Learning Schemas for Unordered XML

Radu Ciucanu, Slawomir Staworko

Learning Schemas for Unordered XML

Radu Ciucanu
University of Lille & INRIA, France
radu.ciucanu@inria.fr

Sławek Staworko
University of Lille & INRIA, France
slawomir.staworko@inria.fr

Abstract

We consider unordered XML, where the relative order among siblings is ignored, and we investigate the problem of learning schemas from examples given by the user. We focus on the schema formalisms proposed in [10]: *disjunctive multiplicity schemas* (DMS) and its restriction, *disjunction-free multiplicity schemas* (MS). A learning algorithm takes as input a set of XML documents which must satisfy the schema (i.e., *positive examples*) and a set of XML documents which must not satisfy the schema (i.e., *negative examples*), and returns a schema consistent with the examples. We investigate a learning framework inspired by Gold [18], where a learning algorithm should be *sound* i.e., always return a schema consistent with the examples given by the user, and *complete* i.e., able to produce every schema with a sufficiently rich set of examples. Additionally, the algorithm should be *efficient* i.e., polynomial in the size of the input. We prove that the DMS are learnable from positive examples only, but they are not learnable when we also allow negative examples. Moreover, we show that the MS are learnable in the presence of positive examples only, and also in the presence of both positive and negative examples. Furthermore, for the learnable cases, the proposed learning algorithms return minimal schemas consistent with the examples.

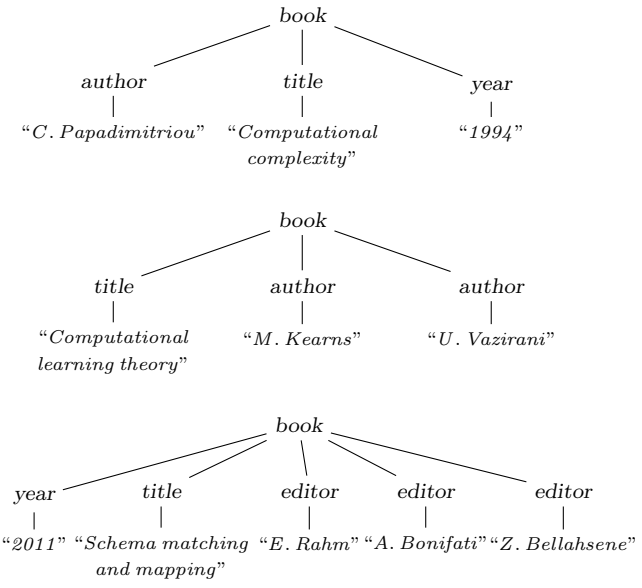

Figure 1. Three XML documents storing information about books.

1. Introduction

When XML is used for *document-centric* applications, the relative order among the elements is typically important e.g., the relative order of paragraphs and chapters in a book. On the other hand, in case of *data-centric* XML applications, the order among the elements may be unimportant [1]. In this paper we focus on the latter use case. As an example, take in Figure 1 three XML documents storing information about books. While the order of the elements *title*, *year*, *author*, and *editor* may differ from one *book* to another, it has no impact on the semantics of the data stored in this semi-structured database.

A *schema* for XML is a description of the type of admissible documents, typically defining for every node its content model i.e., the children nodes it must, may, or cannot contain. In this paper we study the problem of *learning* unordered schemas from document examples given by the user. For instance, consider the three XML documents from Figure 1 and assume that the user wants to obtain a schema which is satisfied by all the three documents. A desirable solution is a schema which allows a *book* to have, in any order, exactly one *title*, optionally one *year*, and either at least one *author* or at least one *editor*.

Studying the theoretical foundations of learning unordered schemas has several practical motivations. A schema serves as a reference for users who do not know yet the structure of the XML document, and attempt to query or modify its contents. If the schema is not given explicitly, it can be learned from document examples and then read by the users. From another point of view, Florescu [14] pointed out the need to automatically infer good-quality schemas and to apply them in the process of *data integration*. This is clearly a data-centric application, therefore unordered schemas might be more appropriate. Another motivation of learning the unordered schema of a XML collection is *query minimization* [2] i.e., given a query and a schema, find a smaller yet equivalent query in the presence of the schema. Furthermore, we want to use inferred unordered schemas and optimization techniques to boost the learning algorithms for twig queries [26], which are order-oblivious.

Previously, schema learning has been studied from *positive examples* only i.e., documents which must satisfy the schema. For instance, we have already shown a schema learned from the three documents from Figure 1 given as positive examples. However, it is conceivable to find applications where *negative examples* (i.e., documents that must not satisfy the schema) might be useful. For instance, assume a scenario where the schema of a data-centric XML collection evolves over time and some documents may become obsolete w.r.t. the new schema. A user can employ these documents as negative examples to extract the new schema of the collection. Thus, the *schema maintenance* [14]

can be done incrementally, with little feedback needed from the user. This kind of application motivates us to investigate the problem of learning unordered schemas when we also allow negative examples.

We focus our research on learning the unordered schema formalisms recently proposed in [10]: the *disjunctive multiplicity schemas* (DMS) and its restriction, *disjunction-free multiplicity schemas* (MS). While they employ a user-friendly syntax inspired by DTDs, they define unordered content model only, and, therefore, they are better suited for unordered XML. They also retain much of the expressiveness of DTDs without an increase in computational complexity. Essentially, a DMS is a set of rules associating with each label the possible number of occurrences for all the allowed children labels by using *multiplicities*: "$*$" (0 or more occurrences), "+" (1 or more), "?" (0 or 1), "1" (exactly one occurrence; often omitted for brevity). Additionally, alternatives can be specified using restricted *disjunction* ("|") and all the conditions are gathered with *unordered concatenation* ("||"). For example, the following schema is satisfied by the three documents from Figure 1.

$$book \rightarrow title \parallel year^{?} \parallel (author^{+} \mid editor^{+}).$$

This DMS allows a *book* to have, in any order, exactly one *title*, optionally one *year*, and either at least one *author* or at least one *editor*. Moreover, this is a *minimal* schema satisfied by the documents from Figure 1 because it captures the most specific schema satisfied by them. On the other hand, the following schema is also satisfied by the documents from Figure 1, but it is more general:

$$book \rightarrow title \parallel year^{?} \parallel author^{*} \parallel editor^{*}.$$

This schema allows a *book* to have, in any order, exactly one *title*, optionally one *year*, and any number of *author*'s and *editor*'s. It is not minimal because it accepts a *book* having at the same time *author*'s and *editor*'s, unlike the first example of schema. Moreover, the second schema is a MS because it does not use the disjunction operation.

In this paper we address the problem of learning DMS and MS from examples given by the user. We propose a definition of the learnability influenced by computational learning theory [21], in particular by the inference of languages [13, 18]. A learning algorithm takes as input a set of XML documents which must satisfy the schema (i.e., *positive examples*), and a set of XML documents which must not satisfy the schema (i.e., *negative examples*). Essentially, a class of schemas is *learnable* if there exists an algorithm which takes as input a set of examples given by the user and returns a schema which is consistent with the examples. Moreover, the learning algorithm should be *sound* i.e., always return a schema consistent with the examples given by the user, *complete* i.e., able to produce every schema with a sufficiently rich set of examples, and *efficient* i.e., polynomial in the size of the input. Our approach is novel in two directions:

- Previous research on schema learning has been done in the context of ordered XML, typically on learning restricted classes of regular expressions as content models of the DTDs. We focus on learning unordered schema formalisms and the results are positive: the DMS and the MS are learnable from positive examples only.
- The learning frameworks investigated before in the literature typically infer a schema using a collection of documents serving as positive examples. We study the impact of negative examples in the process of schema learning. In this case, the learning algorithm should return a schema satisfied by all the positive examples and by none of the negative ones. We show that the MS are learnable in the presence of both positive and negative examples, while the DMS are not.

We summarize our learnability results in Table 1. For the learnable cases, we propose learning algorithms which return a minimal schema consistent with the examples.

Schema formalism	*+ examples only*	*+ and - examples*
DMS	**Yes** (Th. 4.4)	No (Th. 6.4)
MS	**Yes** (Th. 5.1)	**Yes** (Th. 6.1)

Table 1. Summary of learnability results.

Related work. The *Document Type Definition (DTD)*, the most widespread XML schema formalism [8, 19], is essentially a set of rules associating with each label a regular expression that defines the admissible sequences of children. Therefore, learning DTDs reduces to learning regular expressions. Gold [18] showed that the entire class of regular languages is not identifiable in the limit. Consequently, research has been done on restricted classes of regular expressions which can be efficiently learnable [24]. Hegewald et al. [20] extended the approach from [24] and proposed a system which infers one-unambiguous regular expressions [11] as the content models of the labels. Garofalakis et al. [17] designed a practical system which infers concise and semantically meaningful DTDs from document examples. Bex et al. [6, 7] proposed learning algorithms for two classes of regular expressions which capture many practical DTDs and are succinct by definition: *single occurrence regular expressions* (SOREs) and its subclass consisting of *chain regular expressions* (CHAREs). Bex et al. [5] also studied learning algorithms for the subclass of deterministic regular expressions in which each alphabet symbol occurs at most k times (k-OREs). More recently, Freydenberger and Kötzing [15] proposed more efficient algorithms for the above mentioned restricted classes of regular expressions.

Since the DMS disallow repetitions of symbols among the disjunctions, they can be seen as restricted SOREs interpreted under commutative closure i.e., an unordered collection of children matches a regular expression if there exists an ordering that matches the regular expression in the standard way. The algorithms proposed for the inference of SOREs [7, 15] are typically based on constructing an automaton and then transforming it into an equivalent SORE. Being based on automata techniques, the algorithms for learning SOREs take ordered input, therefore an additional input that the DMS do not have i.e., the order among the labels. For this reason, we cannot reduce learning DMS to learning SOREs. Consequently, we have to investigate new techniques to solve the problem of learning unordered schemas. Moreover, all the existing learning algorithms take into account only positive examples.

We also mention some of the related work on learning schema formalisms more expressive than DTDs. *XML Schema*, the second most widespread schema formalism [8, 19], allow the content model of an element to depend on the context in which it is used, therefore it is more difficult to learn. Bex et al. [9] proposed efficient algorithms to automatically infer a concise XML Schema describing a given set of XML documents. In a different approach, Chidlovskii [12] used *extended context-free grammars* to model schemas for

XML and proposed a schema extraction algorithm.

Organization. This paper is organized as follows. In Section 2 we present preliminary notions. In Section 3 we formally define the learning framework. In Section 4 and Section 5 we present the learnability results for DMS and MS, respectively, when only positive examples are allowed. In Section 6 we discuss the impact of negative examples on learning. Finally, we summarize our results and outline further directions in Section 7.

2. Preliminaries

Throughout this paper we assume an alphabet Σ which is a finite set of symbols. We also assume that Σ has a total order $<_\Sigma$, that can be tested in constant time.

Trees. We model XML documents with unordered labeled trees. Formally, a *tree* t is a tuple $(N_t, root_t, lab_t, child_t)$, where N_t is a finite set of nodes, $root_t \in N_t$ is a distinguished root node, $lab_t : N_t \to \Sigma$ is a labeling function, and $child_t \subseteq N_t \times N_t$ is the parent-child relation. We assume that the relation $child_t$ is acyclic and require every non-root node to have exactly one predecessor in this relation. By *Tree* we denote the set of all finite trees. We present an example of tree in Figure 2.

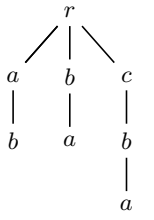

Figure 2. An example of tree.

Unordered words. An *unordered word* is essentially a multiset of symbols i.e., a function $w : \Sigma \to \mathbb{N}_0$ mapping symbols from the alphabet to natural numbers, and we call $w(a)$ the number of occurrences of the symbol a in w. We denote by W_Σ the set containing all the unordered words over the alphabet Σ. We also write $a \in w$ as a shorthand for $w(a) \neq 0$. An empty word ε is an unordered word that has 0 occurrences of every symbol i.e., $\varepsilon(a) = 0$ for every $a \in \Sigma$. We often use a simple representation of unordered words, writing each symbol in the alphabet the number of times it occurs in the unordered word. For example, when the alphabet is $\Sigma = \{a, b, c\}$, $w_0 = aaacc$ stands for the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$.

The (unordered) concatenation of two unordered words w_1 and w_2 is defined as the multiset union $w_1 \uplus w_2$ i.e., the function defined as $(w_1 \uplus w_2)(a) = w_1(a) + w_2(a)$ for all $a \in \Sigma$. For instance, $aaacc \uplus abbc = aaaabbccc$. Note that ε is the identity element of the unordered concatenation $\varepsilon \uplus w = w \uplus \varepsilon = w$ for all unordered word w. Also, given an unordered word w, by w^i we denote the concatenation $w \uplus \ldots \uplus w$ (i times).

A *language* is a set of unordered words. The unordered concatenation of two languages L_1 and L_2 is a language $L_1 \uplus L_2 = \{w_1 \uplus w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}$. For instance, if $L_1 = \{a, aac\}$ and $L_2 = \{ac, b, \varepsilon\}$, then $L_1 \uplus L_2 = \{a, ab, aac, aabc, aaacc\}$.

Multiplicity schemas. A *multiplicity* is an element from the set $\{*, +, ?, 0, 1\}$. We define the function $[\![\cdot]\!]$ mapping multiplicities to sets of natural numbers. More precisely:

$$[\![*]\!] = \{0, 1, 2, \ldots\}, \quad [\![+]\!] = \{1, 2, \ldots\}, \quad [\![?]\!] = \{0, 1\},$$
$$[\![1]\!] = \{1\}, \qquad [\![0]\!] = \{0\}.$$

Given a symbol $a \in \Sigma$ and a multiplicity M, the language of a^M, denoted $L(a^M)$, is $\{a^i \mid i \in [\![M]\!]\}$. For example, $L(a^+) = \{a, aa, \ldots\}$, $L(b^0) = \{\varepsilon\}$, and $L(c^?) = \{\varepsilon, c\}$.

A *disjunctive multiplicity expression* E is:

$$E := D_1^{M_1} \parallel \ldots \parallel D_n^{M_n},$$

where for all $1 \leqslant i \leqslant n$, M_i is a multiplicity and each D_i is:

$$D_i := a_1^{M'_1} \mid \ldots \mid a_k^{M'_k},$$

where for all $1 \leqslant j \leqslant k$, M'_j is a multiplicity and $a_j \in \Sigma$. Moreover, we require that every symbol $a \in \Sigma$ is present at most once in a disjunctive multiplicity expression. For instance, $(a \mid b) \parallel (c \mid d)$ is a disjunctive multiplicity expression, but $(a \mid b) \parallel c \parallel (a \mid d)$ is not because a appears twice. A *disjunction-free multiplicity expression* is an expression which uses no disjunction symbol "|" i.e., an expression of the form $a_1^{M_1} \parallel \ldots \parallel a_k^{M_k}$, where the a_i's are pairwise distinct symbols in the alphabet and the M_i's are multiplicities (with $1 \leqslant i \leqslant k$). We denote by *DME* the set of all the disjunctive multiplicity expressions and by *ME* the set of all the disjunction-free multiplicity expressions.

The language of a disjunctive multiplicity expression is:

$$L(a_1^{M_1} \mid \ldots \mid a_k^{M_k}) = L(a_1^{M_1}) \cup \ldots \cup L(a_k^{M_k}),$$
$$L(D^M) = \{w_1 \uplus \ldots \uplus w_i \mid w_1, \ldots, w_i \in L(D) \wedge i \in [\![M]\!]\},$$
$$L(D_1^{M_1} \parallel \ldots \parallel D_n^{M_n}) = L(D_1^{M_1}) \uplus \ldots \uplus L(D_n^{M_n}).$$

If an unordered word w belongs to the language of a disjunctive multiplicity expression E, we denote it by $w \models E$, and we say that w *satisfies* E. When a symbol a (resp. a disjunctive multiplicity expression E) has multiplicity 1, we often write a (resp. E) instead of a^1 (resp. E^1). Moreover, we omit writing symbols and disjunctive multiplicity expressions with multiplicity 0. Take, for instance, $E_0 = a^+ \parallel (b \mid c) \parallel d^?$ and note that both the symbols b and c as well as the disjunction $(b \mid c)$ have an implicit multiplicity 1. The language of E_0 is:

$$L(E_0) = \{a^i b^j c^k d^\ell \mid i, j, k, \ell \in \mathbb{N}_0,\ i \geqslant 1,\ j + k = 1,\ \ell \leqslant 1\}.$$

Next, we recall the unordered schema formalisms from [10]:

Definition 2.1 *A* disjunctive multiplicity schema (DMS) *is a tuple $S = (root_S, R_S)$, where $root_S \in \Sigma$ is a designated root label and R_S maps symbols in Σ to disjunctive multiplicity expressions. By DMS we denote the set of all disjunctive multiplicity schemas. A* disjunction-free multiplicity schema (MS) *$S = (root_S, R_S)$ is a restriction of the DMS, where R_S maps symbols in Σ to disjunction-free multiplicity expressions. By MS we denote the set of all disjunction-free multiplicity schemas.*

To define satisfiability of a DMS S by a tree t we first define the unordered word ch_t^n of children of a node $n \in N_t$ i.e.,

$$ch_t^n(a) = |\{m \in N_t \mid (n, m) \in child_t \wedge lab_t(m) = a\}|.$$

Now, a tree t *satisfies* S, in symbols $t \models S$, if $lab_t(root_t) = root_S$ and for any node $n \in N_t$, $ch_t^n \in L(R_S(lab_t(n)))$. By $L(S) \subseteq Tree$ we denote the set of all the trees satisfying S.

In the sequel, we present a schema $S = (root_S, R_S)$ as a set of rules of the form $a \to R_S(a)$, for any $a \in \Sigma$. If

$L(R_S(a)) = \varepsilon$, then we write $a \to \epsilon$ or we simply omit writing such a rule.

Example 2.2 We present schemas S_1, S_2, S_3, S_4 illustrating the formalisms defined above. They have the root label r and the rules:

$$\begin{array}{lllll} S_1: & r \to a \parallel b^* \parallel c^? & a \to b^? & b \to a^? & c \to b \\ S_2: & r \to c \parallel b \parallel a & a \to b^? & b \to a & c \to b \\ S_3: & r \to (a \mid b)^+ \parallel c & a \to b^? & b \to a^? & c \to b \\ S_4: & r \to (a \mid b \mid c)^* & a \to \epsilon & b \to a^? & c \to b \end{array}$$

S_1 and S_2 are MS, while S_3 and S_4 are DMS. The tree from Figure 2 satisfies only S_1 and S_3. ▫

Note that there exist DMS such that the smallest tree in their language has a size exponential in the size of the alphabet, as we observe in the following example.

Example 2.3 We consider for $n > 1$ the alphabet $\Sigma = \{r, a_1, b_1, \ldots, a_n, b_n\}$ and the DMS S_5 having the root label r and the following rules:

$$\begin{array}{l} r \to a_1 \parallel b_1, \\ a_i \to a_{i+1} \parallel b_{i+1} \quad (\text{for } 1 \leqslant i < n), \\ b_i \to a_{i+1} \parallel b_{i+1} \quad (\text{for } 1 \leqslant i < n), \\ a_n \to \epsilon, \\ b_n \to \epsilon. \end{array}$$

We present in Figure 3 the unique tree satisfying this schema and we observe that its size is exponential in the size of the alphabet. ▫

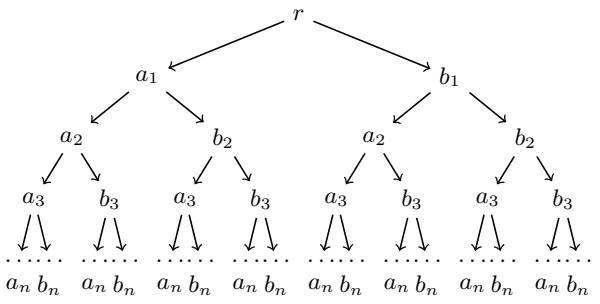

Figure 3. The unique tree satisfying the schema S_5.

Alternative definition with characterizing triples. Any disjunctive multiplicity expression E can be expressed alternatively by its (*characterizing*) *triple* (C_E, N_E, P_E) consisting of the following sets:

- The *conflicting pairs of siblings* C_E contains pairs of symbols in Σ such that E defines no word using both symbols simultaneously:

$$C_E = \{(a_1, a_2) \in \Sigma \times \Sigma \mid \nexists w \in L(E).\ a_1 \in w \wedge a_2 \in w\}.$$

- The *extended cardinality map* N_E captures for each symbol in the alphabet the possible numbers of its occurrences in the unordered words defined by E:

$$N_E = \{(a, w(a)) \in \Sigma \times \mathbb{N}_0 \mid w \in L(E)\}.$$

- The *sets of required symbols* P_E which captures symbols that must be present in every word; essentially, a set of symbols X belongs to P_E if every word defined by E contains at least one element from X:

$$P_E = \{X \subseteq \Sigma \mid \forall w \in L(E).\ \exists a \in X.\ a \in w\}.$$

As an example we take $E_0 = a^+ \parallel (b \mid c) \parallel d^?$. Because P_E is closed under supersets, we list only its minimal elements:

$$C_{E_0} = \{(b, c), (c, b)\}, \qquad P_{E_0} = \{\{a\}, \{b, c\}, \ldots\},$$
$$N_{E_0} = \{(b, 0), (b, 1), (c, 0), (c, 1), (d, 0), (d, 1), (a, 1), (a, 2), \ldots\}.$$

Two equivalent disjunctive multiplicity expressions yield the same triples and hence (C_E, N_E, P_E) can be viewed as the *normal form* of a given expression E [10]. Moreover, each set has a compact representation of size polynomial in the size of the alphabet and computable in PTIME. We illustrate them on the same $E_0 = a^+ \parallel (b \mid c) \parallel d^?$:

- C^*_E consists of sets of symbols present in E such that any pairwise two of them are conflicting:

$$C^*_{E_0} = \{\{b, c\}\}.$$

- N^*_E is a function mapping symbols to multiplicities such that for any unordered word $w \in L(E)$, and for any symbol $a \in \Sigma$, $w(a) \in [\![N^*_E(a)]\!]$:

$$N^*_{E_0}(a) = +, \quad N^*_{E_0}(b) = N^*_{E_0}(c) = N^*_{E_0}(d) = ?.$$

- P^*_E contains only the $\subseteq$-minimal elements of P_E:

$$P^*_{E_0} = \{\{a\}, \{b, c\}\}.$$

Also note that we can easily construct a disjunctive multiplicity expression from its characterizing triple. A simple algorithm has to loop over the sets from C^*_E and P^*_E to compute for each label with which other labels it is linked by the disjunction operator. Then, using N^*_E, the algorithm associates to each label and each disjunction the correct multiplicity. For example, take the following compact triples:

$$C^*_{E_1} = \{\{a, e\}, \{c, d\}\}, \qquad P^*_{E_1} = \{\{a, e\}, \{b\}\},$$
$$N^*_{E_1}(a) = *, \quad N^*_{E_1}(b) = 1, \quad N^*_{E_1}(c) = N^*_{E_1}(d) = N^*_{E_1}(e) = ?.$$

Note that they characterize the expression:

$$E_1 = (a^+ \mid e) \parallel b \parallel (c^? \mid d^?).$$

We have introduced the alternative definition with characterizing triples because we later propose an algorithm which learns characterizing triples from unordered word examples (Algorithm 1 from Section 4). Then, from this information, the corresponding disjunctive multiplicity expression can be constructed in a straightforward manner.

3. Learning framework

We use a variant of the standard language inference framework [13, 18] adapted to learning disjunctive multiplicity expressions and schemas. A *learning setting* is a tuple containing the set of *concepts* that are to be learned, the set of *instances* of the concepts that are to serve as examples in learning, and the *semantics* mapping every concept to its set of instances.

Definition 3.1 *A* learning setting *is a tuple $(\mathcal{E}, \mathcal{C}, \mathcal{L})$, where $\mathcal{E}$ is a set of examples, $\mathcal{C}$ is a class of concepts, and $\mathcal{L}$ is a function that maps every concept in $\mathcal{C}$ to the set of all its examples (a subset of $\mathcal{E}$).*

For example, the setting for learning disjunctive multiplicity expressions from positive examples is the tuple (W_Σ, DME, L) and the setting for learning disjunctive multiplicity schemas from positive examples is $(Tree, DMS, L)$. We obtain analogously the learning settings for disjunction-free multiplicity expressions and schemas: (W_Σ, ME, L) and

$(Tree, MS, L)$, respectively. The general formulation of the definition allows us to easily define settings for learning from both positive and negative examples, which we present in Section 6.

To define a learnable concept, we fix a learning setting $\mathcal{K} = (\mathcal{E}, \mathcal{C}, \mathcal{L})$ and we introduce some auxiliary notions. A *sample* is a finite nonempty subset D of $\mathcal{E}$ i.e., a set of examples. A sample D is *consistent* with a concept $c \in \mathcal{C}$ if $D \subseteq \mathcal{L}(c)$. A *learning algorithm* is an algorithm that takes a sample and returns a concept in $\mathcal{C}$ or a special value *null*.

Definition 3.2 *A class of concepts $\mathcal{C}$ is* learnable in polynomial time and data *in the setting $\mathcal{K} = (\mathcal{E}, \mathcal{C}, \mathcal{L})$ if there exists a polynomial learning algorithm learner satisfying the following two conditions:*

1. **Soundness.** *For any sample D, the algorithm learner(D) returns a concept consistent with D or a special* null *value if no such concept exists.*
2. **Completeness.** *For any concept $c \in \mathcal{C}$ there exists a sample CS_c such that for every sample D that extends CS_c consistently with c i.e., $CS_c \subseteq D \subseteq \mathcal{L}(c)$, the algorithm learner$(D)$ returns a concept equivalent to c. Furthermore, the cardinality of CS_c is polynomially bounded by the size of the concept.*

The sample CS_c is called the *characteristic sample* for c w.r.t. *learner* and $\mathcal{K}$. For a learning algorithm there may exist many such samples. The definition requires that one characteristic sample exists. The soundness condition is a natural requirement, but alone it is not sufficient to eliminate trivial learning algorithms. For instance, if we want to learn disjunctive multiplicity expressions from positive examples over the alphabet $\{a_1, \ldots, a_n\}$, an algorithm always returning $a_1^* \parallel \ldots \parallel a_n^*$ is sound. Consequently, we require the algorithm to be complete analogously to how it is done for grammatical language inference [13, 18].

Typically, in the case of polynomial grammatical inference, the *size* of the characteristic sample is required to be polynomial in the size of the concept to be learned [13], where the size of a sample is the sum of the sizes of the examples that it contains. From the definition of the DMS, since repetitions of symbols are discarded among the disjunctions, the size of a schema is polynomial in the size of the alphabet. Thus, a natural requirement would be that the size of the characteristic sample is polynomially bounded by the size of the alphabet. There exist DMS such that the smallest tree in their language is exponential in the size of the alphabet (cf. Example 2.3). Because of space restrictions, we have imposed in the definition of learnability that the *cardinality* (and not the size) of the characteristic sample is polynomially bounded by the size of the concept, hence by the size of the alphabet. However, we are able to obtain characteristic samples of size polynomial in the size of the alphabet by using a *compressed* representation of the XML trees, for example with *directed acyclic graphs* [23]. We will provide in the full version of the paper the details about this compression technique and the new definition of the learnability. The algorithms that we propose in this paper transfer without any alteration for the definition using compressed trees.

Additionally to the conditions imposed by the definition of learnability, we are interested in the existence of learning algorithms which return *minimal* concepts for a given set of examples. It is important to emphasize that we mean minimality in terms on language inclusion. When only positive examples are allowed, a DMS S is a *minimal* DMS consistent with a set of trees D iff $D \subseteq L(S)$, and, for any $S' \neq S$, if $D \subseteq L(S')$, then $L(S') \not\subset L(S)$. We similarly obtain the definition of minimality for learning disjunctive multiplicity expressions. Intuitively, a minimal schema consistent with a set of examples is the most specific schema consistent with them. For example, recall the three XML documents storing information about books from Figure 1. Assume that the user provides the three documents as positive examples to a learning algorithm. The most specific schema consistent with the examples is:

$$book \rightarrow title \parallel year^? \parallel (author^+ \mid editor^+).$$

Another possible solution is the schema:

$$book \rightarrow title \parallel year^? \parallel author^* \parallel editor^*.$$

It is less likely that a user wants to obtain such a schema which allows a *book* to have at the same time *author*'s and *editor*'s. In this case, the most specific schema also corresponds to the natural requirements that one might want to impose on a XML collection storing information about books, in particular a *book* has either at least one *author* or at least one *editor*. Minimality is often perceived as a better fitted learning solution [3–5, 16], and this motivates our requirement for the learning algorithms to return minimal concepts consistent with the examples.

4. Learning DMS from positive examples

The main result of this section is the learnability of the disjunctive multiplicity schemas from positive examples i.e., in the setting $(Tree, DMS, L)$. We present a learning algorithm that constructs a *minimal* schema consistent with the input set of trees.

First, we study the problem of learning a disjunctive multiplicity expression from positive examples i.e., in the setting (W_Σ, DME, L). We present a learning algorithm that constructs a minimal disjunctive multiplicity expression consistent with the input collection of unordered words. Given a set of unordered words, there may exist many consistent minimal disjunctive multiplicity expressions. In fact, for some sets of positive examples there may be an exponential number of such expressions (cf. the proof of Lemma 6.2). Take in Example 4.1 a sample and two consistent minimal disjunctive multiplicity expressions.

Example 4.1 Consider the alphabet $\Sigma = \{a, b, c, d, e\}$ and the set of unordered words $D = \{aabc, abd, be\}$. Take the following two disjunctive multiplicity expressions:

$$E_1 = (a^+ \mid e) \parallel b \parallel (c^? \mid d^?),$$
$$E_2 = a^* \parallel b \parallel (c \mid d \mid e).$$

Note that $D \subseteq L(E_1)$ and $D \subseteq L(E_2)$. Also note that $L(E_1) \not\subseteq L(E_2)$ (because of bce) and $L(E_2) \not\subseteq L(E_1)$ (because of abe). On the other hand, we easily observe that both E_1 and E_2 are minimal disjunctive multiplicity expressions with languages including D. ▫

Before we present the learning algorithms, we have to introduce additional notions. First, we define the function $min\_fit\_multiplicity(\cdot)$ which, given a set of unordered words D and a label $a \in \Sigma$, computes the multiplicity M such that $\forall w \in D.\ w(a) \in [\![M]\!]$ and there does not exist another multiplicity M' such that $[\![M']\!] \subset [\![M]\!]$ and $\forall w \in D.\ w(a) \in [\![M']\!]$. For example, given the set of unordered

words $D = \{aabc, abd, be\}$, we have:

$$min\_fit\_multiplicity(D, a) = *,$$
$$min\_fit\_multiplicity(D, b) = 1,$$
$$min\_fit\_multiplicity(D, c) = ?.$$

Next, we introduce the notion of *maximal-clique partition of a graph*. Given a graph $G = (V, E)$, a maximal-clique partition of G is a graph partition $(V_1, \ldots, V_k)$ such that:

- The subgraph induced in G by any V_i is a clique (with $1 \leqslant i \leqslant k$),
- The subgraph induced in G by the union of any V_i and V_j is not a clique (with $1 \leqslant i \neq j \leqslant k$).

In Figure 4 we present a graph and a maximal-clique partition of it i.e., $\{\{a, e\}, \{b\}, \{c, d\}\}$. Note that the graph from Figure 4 allows one other maximal-clique partition i.e., $\{\{a\}, \{b\}, \{c, d, e\}\}$. On the other hand, $\{\{a\}, \{b\}, \{c, d\}, \{e\}\}$ is not a maximal-clique partition because it contains two sets such that their union induces a clique i.e., $\{a\}$ and $\{e\}$.

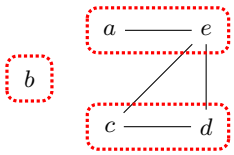

Figure 4. A graph and a maximal-clique partition of it. Vertices from the same rectangle belong to the same set.

Unlike the *clique* problem, which is known to be NP-complete [25], we can partition in PTIME a graph in maximal cliques with a greedy algorithm. In the sequel, we assume that the vertices of the graph are labels from Σ. For a given graph there may exist many maximal-clique partitions and we use the total order $<_\Sigma$ to propose a deterministic algorithm constructing a maximal-clique partition. The algorithm works as follows: we take the smallest label from Σ w.r.t. $<_\Sigma$ and not yet used in a clique, and we iteratively extend it to a maximal clique by adding connected labels. Every time when we have a choice to add a new label to the current clique, we take the smallest label w.r.t. $<_\Sigma$. We repeat this until all the labels are used. This algorithm yields to a unique maximal-clique partition. For example, for the graph from Figure 4, we compute the maximal-clique partition marked on the figure i.e., $\{\{a, e\}, \{b\}, \{c, d\}\}$. We additionally define the function $max\_clique\_partition(\cdot)$ which takes as input a graph, computes a maximal-clique partition using the greedy algorithm described above and, at the end, for technical reasons, the algorithm discards the singletons. For example, for the graph from Figure 4, the function $max\_clique\_partition(\cdot)$ returns $\{\{a, e\}, \{c, d\}\}$. Clearly, the function $max\_clique\_partition(\cdot)$ works in PTIME.

Next, we present Algorithm 1 and we claim that, given a set of unordered words D, it computes in polynomial time a disjunctive multiplicity expression E consistent with D. Algorithm 1 works in three steps and we illustrate each of them on the sample $D = \{aabc, abd, be\}$ from Example 4.1. The first step (lines 1-2) computes the compact representation of the extended cardinality map for each symbol from Σ, using the function $min\_fit\_multiplicity(\cdot)$. We ignore in the sequel the symbols never occurring in words from D (line 3). For the sample from Example 4.1, we infer:

$$N^*_E(a) = *, \qquad N^*_E(b) = 1,$$
$$N^*_E(c) = N^*_E(d) = N^*_E(e) = ?.$$

Algorithm 1 Learning disjunctive multiplicity expressions from positive examples.

algorithm $learner^+_{DME}(D)$
Input: A set of unordered words $D = \{w_1, \ldots, w_n\}$
Output: A minimal disjunctive multiplicity expression E consistent with D

1: **for** $a \in \Sigma$ **do**
2: **let** $N^*_E(a) = min\_fit\_multiplicity(D, a)$
3: **let** $\Sigma' = \{a \in \Sigma \mid N^*_E(a) \in \{?, 1, *, +\}\}$
4: **let** $G = (\Sigma',\ \{(a, b) \in \Sigma' \times \Sigma' \mid \forall w \in D.\ a \notin w \vee b \notin w\})$
5: **let** $C^*_E = max\_clique\_partition(G)$
6: **let** $P^*_E = \{\{a\} \mid N^*_E(a) \in \{1, +\}\}$
$\cup\ \{X \in C^*_E \mid \forall w \in D.\ \exists a \in X.\ a \in w\}$
7: **return** E characterized by the triple (C^*_E, N^*_E, P^*_E)

The second step of the algorithm (lines 4-5) computes the compact sets of conflicting siblings. First, we construct the graph G having as set of vertices the labels occurring at least once in unordered words from D. Two labels are linked by an edge in G if there does not exist an unordered word in D where both of them are present at the same time, in other words the two labels are a candidate pair of conflicting siblings. Next, we apply the function $max\_clique\_partition(\cdot)$ on the graph G. For the unordered words from Example 4.1 we obtain the graph from Figure 4, and we infer $C^*_E = \{\{a, e\}, \{c, d\}\}$. Note that the maximal-clique partition implies the minimality of the disjunctive multiplicity expression constructed later using the inferred C^*_E.

The third step of the algorithm (line 6) computes the $\subseteq$-minimal sets of required symbols P^*_E. Each symbol having associated a multiplicity 1 or + belongs to a required set of symbols containing only itself because it is present in all the unordered words from D and we want to learn a minimal concept. Moreover, we add in P^*_E the sets of conflicting siblings inferred at the previous step with the property that one of them is present in any unordered word from D, to guarantee the minimality of the inferred language. For the sample from Example 4.1, $\{b\}$ belongs to P^*_E. Since from the previous step we have $C^*_E = \{\{a, e\}, \{c, d\}\}$, at this step we have to add $\{a, e\}$ to P^*_E because all the words in the sample contain either a or e. On the other hand, we do not add $\{c, d\}$ because the sample contains the word be. The inferred P^*_E is $\{\{a, e\}, \{b\}\}$.

Finally, the algorithm returns the disjunctive multiplicity expression characterized by the inferred triple (line 7). For the sample D, it returns $E = (a^+ \mid e) \parallel b \parallel (c^? \mid d^?)$. Note that if at step 2 we take a partition which is not a maximal-clique one, for example $\{\{a\}, \{b\}, \{c, d\}, \{e\}\}$, and we later construct a disjunctive multiplicity expression using it, we get $a^* \parallel b \parallel (c^? \mid d^?) \parallel e^?$, which includes both E_1 and E_2 from Example 4.1, therefore is not minimal. Also note that at step 3, without $\{a, e\}$ added to P^*_E, the resulting schema would accept an unordered word without any a and e, so the learned language would not be minimal.

Algorithm 1 is sound and each of its three steps requires polynomial time. Next, we prove the completeness of the algorithm. Given a disjunctive multiplicity expression E, we construct in three steps its characteristic sample CS_E. At the same time, we illustrate the construction on the disjunctive multiplicity expression $E_1 = (a^+ \mid e) \| b \| (c^? \mid d^?)$:

1. We take the pairs of symbols which can be found together in an unordered word in $L(E)$. For each of them, we add in CS_E an unordered word containing only

the two symbols. Next, for each symbol occurring in the disjunctions from E, we add in CS_E an unordered word containing only one occurrence of that symbol. We also add in CS_E the empty word. For E_1 we obtain: $\{ab, ac, ad, bc, bd, be, ce, de, a, b, c, d, e, \varepsilon\}$.

2. We replace each unordered word w obtained at the previous step with $w \uplus w'$, where w' is a minimal unordered word such that $w \uplus w' \in L(E)$. The newly obtained CS_E contains unordered words from $L(E)$. For E_1 we obtain: $\{ab, abc, abd, be, bce, bde\}$.
3. For each symbol a from the alphabet such that $N_E^*(a)$ is $*$ or $+$, we randomly take an unordered word w from CS_E and containing a and we add to CS_E the unordered word $w \uplus a$. In the worst case, at this step the number of words in the characteristic sample is doubled, but it remains polynomial in the size of the alphabet. For E_1 we obtain: $\{ab, aab, abc, abd, be, bce, bde\}$.

Note that there may exist many equivalent characteristic samples. The first step of the construction implies that the only potential conflicts to be considered in Algorithm 1 are the conflicts implied by the expression. In other words, all the connected components of the graph of potential conflicts from Algorithm 1 are cliques. Thus, there is only one possible maximal-clique partition to be done in the algorithm. Moreover, the second and third steps of the construction ensure that, for any sample consistently extending the characteristic sample, Algorithm 1 infers the correct sets of required symbols and the extended cardinality map, respectively.

We have proposed Algorithm 1, which is a sound and complete algorithm for learning minimal disjunctive multiplicity expressions from unordered words positive examples. Thus, we can state the following result:

Lemma 4.2 *The concept class DME is learnable in polynomial time and data from positive examples i.e., in the setting* (W_Σ, DME, L).

Next, we extend the result for DMS. We propose Algorithm 2, which learns a disjunctive multiplicity schema from a set of trees. We assume w.l.o.g. that all the trees from the sample have as root label the same label r. If this assumption is not satisfied, the sample is not consistent. The algorithm infers, for each label a from the alphabet, the minimal disjunctive multiplicity expression consistent with the children of all the nodes labeled a from the trees from the sample.

Algorithm 2 Learning DMS from positive examples.

algorithm: $learner^+_{DMS}(D)$
Input: A set of trees $D = \{t_1, \ldots, t_n\}$ s.t. $lab_{t_i}(root_{t_i}) = r$ (with $1 \leqslant i \leqslant n$)
Output: A minimal DMS S consistent with D

```
for a ∈ Σ do
    let D' = {ch_t^n | t ∈ D. n ∈ N_t. lab_t(n) = a}
    let R_S(a) = learner^+_DME(D')
return S = (r, R_S)
```

Algorithm 2 returns a minimal disjunctive multiplicity schema consistent with the sample because the inferred rule for each label represents a minimal disjunctive multiplicity expression obtained using Algorithm 1. Next, we show that Algorithm 2 is also complete by providing a construction of a characteristic sample of cardinality polynomial in the size of the alphabet. For this purpose, we have to define first two additional notions. Given a DMS $S = (root_S, R_S)$ and a label $a \in \Sigma$, we define the following two trees:

- $\min_{t\uparrow(S,a)}$ is a minimal tree satisfying S and containing a node labeled a,
- $\min_{t\downarrow(S,a)}$ is a minimal tree satisfying $S' = (a, R_S)$. It is equivalent to $\min_{t\uparrow(S',a)}$.

We illustrate the two notions defined above in the following example:

Example 4.3 Consider the DMS S having the root label r and the rules:

$$r \to a^* \parallel (b \mid c) \qquad a \to d^?$$
$$b, c \to e^+ \qquad d, e \to \epsilon$$

We present in Figure 5 some trees and we explain for each of them how it can be used. ▫

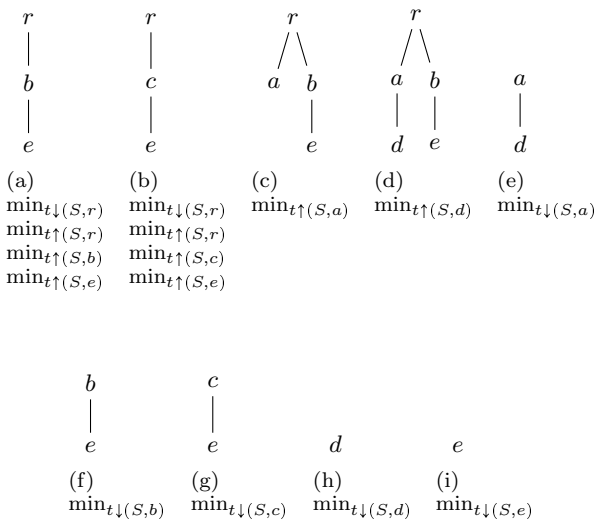

Figure 5. Trees used for Example 4.3.

Next, we present the construction of the characteristic sample for learning a DMS from positive examples. We take a DMS $S = (root_S, R_S)$ over an alphabet Σ and we assume w.l.o.g. that any symbol of the alphabet can be present in at least one tree from $L(S)$. For each $a \in \Sigma$, for each $w \in CS_{R_S(a)}$, we compute a tree t as follows: we generate a tree $\min_{t\uparrow(S,a)}$, we take the node labeled by a (let it n_a), and for any $b \in \Sigma$, while $ch_t^{n_a}(b) < w(b)$ we fuse in n_a a copy of $\min_{t\downarrow(S,b)}$. We obtain a sample of cardinality polynomially bounded by the size of the alphabet. Given a DMS S, there may exist many characteristic samples CS_S. Each of them has the property that, if we construct a sample D which extends CS_S consistently with S, then $learner^+_{DMS}(D)$ returns S. This proves the completeness of Algorithm 2.

We illustrate the construction of the characteristic sample on the schema S from Example 4.3. Recall that we have already presented the trees $\min_{t\uparrow(S,a)}$ and $\min_{t\downarrow(S,a)}$ for each a from the alphabet. We also construct the characteristic samples for the disjunctive multiplicity expressions from the rules of S:

- $CS_{R_S(r)} = \{aab, ab, ac, b, c\}$,
- $CS_{R_S(a)} = \{\varepsilon, d\}$,
- $CS_{R_S(b)} = CS_{R_S(c)} = \{e, ee\}$,
- $CS_{R_S(d)} = CS_{R_S(e)} = \{\varepsilon\}$.

In Figure 6 we present a characteristic sample CS_S for the DMS S and we explain the purpose of each tree:

- (a), (b), (c), (d), and (e) ensure that there is inferred the correct rule for the root i.e., $R_S(r)$,
- (b) and (f) ensure that there is inferred the correct $R_S(a)$,
- (d) and (g) ensure that there is inferred the correct $R_S(b)$,
- (e) and (h) ensure that there is inferred the correct $R_S(c)$,
- The nodes labeled by d and e never have children in the trees from CS_S, so there are inferred the correct rules for $R_S(d)$ and $R_S(e)$.

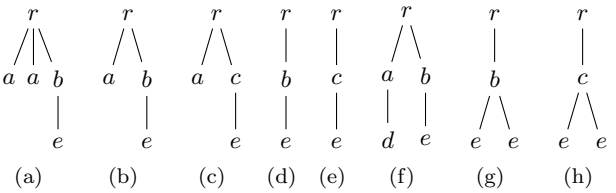

Figure 6. Characteristic sample for the schema S from Example 4.3.

We have proposed Algorithm 2, which is a sound and complete algorithm for learning disjunctive multiplicity schemas from trees positive examples. Thus, we can state the main result of this section:

Theorem 4.4 *The concept class DMS is learnable in polynomial time and data from positive examples i.e., in the setting* $(Tree, DMS, L)$.

5. Learning MS from positive examples

In this section we show that the MS are learnable from positive examples i.e., in the setting $(Tree, MS, L)$. Recall that the MS allow no disjunction in the rules, in other words they use expressions of the form $a_1^{M_1} \| \ldots \| a_n^{M_n}$. Due to this very particular form, we can *capture* a MS $S = (root_S, R_S)$ using a function $\mu : \Sigma \times \Sigma \to \{0, 1, ?, +, *\}$ obtained directly from the rules of S:

$$a \to a_1^{\mu(a,a_1)} \| \ldots \| a_n^{\mu(a,a_n)}.$$

For example, given the schema S having the root r and the rules:

$$r \to a^+ \| b, \qquad a \to b^*, \qquad b \to a^? \| b^?,$$

we have :

$$\begin{array}{lll} \mu(r,a) = +, & \mu(r,b) = 1, & \mu(r,r) = 0, \\ \mu(a,a) = 0, & \mu(a,b) = *, & \mu(a,r) = 0, \\ \mu(b,a) = ?, & \mu(b,b) = ?, & \mu(b,r) = 0. \end{array}$$

Note that given the function $\mu(\cdot)$ we can easily construct the initial S. We use this characterization in Algorithm 3, a polynomial and sound algorithm which learns a minimal MS from a set of trees. We assume w.l.o.g. that all the trees from the sample have as root label the same label r. If this assumption is not satisfied, the sample is not consistent. The minimality of the algorithm follows from the minimality of the inferred multiplicity for each pair of labels (a, b), using the function *min_fit_multiplicity*$(\cdot)$ (cf. Section 4). Moreover, Algorithm 3 is complete. We can easily construct a characteristic sample of cardinality polynomial in the size of the alphabet by using the same steps provided in the previous section, for unordered words and for trees.

Algorithm 3 Learning MS from positive examples.

algorithm $learner^+_{MS}(D)$
Input A set of trees $D = \{t_1, \ldots, t_n\}$ s.t. $lab_{t_i}(root_{t_i}) = r$ (with $1 \leqslant i \leqslant n$)
Output A minimal MS S consistent with D

```
for a ∈ Σ do
    let D' = {ch_t^n | t ∈ D. n ∈ N_t. lab_t(n) = a}
    for b ∈ Σ do
        let μ(a, b) = min_fit_multiplicity(D', b)
return S having the root label r and captured by μ
```

We have proposed a sound and complete algorithm which learns a minimal MS consistent with a set of positive examples, so we can state the following result:

Theorem 5.1 *The concept class MS is learnable in polynomial time and data from positive examples i.e., in the setting* $(Tree, MS, L)$.

6. Impact of negative examples

In the previous sections, we have considered the settings where the user provides positive examples only. In this section, we allow the user to additionally specify negative examples. The main results of this section are that the MS are learnable in polynomial time and data in the presence of both positive and negative examples, while the DMS are not. We use two symbols $+$ and $-$ to mark whether an example is positive or negative, and we define:

- $W_\Sigma^\pm = W_\Sigma \times \{+, -\}$,
- $L^\pm(E) = \{(w, +) \mid w \in L(E)\} \cup \{(w, -) \mid w \in W_\Sigma \setminus L(E)\}$, where E is a disjunctive multiplicity expression,
- $Tree^\pm = Tree \times \{+, -\}$,
- $L^\pm(S) = \{(t, +) \mid t \in L(S)\} \cup \{(t, -) \mid t \in Tree \setminus L(S)\}$, where S is a disjunctive multiplicity schema.

Formally, the setting for learning disjunctive multiplicity expressions from positive and negative examples is $(W_\Sigma^\pm, DME, L^\pm)$, while for learning DMS from positive and negative examples we have $(Tree^\pm, DMS, L^\pm)$. We obtain analogously the settings for disjunction-free multiplicity expressions and schemas: $(W_\Sigma^\pm, ME, L^\pm)$ and $(Tree^\pm, MS, L^\pm)$, respectively.

We study the problem of checking whether there exists a concept consistent with the input sample because any sound learning algorithm needs to return *null* if and only if there is no such concept. Therefore, consistency checking is an easier problem than learning and its intractability precludes learnability. Formally, given a learning setting $\mathcal{K} = (\mathcal{E}, \mathcal{C}, \mathcal{L})$, the $\mathcal{K}$-*consistency* is the following decision problem:

$$CONS_\mathcal{K} = \{D \subseteq \mathcal{E} \mid \exists c \in \mathcal{C}.\ D \subseteq \mathcal{L}(c)\}.$$

Note that the consistency checking is trivial when only positive examples are allowed. For instance, if we want to learn disjunctive multiplicity expressions from positive examples over the alphabet $\{a_1, \ldots, a_n\}$, the disjunctive multiplicity expression $a_1^* \| \ldots \| a_n^*$ is always consistent with the examples. When we also allow negative examples, the problem becomes more complex, particularly in the case of disjunctive multiplicity expressions and schemas, where this problem is not tractable.

First, we show that the consistency checking is tractable for MS. In Section 5, we have proposed Algorithm 3, which learns a minimal MS consistent with a set of positive examples. Note that, given a set of trees, there exists a *unique minimal MS* consistent with them. The argument is that Algorithm 3 uses the function *min_fit_multiplicity*$(\cdot)$ (cf. Section 4) to infer minimal multiplicities which are unique and sufficient to capture a MS. Thus, the consistency checking becomes trivial for MS: given a sample containing positive and negative examples, there exists a MS consistent with them iff no tree used as negative example satisfies the minimal MS returned by Algorithm 3. Consequently, we easily adapt Algorithm 3 to handle both positive and negative examples and we propose Algorithm 4.

Algorithm 4 Learning MS from positive and negative examples.

algorithm $learner^{\pm}_{MS}(D)$
Input A sample $D = \{(t, \alpha) \mid t \in \mathit{Tree}, \alpha \in \{+, -\}\}$
Output A minimal MS S such that $D \subseteq L^{\pm}(S)$, or *null* if no such schema exists
1: **let** $D' = \{t \in \mathit{Tree} \mid (t, +) \in D\}$
2: **let** $S = learner^{+}_{MS}(D')$
3: **if** $\exists t \in \mathit{Tree}.\ (t, -) \in D \wedge t \in L(S)$ **then**
4: **return** *null*
5: **return** S

Essentially, Algorithm 4 returns the minimal schema consistent with the positive examples iff there is no negative example satisfying it, and otherwise it returns *null*. Note that Algorithm 4 is sound and works in polynomial time in the size of the input. The completeness of Algorithm 4 follows from the completeness of Algorithm 3. Given a MS S, we can construct a characteristic sample CS_S that contains only positive examples, analogously to how it is done for Algorithm 3. We have proposed a polynomial, sound, and complete algorithm which learns minimal MS from positive and negative examples, so we state the first result of this section:

Theorem 6.1 *The concept class MS is learnable in polynomial time and data from positive and negative examples i.e., in the setting* $(\mathit{Tree}^{\pm}, MS, L^{\pm})$.

Next, we prove that the concept class *DMS* is not learnable in polynomial time and data in the setting $DMS^{\pm} = (\mathit{Tree}^{\pm}, DMS, L^{\pm})$. For this purpose, we first show the intractability of learning disjunctive multiplicity expressions from positive and negative examples i.e., in the setting $DME^{\pm} = (W^{\pm}_{\Sigma}, DME, L^{\pm})$. We study the complexity of checking the consistency of a set of positive and negative examples and we prove the intractability of $CONS_{DME^{\pm}}$. Intuitively, this follows from the fact that, given a set of unordered words, there may exist an exponential number of minimal consistent disjunctive multiplicity expressions, and we may need to check all of them to decide whether there exist negative examples satisfying them. Formally, we have the following result:

Lemma 6.2 $CONS_{DME^{\pm}}$ *is NP-complete.*

Proof We prove the NP-hardness by reduction from *3SAT* which is known as being NP-complete. We take a formula φ in 3CNF containing the clauses $c_1, \ldots, c_k$ over the variables $x_1, \ldots, x_n$. We generate a sample D_φ over the alphabet $\Sigma = \{t_1, f_1, \ldots, t_n, f_n\}$ such that:

- $(t_1 f_1 \ldots t_n f_n, +) \in D_\varphi$,
- $(\varepsilon, -) \in D_\varphi$,
- $(t_i f_i, +), (t_i t_i f_i f_i, -) \in D_\varphi$, for $1 \leqslant i \leqslant n$,
- $(w_j, -) \in D_\varphi$, where $w_j = v_{j1} v_{j1} v_{j2} v_{j2} v_{j3} v_{j3}$, for any j such that $1 \leqslant j \leqslant k$, where x_{j1}, x_{j2}, x_{j3} are the literals used in the clause c_j and for any l such that $1 \leqslant l \leqslant 3$, v_{jl} is t_{jl} if x_{jl} is a negative literal in c_j, and f_{jl} otherwise.

For example, for the formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$, we generate the sample:

$$\begin{array}{cc} (t_1 f_1 t_2 f_2 t_3 f_3 t_4 f_4, +), & (\varepsilon, -), \\ (t_1 f_1, +), & (t_1 t_1 f_1 f_1, -), \\ (t_2 f_2, +), & (t_2 t_2 f_2 f_2, -), \\ (t_3 f_3, +), & (t_3 t_3 f_3 f_3, -), \\ (t_4 f_4, +), & (t_4 t_4 f_4 f_4, -), \\ & (f_1 f_1 t_2 t_2 f_3 f_3, -), \\ & (t_1 t_1 f_3 f_3 t_4 t_4, -). \end{array}$$

For a given φ, a valuation is a function $V : \{x_1, \ldots, x_n\} \to \{\mathit{true}, \mathit{false}\}$. Each of the 2^n possible valuations encodes a minimal disjunctive multiplicity expression E_V consistent with the positive examples from D_φ, constructed as follows:

$$E_V = (v_1 \mid \ldots \mid v_n)^+ \,\|\, \overline{v_1}^? \,\|\, \ldots \,\|\, \overline{v_n}^?,$$

where, for $1 \leqslant i \leqslant n$, if $V(x_i) = \mathit{true}$ then $v_i = t_i$ and $\overline{v_i} = f_i$. Otherwise, $v_i = f_i$ and $\overline{v_i} = t_i$. Next, we show that, for any valuation V, $V \models \varphi$ iff E_V is consistent with D_φ.

For the *only if* case, consider a valuation V such that $V \models \varphi$ and we take the corresponding expression $E_V = (v_1 \mid \ldots \mid v_n)^+ \,\|\, \overline{v_1}^? \,\|\, \ldots \,\|\, \overline{v_n}^?$. Note that $t_1 f_1 \ldots t_n f_n$ and all $t_i f_i$'s (with $1 \leqslant i \leqslant n$) satisfy E_V, while ε does not satisfy E_V. Also note that for $1 \leqslant i \leqslant n$, one symbol between t_i and f_i occurs at least once, while the other occurs at most once, so all $t_i t_i f_i f_i$'s do not satisfy E_V. Assume that there is a w_j (with $1 \leqslant j \leqslant k$) such that w_j satisfies E_V, which by construction implies that the clause c_j is not satisfied by the valuation V, which implies a contradiction. Hence, w_j does not satisfy E_V for any $1 \leqslant j \leqslant k$. Therefore, E_V is consistent with D_φ.

For the *if* case, we assume that E_V is consistent with the sample D_φ. Since the w_j's (with $1 \leqslant j \leqslant k$) encode the valuations making the clauses c_j's false and none of the w_j's satisfies E_V, then the valuation V encoded in E_V makes the formula φ satisfiable.

The construction of D_φ also ensures that if there exists a disjunctive multiplicity expression consistent with D_φ, it has the form of E_V. Therefore, $\varphi \in \mathit{3SAT}$ iff $D_\varphi \in CONS_{DME^{\pm}}$.

To prove the membership of $CONS_{DME^{\pm}}$ to NP, we point out that a Turing machine guesses a disjunctive multiplicity expression E, whose size is linear in $|\Sigma|$ since repetitions are discarded among the disjunctions of E. Moreover, checking whether E is consistent with the sample can be easily done in polynomial time. ▫

We extend the above result to $CONS_{DMS^{\pm}}$:

Corollary 6.3 $CONS_{DMS^{\pm}}$ *is NP-complete.*

Proof The NP-hardness of $CONS_{DME^{\pm}}$ implies the NP-hardness of $CONS_{DMS^{\pm}}$: it is sufficient to consider flat trees having all the same root label. Moreover, to prove the membership of $CONS_{DMS^{\pm}}$ to NP, a Turing machine guesses a disjunctive multiplicity schema S, whose size is polynomial in $|\Sigma|$, and checks whether S is consistent with the sample (which can be done in polynomial time). ▫

Since consistency checking in the presence of positive and negative examples is intractable for DMS, we conclude that:

Theorem 6.4 *Unless $P = NP$, the concept class DMS is not learnable in polynomial time and data from positive and negative examples i.e., in the setting $(Tree^{\pm}, DMS, L^{\pm})$.*

7. Conclusions and future work

We have studied the problem of learning unordered XML schemas from examples given by the user. We have investigated the learnability of DMS and MS in two settings: one allowing positive examples only, and one that allows both positive and negative examples. To the best of our knowledge, no research has been done on learning unordered XML schema formalisms, nor on allowing both positive and negative examples in the process of schema learning. We have proven that the DMS are learnable only from positive examples, and we have shown that they are not learnable from positive and negative examples by using the intractability of the consistency checking. Moreover, we have proven that the MS are learnable in both settings: from only positive examples, and also from positive and negative examples. For all the learnable cases we have proposed learning algorithms that return minimal schemas consistent with the examples.

As future work, we want to use a more specific learnability condition i.e., to require the size (instead of the cardinality) of the characteristic sample to be polynomial in the size of the alphabet. Thus, we will fully adhere to the classical definition of the characteristic sample in the context of grammatical inference [13]. Our preliminary research indicates that we are able to do this by using a compressed representation of the XML documents with directed acyclic graphs [23]. The learning algorithms that we propose in this paper will work without any alteration. Moreover, we would like to extend our learning algorithms for more expressive unordered schemas, for instance schemas which allow *numeric occurrences* [22] of the form $a^{[n,m]}$ that generalize multiplicities by requiring the presence of at least n and at most m elements a. Additionally, we want to use the learning algorithms for unordered schemas to boost the existing learning algorithms for twig queries [26]. For this purpose, we have to investigate first the problem of query minimization [2] in the presence of DMS. Next, we want to propose a twig query learning algorithm which infers the schema of the documents and then it uses the schema to improve the quality of the learned twig query.

References

[1] S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. In *ICDT*, pages 46–60, 2012.

[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB J.*, 11(4):315–331, 2002.

[3] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[4] D. Angluin. Inference of reversible languages. *J. ACM*, 29(3):741–765, 1982.

[5] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *TWEB*, 4(4), 2010.

[6] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB*, pages 115–126, 2006.

[7] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Trans. Database Syst.*, 35(2), 2010.

[8] G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. In *WebDB*, pages 79–84, 2004.

[9] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *VLDB*, pages 998–1009, 2007.

[10] I. Boneva, R. Ciucanu, and S. Staworko. Simple schemas for unordered XML. In *WebDB*, 2013. Technical report at `http://arxiv.org/abs/1303.4277`.

[11] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Inf. Comput.*, 142(2):182–206, 1998.

[12] B. Chidlovskii. Schema extraction from XML: A grammatical inference approach. In *KRDB*, 2001.

[13] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

[14] D. Florescu. Managing semi-structured data. *ACM Queue*, 3(8):18–24, 2005.

[15] D. D. Freydenberger and T. Kötzing. Fast learning of restricted regular expressions and DTDs. In *ICDT*, pages 45–56, 2013.

[16] P. Garcia and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(9):920–925, 1990.

[17] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: Learning document type descriptors from XML document collections. *Data Min. Knowl. Discov.*, 7(1):23–56, 2003.

[18] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[19] S. Grijzenhout and M. Marx. The quality of the XML web. In *CIKM*, pages 1719–1724, 2011.

[20] J. Hegewald, F. Naumann, and M. Weis. XStruct: Efficient schema extraction from multiple and large XML documents. In *ICDE Workshops*, page 81, 2006.

[21] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory.* MIT Press, 1994.

[22] P. Kilpeläinen and R. Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Inf. Comput.*, 205(6):890–916, 2007.

[23] M. Lohrey, S. Maneth, and E. Noeth. XML compression via DAGs. In *ICDT*, pages 69–80, 2013.

[24] J.-K. Min, J.-Y. Ahn, and C.-W. Chung. Efficient extraction of schemas for XML documents. *Inf. Process. Lett.*, 85(1):7–12, 2003.

[25] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[26] S. Staworko and P. Wieczorek. Learning twig and path queries. In *ICDT*, pages 140–154, 2012.