging, with a mean approximation ratio of 0.82. This is, interestingly, the same approximation ratio we obtained for smaller graphs with many little SCCs. It is worth noticing that none of the strongly connected graphs we have used is complete. On complete graphs, we recall that NiduX performs even better: it always finds the optimal solution.

Summing up, the algorithm NiduX we have devised finds the optimal solution at the extreme of the network topology, that is, for acyclic graphs (graphs with many SCCs composed of a single node) and for complete graphs (graphs with a single SCC with the maximum number of edges). Moreover, on randomly generated networks lying between these extremes, it finds an approximated solution that is worth about four-fifths of the exact solution.

We complemented the above experimentation with testing of NiduX on ER schemas taken from real-world application domains, namely, a conceptual schema modeling entities and relations at the basis of the Italian administrative organization; a schema extracted from KEGG Database, which is a widely-used database that integrates genomic, chemical, and systemic functional information; a schema taken from PathCase, which is an integrated software tool for the analysis of biological systems; and a schema underlying the content management system Joomla!.

The conceptual schema for the Italian administrative organization is depicted in Figure 20 on the left. A set of neighboring communes forms a province. One of them is chosen as the representative of the province. Similarly, provinces are organized in regions and regions form a state. In each region, one commune is chosen as the representative. Finally, one commune is the capital of the state. The directed graph corresponding to the schema is depicted in Figure 20 on the right. It is worth pointing out that it is quite similar to the graph of Figure 17, which represents NiduX worst case of approximation. However, due to the lack of a *privileged* root (node 0 in Figure 17), NiduX on the graph of Figure 20 computes a forest of depth 6, which coincides with the exact solution of the maximum depth problem. One can force NiduX to produce a bad approximation on this case using a particular permutation of the names of the nodes. We generalized the graph to the case of 20 nodes. In such a case, NiduX computes a forest of depth 190. Again, the resulting forest is the exact solution of the maximum depth problem as well. However, as one can imagine, the exact solution computed through *lp_solve* requires more than 10 minutes, while NiduX takes few milliseconds.

The KEGG schema we considered has 37 nodes organized in 36 strongly connected components, that is, it has only one cycle involving 2 nodes. On such a case, NiduX computes a forest of depth 66, which coincides with the exact solution of the maximum depth problem. The PathCase schema generates an acyclic graph with 17 nodes. NiduX computes a forest of depth 14. Since this graph is acyclic, this is also the maximum depth forest. (As a matter, the same happens with the XMark schema we extensively used in Section 3.) The graph obtained from the Joomla! schema has 33 nodes organized in 30 strongly connected components. More precisely, it has 28 strongly connected components of size 1, one strongly connected component of size 2, and one strongly connected component of size 3. NiduX generates a forest of depth 39, which is also a maximum depth forest.

Pairing the outcomes of the experimentations on synthetic graphs and on graphs extracted from real-world conceptual schemas, there is a clear evidence that solutions obtained by NiduX are quite satisfactory: in many cases, NiduX computes the exact solution, and, whenever this is not the case, the approximated solution it computes is sufficiently close to the exact one.

6. RELATED WORK

There is a vast literature on the relationships between XML and relational databases, which ranges from expressiveness issues to performance comparison. 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 [Kappel et al. 2004]. Unfortunately, this literature is partly redundant and not well linked. We identified three main themes related to our work: the encoding of (standard) conceptual schemas into some XML schema language, the development of new conceptual models tailored to XML databases, and the mapping of relational schemas into XML schemas. Even though our work presents significant intersections with research about relational-to-XML mappings [Duta et al. 2004; Fong and Cheung 2005; Fong et al. 2006; Lee et al. 2002; Liu et al. 2006; Lv and Yan 2006; Shanmugasundaram et al. 2001; Zhou et al. 2008] and XML-inspired conceptual models [Combi and Oliboni 2002; Dobbie et al. 2001; Fong et al. 2008; Necasky 2007; Psaila 2000] (a detailed analysis of both these research directions can be found in [Franceschet et al. 2012]), the closest research theme is definitely that about the XML encoding of (standard) conceptual schemas.

Some work has been done to provide an XML encoding of formalisms for conceptual modeling like, for instance, UML [Conrad et al. 2000] and ORM [Bird et al. 2000]; however, 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 first mappings from conceptual models to XML proposed in the literature take DTD as their target language [Conrad et al. 2000; Kleiner and Lipeck 2001]. 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 [Kappel et al. 2004]. 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 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 [Liu and Li 2006], including (absence of) redundancy, conformity with applications, and design reversibility. In fact, they are not independent from the basic ones, and thus we shall not discuss them separately. In the following, we first focus our attention on the preservation of information and then on the nesting degree of the resulting XML schemas.

6.1. 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 in Section 2 can be viewed as the merge (and the improvement) of a number of existing proposals, coping with all distinctive features of the ER model. 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 much more complex than that of entities as one must take into account all their distinctive features. 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 cardinality 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 [Kleiner and Lipeck 2001; Duta et al. 2004; Pigozzo and Quintarelli 2005]. Minor variants to such a translation schema have been proposed in [Liu and Li 2006], where the relative positions of R and A are exchanged, and in [Schroeder and dos Santos Mello 2009], 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) [Kleiner and Lipeck 2001; Schroeder and dos Santos Mello 2009].

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 cardinality 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 cardinality 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 [Pigozzo and Quintarelli 2005], the B-element corresponds to an entity with cardinality constraint $(1,N)$ (if any). Such a solution aims at minimizing the number of additional (external) constraints. In [Liu and Li 2006], 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 [Schroeder and dos Santos Mello 2009]. The domi-

nance/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 [Kleiner and Lipeck 2001], 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 [Duta et al. 2004]. The mapping of many-to-many relationships with partial participation of entities is the same as in [Kleiner and Lipeck 2001], while the treatment of the other cases is the same as in [Pigozzo and Quintarelli 2005], 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 2, 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 [Link and Trinh 2007]. 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 2. 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. Most translations proposed in the literature do not deal with them. This is the case, for instance, with [Schroeder and dos Santos Mello 2009]. The few exceptions assimilate identifying relationships to functional relationships, neglecting the problem of key definition. This is the case, for instance, with [Pigozzo and Quintarelli 2005]. In [Kleiner and Lipeck 2001], 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 [Liu and Li 2006] does not work, as we already argued in Section 2.

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) [Liu and Li 2006]. 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. The second approach embeds the elements for the children in the element for the parent and it deals with constraints on specializations by using the XML constructs *sequence* and *choice* in combination with occurrence constraints [Conrad et al. 2000; Kleiner and Lipeck 2001; Pigozzo and Quintarelli 2005]. Despite the reservations formulated in [Schroeder and

dos Santos Mello 2008], such an approach makes it possible to capture all constraints on specializations, including the total/overlapping constraints, as shown in [Conrad et al. 2000]. An alternative solution is outlined in [Schroeder and dos Santos Mello 2009], 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, such removals may introduce redundancy and they do not preserve design reversibility.

6.2. Structure of the resulting XML schemas

The XML nesting problem has not been systematically dealt with in the literature. Various contributions, indeed, underline its importance, e.g., [Kleiner and Lipeck 2001; Pigozzo and Quintarelli 2005; Liu and Li 2006]; 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 depth nor maximal density. In particular, they do not explicitly address the problems of nesting confluences and loops.

The translation algorithm outlined in [Pigozzo and Quintarelli 2005] 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 [Liu and Li 2006]. 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 [Schroeder and dos Santos Mello 2009]. Besides the ER schema, the input to such an algorithm includes information about data volumes and types and frequencies of estimated operations. The algorithm con-

sists 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 depth and/or density. 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.

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 proposed translation achieves maximum connectivity and deep nesting in the structure used to embed the elements of the conceptual design in order to optimize both validation and query performances. The paper paid a special attention to the XML nesting problem, that plays a central role in the proposed design methodology. First, it provided a characterization of such a problem in terms of the maximum depth and maximum density problems. Then, it systematically analyzed their computational complexity, showing that the maximum depth problem is NP-complete and it admits no constant ratio approximation algorithm, while the maximum density problem can be solved in linear time (algorithm NiduX). Finally, it proved that NiduX finds a maximum depth nesting in case of acyclic or complete graphs, and it experimentally showed that, in the other cases, its outcome is quite close to such a nesting. A detailed survey of related work concludes the paper.

Among the various possible developments of the work done, we are focusing our attention on two directions. First, in Subsection 4.5, we defined a constrained variant of the maximum depth and maximum density problems, where the designer can force the maintenance of some edges of the original digraph. Additional flexibility can be achieved by associating a weight with each edge, where higher (resp., lower) weights are assigned to edges that should be maintained (resp., can be removed). The constrained case can be recovered as a special case of weighted graphs, where the same higher weight is assigned to the edges to be possibly maintained and the same lower weight is assigned to the remaining edges. Weights can be assigned according to different criteria. As an example, one can give a higher weight to edges representing relationships involved in frequently asked/dominant queries. As an alternative, the weight of an edge can be defined as the number of constraints that should be added in case the edge was removed. Second, to keep the translation algorithm as simple as possible, we replaced higher-degree relationships and specializations of ER schemas by a suitable number of total functional binary relationships (see Section 4). An explicit treatment of higher-degree relationships and specializations would allow us to trade simplicity of the algorithm for compactness of its output (the resulting graph). From a technical point of view, it requires the replacement of edges by hyperedges, thus leading to the replacement of digraphs by hypergraphs.

ACKNOWLEDGMENTS

We would like to thank the reviewers whose comments and suggestions helped us a lot in improving the paper.

REFERENCES

BARABÁSI, A.-L. AND ALBERT, R. 1999. Emergence of scaling in random networks. *Science 286*, 509–512.

BARABÁSI, A.-L., ALBERT, R., AND JEONG, H. 1999. Mean-field theory for scale-free random networks. *Physica A 272,* 1-2, 173–187.

BAZGAN, C., SANTHA, M., AND TUZA, Z. 1999. On the approximation of finding a(nother) hamiltonian cycle in cubic hamiltonian graphs. *Journal of Algorithms 31,* 1, 249–268.

BIRD, B., GOODCHILD, A., AND HALPIN, T. 2000. Object role modelling and XML-Schema. In *Proceedings of the 19th International Conference on Conceptual Modeling*. Springer, 309–322.

BONCZ, P., GRUST, T., VAN KEULEN AMD S. MANEGOLD, M., RITTINGER, J., AND TEUBNER, J. 2011. MonetDB/XQuery. MonetDB database system with XQuery front-end. http://monetdb.cwi.nl/XQuery/index.html.

CAYLEY, A. 1889. A theorem on trees. *Quarterly Journal of Mathematics 23*, 376–378.

COMBI, C. AND OLIBONI, B. 2002. Conceptual modeling of XML data. In *Proceedings of the 17th Symposium on Applied Computing*. ACM, 467–473.

CONRAD, R., SCHEFFNER, D., AND FREYTAG, J. 2000. XML conceptual modeling using UML. In *Proceedings of the 19th International Conference on Conceptual Modeling*. Springer, 558–571.

DBIS RESEARCH GROUP. 2011. BaseX – Processing and visualizing XML with a native XML database. http://www.inf.uni-konstanz.de/dbis/basex/.

DOBBIE, G., XIAOYING, W., LING, T., AND LEE, M. 2001. Designing semistructured databases using ORA-SS model. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering*. IEEE Computer Society, 171–180.

DOVIER, A., PIAZZA, C., AND POLICRITI, A. 2004. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science 311,* 1–3, 221–256.

DUTA, A., BARKER, K., AND ALHAJJ, R. 2004. Converting relationships to XML nested structures. *Journal of Information and Organizational Sciences 28,* 1-2, 15–29.

ELMASRI, R. AND NAVATHE, S. 2010. *Fundamentals of Database Systems. 6th edn.* Addison-Wesley.

FONG, J. AND CHEUNG, S. 2005. Translating relational schema into XML schema definition with data semantic preservation and XSD graph. *Information & Software Technology 47,* 7, 437–462.

FONG, J., CHEUNG, S., AND SHIU, H. 2008. The XML tree model - toward an XML conceptula schema reversed from XML Schema Definition. *Data & Knowledge Engineering 64*, 624–661.

FONG, J., FONG, A., WONG, H., AND YU, P. 2006. Translating relational schema with constraints into XML schema. *International Journal of Software Engineering and Knowledge Engineering 16,* 2, 201–244.

FRANCESCHET, M. AND DE RIJKE, M. 2006. Model checking for hybrid logics (with an application to semistructured data). *Journal of Applied Logic 4(3)*, 279–304.

FRANCESCHET, M., GUBIANI, D., MONTANARI, A., AND PIAZZA, C. 2012. From entity relationship to XML Schema: a graph-theoretic approach. Technical Report UDMI/01/12/RR, University of Udine.

GALBIATI, G., MORZENTI, A., AND MAFFIOLI, F. 1997. On the approximability of some maximum spanning tree problems. *Theoretical Computer Science 181*, 107–118.

GUBIANI, D. AND MONTANARI, A. 2007. A tool for the visual synthesis and the logical translation of spatio-temporal conceptual schemas. In *Proceedings of the 15th Italian Symposium on Advanced Database Systems*. 495–498.

KAPPEL, G., KAPSAMMER, E., AND RETSCHITZEGGER, W. 2004. Integrating XML and relational database systems. *World Wide Web 7,* 4, 343–384.

KARGER, D., MOTWANI, R., AND RAMKUMAR, G. 1997. On approximating the longest path in a graph. *Algorithmica 18,* 1, 82–98.

KAY, M. 2011. Saxon. the XSLT and XQuery Processor. http://saxon.sourceforge.net.

KIRCHHOFF, G. 1847. Über die auflösung der gleichungen, auf welche man bei der untersuchung der linearen verteilung galvanischer ströme geführt wird. *Annalen für der Physik und der Chemie 72*, 497–508.

KLEINER, C. AND LIPECK, U. 2001. Automatic generation of XML DTDs from conceptual database schemas. *GI Jahrestagung 1*, 396–405.

KRISHNAN, R. AND RAGHAVACHARI, B. 2001. The directed minimum-degree spanning tree problem. In *Proceedings of 21st Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 232–243.

LEE, D., MANI, M., CHIU, F., AND CHU, W. W. 2002. NeT & CoT: translating relational schemas to XML schemas using semantic constraints. In *Proceedings of the 11th International Conference on Information and Knowledge Management*. ACM, 282–291.

LINK, S. AND TRINH, T. 2007. Know your limits: Enhanced XML modeling with cardinality constraints. In *Proceedings of the 26th International Conference on Conceptual Modeling*. Springer, 19–30.

LIU, C. AND LI, J. 2006. Designing quality XML schemas from E-R diagrams. In *Proceedings of the 7th International Conference on Advances in Web-Age Information Management*. Springer, 508–519.

LIU, C., VINCENT, M., AND LIU, J. 2006. Constraint preserving transformation from relational schema to XML Schema. *World Wide Web 9,* 1, 93–110.

LV, T. AND YAN, P. 2006. Mapping relational schemas to XML DTDs with constraints. In *Proceedings of the 1st International Multi-Symposiums of Computer and Computational Sciences*. IEEE Computer Society, 528–533.

NECASKY, M. 2007. XSEM - A Conceptual Model for XML. In *Proceedings of the 4th Asia-Pacific Conference on Conceptual Modelling*. Australian Computer Society, 37–48.

PAPADIMITRIOU, C. 1995. *Computational Complexity.* Addison Wesley Longman.

PIGOZZO, P. AND QUINTARELLI, E. 2005. An algorithm for generating XML schemas from ER schemas. In *Proceedings of the 15th Italian Symposium on Advanced Database Systems*. 192–199.

PSAILA, G. 2000. ERX: A conceptual model for XML documents. In *Proceedings of the 7th Annual International Workshop on Selected Areas in Cryptography*. Springer, 898–903.

SCHMIDT, A., WAAS, F., KERSTEN, M., CAREY, M., MANOLESCU, I., AND BUSSE, R. 2002. XMark: A benchmark for XML data management. In *Proceedings of the 28th International Conference on Very Large Data Bases*. ACM, 974–985.

SCHROEDER, R. AND DOS SANTOS MELLO, R. 2008. Conversion of generalization hierarchies and union types from extended Entity-Relationship model to an XML logical model. In *Proceedings of the 23rd Symposium on Applied Computing*. ACM, 1036–1037.

SCHROEDER, R. AND DOS SANTOS MELLO, R. 2009. Designing XML documents from conceptual schemas and workload information. *Multimedia Tools and Applications 43,* 3, 303–326.

SHANMUGASUNDARAM, J., SHEKITA, E. J., BARR, R., CAREY, M. J., LINDSAY, B. G., PIRAHESH, H., AND REINWALD, B. 2001. Efficiently publishing relational data as xml documents. *VLDB Journal 10(2-3)*, 133–154.

YAO, G., ZHU, D., LI, H., AND MA, S. 2008. A polynomial algorithm to compute the minimum degree spanning trees of directed acyclic graphs with applications to the broadcast problem. *Discrete Mathematics 308,* 17, 3951–3959.

ZHOU, R., LIU, C., AND LI, J. 2008. Holistic constraint-preserving transformation from relational schema into XML Schema. In *Proceedings of the 13th International Conference on Database Systems for Advanced Applications*. Springer, 4–18.