Accepted Manuscript

ISSN 0022-0000
ELSEVIER
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
Available online at www.sciencedirect.com
ScienceDirect

Grammatical inference of directed acyclic graph languages with polynomial time complexity

Antonio-Javier Gallego, Damián López, Jorge Calera-Rubio

PII: S0022-0000(17)30286-6
DOI: https://doi.org/10.1016/j.jcss.2017.12.002
Reference: YJCSS 3151

To appear in: *Journal of Computer and System Sciences*

Received date: 15 July 2016
Revised date: 25 September 2017
Accepted date: 14 December 2017

Please cite this article in press as: A.-J. Gallego et al., Grammatical inference of directed acyclic graph languages with polynomial time complexity, *J. Comput. Syst. Sci.* (2018), https://doi.org/10.1016/j.jcss.2017.12.002

Highlights

- We tackle the task of graph language learning.
- We extend the classes of k-testability and k-TSS languages to directed graph languages.
- We propose a grammatical inference algorithm to learn this class of languages.
- The algorithm runs in polynomial time.
- The algorithm identifies this class of languages from positive data.

Grammatical inference of directed acyclic graph languages with polynomial time complexity

Antonio-Javier Gallego[a,*], Damián López[b], Jorge Calera-Rubio[a]

[a] *Pattern Recognition and Artificial Intelligence Group, Department of Software and Computing Systems, University of Alicante, 03690 Alicante, Spain*
[b] *Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 46022, Spain*

Abstract

In this paper we study the learning of graph languages. We extend the well-known classes of k-testability and k-testability in the strict sense languages to directed graph languages. We propose a grammatical inference algorithm to learn the class of directed acyclic k-testable in the strict sense graph languages. The algorithm runs in polynomial time and identifies this class of languages from positive data. We study its efficiency under several criteria, and perform a comprehensive experimentation with four datasets to show the validity of the method. Many fields, from pattern recognition to data compression, can take advantage of these results.

Keywords: Graph languages, graph automata, grammatical inference, k-testable languages

1. Introduction

Among the different approaches to machine learning, *inductive learning* can be roughly defined as the inductive search for a model that represents the supplied (training) data. Formal languages are suitable models under this approach. When formal languages are used, the inductive learning process is known as

[*] Corresponding author: Tel.: +349-65-903772; Fax: +349-65-909326
Email addresses: jgallego@dlsi.ua.es (Antonio-Javier Gallego), dlopez@dsic.upv.es (Damián López), calera@dlsi.ua.es (Jorge Calera-Rubio)

grammatical inference (GI). This approach to automatic learning is usually focused on the learning of regular languages or their subclasses. Regular string language inference is usually applied in many fields, from natural language processing [46] or script recognition [38] to bioinformatics [48, 42, 32, 33]. See [16] for a bibliographic study.

Even though regular language inference has been widely applied, it is important to note that in many contexts structural information is of great importance. This kind of information is not easy to model with subclasses of the regular language class. Nevertheless, context-free grammars can easily model this kind of information. In this line of work, Sakakibara presented the first algorithms for learning context-free languages with polynomial time complexity [40, 41]. Some other results for non-regular language inference study the inference of context-free languages, for instance, [43, 30, 7, 8], or the inference of context sensitive ones [44].

Looking also to enhance the possibilities of language inference, several works study the task of tree language inference [14, 6, 23], as well as its application to real tasks [31, 36, 46, 20]. In the grammatical inference framework, when more general graphs are considered, the main problem that arises is computational complexity, and, usually, graphs are reduced to less complex representations (usually some kind of graph traversal).

Our paper considers graphs as elements of some formal language. In this framework, and concerning general graphs, the handbook on graphs and graph transformations edited by Rozemberg [39], summarizes, among other results, the two main formalisms used to generate graph languages (node and hyperedge replacement grammars) as well as many theoretical results that relate graph grammars with logic. But, although the generating paradigm has been widely-studied and various graph automata models have been proposed [35, 5, 2], there does not exist a recognizing device that properly fits all the different characterized classes of graph languages.

Usually, the results concerning the inference of graph grammars are based on the problem of isomorphism on general (unrestricted) graphs, and therefore, have high time complexity [17, 18, 21]. Some works consider results on mining in graphs in order to propose graph grammar inference methods [3, 11]. There also exist works devoted to estimate the parameters of stochastic graph grammars [29] or the definition of new learnable classes of graph grammars [10].

In this paper, we first extend the notion of k-testability [27] to directed-graph languages. Thus, the classes of *k-testable* and *k-testable in the strict sense* (k-TSS) graph languages are defined. Briefly, k-testable and k-TSS language definitions take into account a finite set of structures of size k that are allowed to appear in the elements of the language. We also prove some lemmas that show that the main features of the class of k-TSS languages are still applicable to graph languages. This fact is important in order to be able to use this class in applied tasks, as has been done before with string and tree languages.

Second, we propose a polynomial grammatical inference algorithm to learn the class of k-TSS directed acyclic graph languages from positive data. Let us note that the k-testable structures our approach takes into account help to bound the above mentioned high complexity of graph isomorphism. Besides, the consideration of directed acyclic graphs also helps to further ease the general complexity. Finally, we study the time complexity of this algorithm and prove its polynomial behavior.

This work is structured as follows. In the next section some definitions and notation of graph languages and multisets are presented. In Section 3 we define the classes of k-testable and k-TSS graph languages. Some lemmas concerning the class of k-TSS graph languages are also proved. The model of graph automata used for directed acyclic graphs is defined in Section 4. Next, in Section 5, we propose the new inference algorithm and include a running example. We prove that it identifies the class of directed acyclic k-TSS graph languages from positive data. Section 6 examines the efficiency of this algorithm based on different criteria, and Section 7 shows the experimental results. We end the paper with the conclusions and some lines of future work.

2. Notation and Definitions

In the following, if not stated otherwise, we will consider node-labeled directed graphs, which we refer to as graphs, and which can be defined by a tuple $g = (V, E, \mu)$, where V is a finite set of nodes (or vertexes), $E \subseteq (V \times V) - id_V$ is the set of edges (where id_V denotes the smallest reflexive relation), and $\mu : V \to \Sigma$ is the node labeling function. Note that the definition does not allow a single node to be the two components of the same edge (i.e. loops are not allowed). When necessary, we will refer to the components of a graph g as V_g, E_g and μ_g. An acyclic graph is such that the reflexive-transitive closure of E is a partial order. Two graphs $g = (V, E, \mu)$ and $g' = (V', E', \mu')$ are *isomorphic* if there is a bijection $f : V \to V'$ such that, for any nodes $u, v \in V$, $\mu(v) = \mu'(f(v))$ and $(u, v) \in E$ if and only if $(f(u), f(v)) \in E'$.

For any given node v, an edge (u, v) is called *incoming* (resp. *outgoing* for edges of the form (v, u)). The *incoming degree* of a node v (resp. *outgoing degree*) is the number of incoming (resp. outgoing) edges of v will be denoted by $idg(v)$ and defined as $idg(v) = |\{(u, v) \in E\}|$ (resp. the outgoing degree is defined as $odg(v) = |\{(v, w) \in E\}|$). For any graph $g = (V, E, \mu)$, let $V^n_m(g)$ be defined as the set of nodes with incoming degree n and outgoing degree m. In the following, two sets of nodes will be of special interest: the set of nodes with zero incoming degree and the set of nodes with zero outgoing degree of a graph g, which we will denote respectively with $V^0(g)$ and $V_0(g)$.

We define a typed alphabet Σ_r as the association of an alphabet Σ with a finite relation $r \subseteq (\Sigma \times \mathbb{N} \times \mathbb{N})$. This relation allows to establish the allowed incoming and outgoing degrees for each label in the alphabet. Thus, the typed alphabet plays the same role as the plain alphabet in string languages or the ranked alphabet in tree languages. In order to explicitly refer to the subset of symbols of the ranked alphabet with a given input and output degree, we will denote with Σ^n_m the set: $\{s \in \Sigma \ : \ (s, n, m) \in r\}$. Note that the fact that r is a finite relation implies that Σ^n_m is non-empty for only a finite number of pairs of n and m.

Note that, once the typed alphabet is defined, the set of all possible consistently labeled graphs can be defined. Formally, let $\mathcal{G}(\Sigma_r)$ denote the set of graphs over Σ_r. A *graph language* is any set $L_G \subseteq \mathcal{G}(\Sigma_r)$.

Given a typed alphabet Σ_r, let the *extended alphabet* $\widehat{\Sigma}$ be the alphabet defined as the set:

$$\widehat{\Sigma} = \{a_m^n \ : \ a \in \Sigma, \ (a, n, m) \in r\}$$

Taking into account the extended alphabet, given a graph $g = (V, E, \mu)$, we define its *extended graph* as $\hat{g} = (V, E, \hat{\mu})$, where $\hat{\mu} : V \to \widehat{\Sigma}$, and such that for each node v in $V_m^n(g)$, if $\mu(v) = a$, then $\hat{\mu}(v) = a_m^n$. Intuitively, the use of the extended alphabet allows the labels of the nodes of the extended subgraph $\hat{g}$ to explicitly include the incoming and outgoing degrees. Figure 1 shows an example.

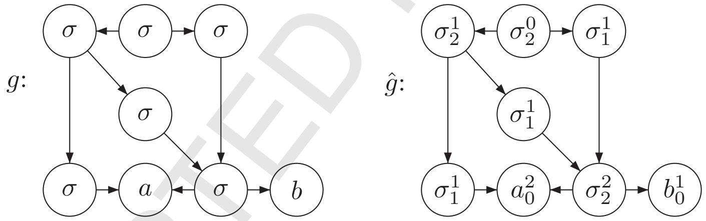


Figure 1: Example of a graph g and its corresponding extended graph $\hat{g}$.

In order to provide simple graph representations to illustrate our results, from now on we will consider skeletal graphs and skeletal graph languages (graphs where the nodes v such that $odg(v) \neq 0$ (internal nodes) are labeled with the same symbol). We will use Greek symbols to label internal nodes and Latin symbols to label frontier nodes (those with outgoing degree zero).

For any given sequence of nodes $w_1, w_2, \ldots, w_k$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq i < k$, we say that there exists a path from w_1 to w_k. We define the length of the path as the number of nodes in the sequence, including the starting and ending nodes of the path. A graph may have more than one path between a pair of nodes u and v. Thus, let $|u \mapsto v|$ denote the length of the (possibly multiple) shortest path. This is consistent with assigning infinite to the length of a non-existent path and $|u \mapsto u| = 1$. We define the *diameter* of a graph $g = (V, E, \mu)$ as the length of the shortest path between the two more distant connected nodes of the graph. More formally:

$$diameter(g) = \max_{u,v \in V} \{|u \mapsto v| \ : \ |u \mapsto v| < \infty\}$$

Given a graph $g = (V, E, \mu)$, the *subgraph of g rooted* in the node v with radius k is defined as $R_g(v, k) = (W, E', \mu')$ such that:

$$W = \{u \in V \ : \ |v \mapsto u| \leq k\}$$

and where $E' = E \cap (W \times W)$, that is, the set of edges restricted to the nodes in W. In the same way, μ' is the restriction of μ to the nodes in W. We extend this definition to consider, for any graph $g = (V, E, \mu)$, the *subgraph of g rooted* in the node v, denoted by $R_g(v) = (W, E', \mu')$ where:

$$W = \{u \in V \ : \ |v \mapsto u| < \infty\}$$

with the set E' and the labeling function μ' defined as above.

Multisets

We now recall some definitions from multiset theory [45] that will be used in the transition function of a new graph automata model and in the inference algorithm. For any given set D, a *multiset* over D is a pair $\langle D, f\rangle$ where $f : D \to \mathbb{N}$ is an enumeration function and $\mathbb{N}$ denotes the set of natural numbers. That is, for any $a \in D$, the function $f(a)$ denotes the number of elements a in the multiset. The size of a multiset $A = \langle D, f\rangle$ is denoted with $|A|$ and defined as $|A| = \sum_{a \in D} f(a)$.

Two multisets, $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$, are equal ($A = B$) if and only if, for all $a \in D$, $f(a) = g(a)$. In the same way, A is a *subset* of B ($A \subseteq B$) if and only if, for all $a \in D$, $f(a) \leq g(a)$. Let also the sum of two multisets $(A \oplus B)$ be defined as the multiset $C = \langle D, h \rangle$ where for all $a \in D$, $h(a) = f(a) + g(a)$.

A very useful concept for dealing with multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \rightarrow \mathbb{N}^n$ where $D = \{d_1, d_2, \ldots, d_n\}$ and D^* is the set of strings over D. For any $x \in D^*$, this mapping is defined as $\Psi(x) = (\#_{d_1}, \#_{d_2}, \ldots, \#_{d_n})$ where $\#_{d_i}$ denotes the number of occurrences of d_i in x. Note that this allows to represent a multiset using whichever string with the correct Parikh mapping. In the following, we will denote with $\mathcal{M}_n(D)$ the class of multisets whose size is equal to a constant n and with $\mathcal{M}(D)$ the class of multisets (of any size) over D.

3. *k*-testable graph languages

Testable and *testable in the strict sense* languages [27] are defined by a vector (I, S, F) which represents those structures that are allowed to appear in the members of the language. These families have been defined over string and tree languages [13, 14]. In this section, we extend the definition to consider graph languages.

Given a typed alphabet Σ_r and the corresponding set of graphs over it $\mathcal{G}(\Sigma_r)$, for any $g = (V, E, \mu) \in \mathcal{G}(\Sigma_r)$ and any $k \geq 2$, let us define the *k-testability vector* $T_k(g) = (I_{k-1}(g), P_k(g), F_{k-1}(g))$ where:

$$\begin{aligned}
I_{k-1}(g) &= \left\{R_{\hat{g}}(v, k-1) \ : \ v \in V^0(g)\right\} \\
P_k(g) &= \{R_{\hat{g}}(v, k) \ : \ v \in V, \ diameter(R_g(v)) \geq k\} \\
F_{k-1}(g) &= \{R_{\hat{g}}(v, k-1) \ : \ v \in V, \ diameter(R_g(v)) \leq k-1\}
\end{aligned}$$

Note that $P_k(g) = \emptyset$ if $diameter(g) < k$. Note also that the nodes of the graphs in each component of the k-testability vector are labeled with the extended function $\hat{\mu}$. Examples 1 and 2 show the k-testability vector of some graphs. Note that it is not necessary for the graph to be acyclic in order to obtain the k-testability vector. Example 2 illustrates this fact.

Example 1. *Given the graph in Figure 1, Figure 2 shows the components of the 3-testability vector. For each node, this extended labeling function depicts what the neighborhood was in the* mother *graph. For instance, note that the nodes labeled a_0^2 in Figure 2 do not always have two incoming edges. The extended labeling allows relevant structural information to be dealt with in a straightforward way. This labeling will play an important role in our grammatical inference algorithm.*

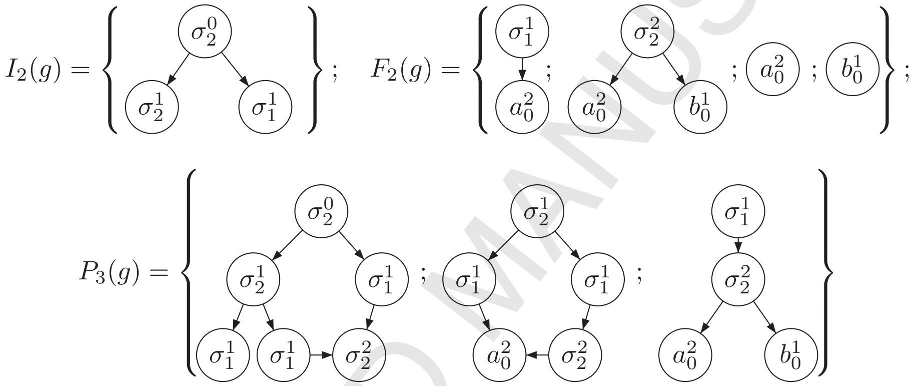


Figure 2: The components of the 3-testability vector for the graph in Figure 1 are shown.

Example 2. *Figure 3 shows a directed non-acyclic graph and its 2-testability vector. Note that, in this case, the set $V^0(g)$ is empty, and therefore the set $I_1(g)$ is empty as well.*

The functions I_k, F_k and P_k can be extended in a natural way to a set G of graphs:

$$I_k(G) = \bigcup_{g \in G} I_k(g);$$

$$P_k(G) = \bigcup_{g \in G} P_k(g);$$

$$F_k(G) = \bigcup_{g \in G} F_k(g)$$

For any pair of graphs g and g', it is possible to define an equivalence relation $\equiv_k$ over $\mathcal{G}(\Sigma_r)$ taking into account the k-testability vector, where $g \equiv_k g'$ if and

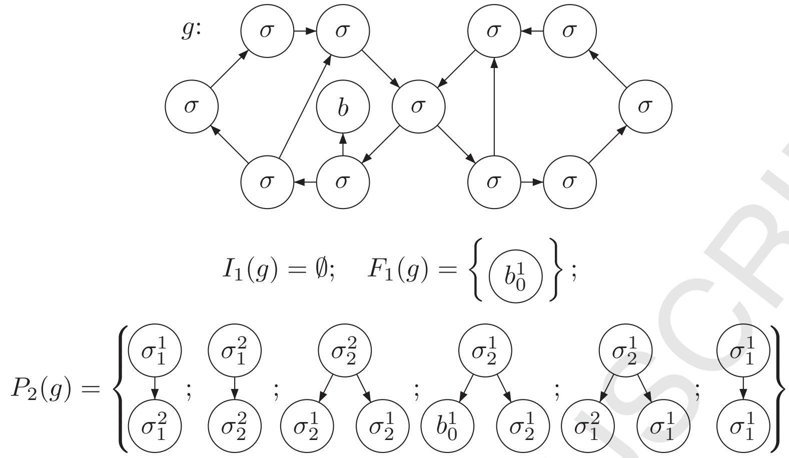


Figure 3: Directed graph and its corresponding 2-testability vector.

only if $T_k(g) = T_k(g')$. This equivalence relation is key in defining the classes of k-testable and k-testable in the strict sense graph languages.

Definition 1. *A graph language G is* k-testable *($k \geq 2$) if it results from the union of a finite number of equivalence classes of the relation $\equiv_k$.*

Intuitively, and in the same way it happens with string or tree languages, if g is a graph in a k-testable graph language G, then, every graph g' such that it has the same k-testability vector as g is also in G.

Definition 2. *For any $k \geq 2$, a graph language G is* k-testable in the strict sense *(k-TSS) if there exist three finite sets of graphs (B, S, E) such that, $g \in G$ if and only if $I_{k-1}(g) \subseteq B$, $P_k(g) \subseteq S$ and $F_{k-1}(g) \subseteq E$.*

According to the definition, and a shared feature of string and tree k-TSS languages, the membership to a k-TSS graph language of graphs with diameter smaller than k depends on $F_{k-1}(g)$. This is because for those graphs $P_k(g) = \emptyset$ and $I_{k-1}(g) \subseteq F_{k-1}(g)$.

We note here the importance of the graph isomorphism problem for directed acyclic graphs in the application of the results we present. Some papers in the literature state that the graph isomorphism problem has polynomial time complexity for some graph families that include the graphs we use in our results. Among those papers, in [4] and [24] graphs with bounded tree-width and bounded degree are studied, and in [28] the author studies the class of graphs with bounded genus. In all these cases, it is proved that isomorphism of such graphs can be determined with polynomial time complexity.

Note that, for a given set of graphs G, the k-testability vector defines a k-TSS language when the sets of graphs B, S and E are set to $I_{k-1}(G)$, $P_k(G)$ and $F_{k-1}(G)$, respectively. Let this language (obtained from the set of graphs G) be denoted by $L_k(G)$. Note that, as well as for the case of string and tree languages, given a k value and a typed alphabet, the class of k-TSS graph languages is finite. We now prove some results related to this class of languages.

Lemma 1. *Let G be a finite set of graphs and $k \geq 2$, then $G \subsetneq L_k(G)$.*

Proof. Let $g \in G$, trivially $I_{k-1}(g) \subseteq I_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $F_{k-1}(g) \subseteq F_{k-1}(G)$. The language $L_k(G)$ is defined by the vector $T_k(G) = (I_{k-1}(G), P_k(G), F_{k-1}(G))$; therefore, $g \in L_k(G)$.

To prove that the inclusion is strict, note that any disconnected graph obtained by joining graphs in the set G will belong to the k-TSS language $L_k(G)$. This implies that every k-TSS graph language, except the empty one, is infinite. □

In string [13] and tree languages [14], it is proved that, when the value of k is greater than the maximum length (depth in the case of trees) of the elements in a set S, then the k-TSS language obtained from that set (of strings or trees) equals S. A consequence of Lemma 1 is that this fact does not hold when graph languages are taken into account because the minimum k-TSS of a set of graphs always generalize the set G.

We note that, in some cases, it is possible to build new graphs taking into account the graphs rooted in the set of nodes with zero incoming degree. As an example, let us consider a set of graphs containing only the graph g shown in Figure 4 (with $diameter(g) = 3$) and its corresponding 4-testable vector. Note that the graph g' shown in the same figure belongs to the language $L_4(\{g\})$.

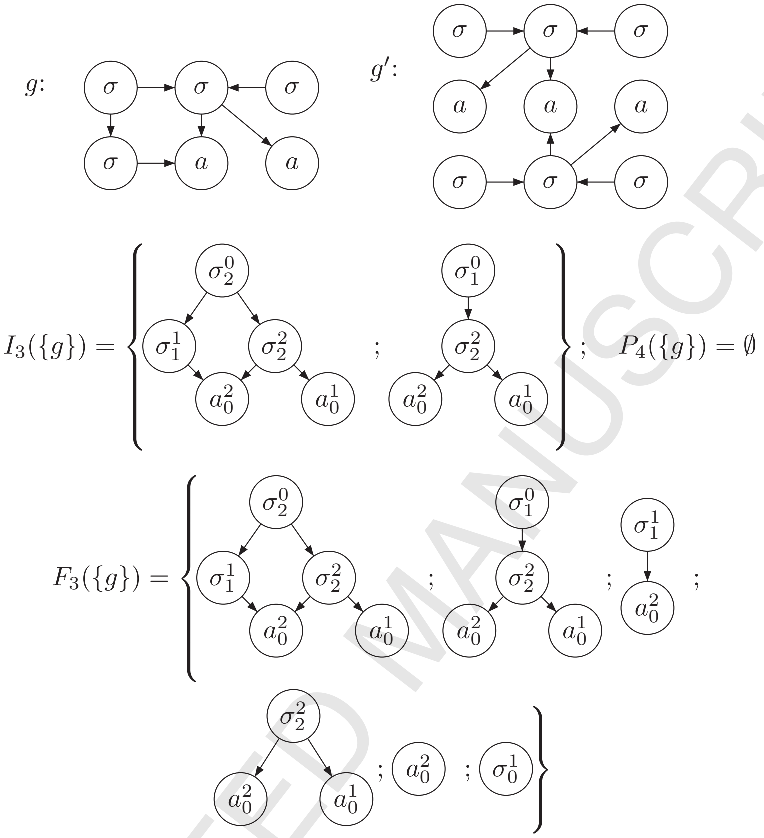


Figure 4: The components of the 4-testability vector for $L_4(\{g\})$ are shown. Note that the graph g' belongs to the language $L_4(\{g\})$.

Lemma 2. *For any set of graphs G and a given $k \geq 2$, the language $L_k(G)$ is the smallest k-TSS language that contains G.*

Proof. We will prove that, for any given k-TSS language T, if $G \subset T$, then $T \not\subset L_k(G)$.

We first note that if $G \subset T$, then all the structures into the k-testability vector $T_k(G)$ are also into the vector that characterizes the language T, and therefore T and $L(G)$ cannot be incomparable.

Let (B_T, S_T, E_T) be the sets that define the language T. Let us suppose that $T \subset L_k(G)$; then there is a graph $g \in L_k(G) - T$. On the one hand, $g \in L_k(G)$, then $I_{k-1}(g) \subseteq I_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $F_{k-1}(g) \subseteq F_{k-1}(G)$. On the other hand, $g \notin T$, therefore, $I_{k-1}(g) \nsubseteq B_T$ or $P_k(g) \nsubseteq S_T$ or $F_{k-1}(g) \nsubseteq E_T$.

In other words, there are some structures in the k-testability vector of $L_k(G)$ that are not present in the one of T. From this, it follows that there exists a graph g' such that $g' \in G$ and $g' \notin T$; therefore, $G \nsubseteq T$, which contradicts the previous assumption. □

Lemma 3. *Let G and G' be two sets of graphs and $k \geq 2$. If $G \subseteq G'$, then $L_k(G) \subseteq L_k(G')$.*

Proof. It is easy to see that, if $G \subseteq G'$, then $I_{k-1}(G) \subseteq I_{k-1}(G')$, $P_k(G) \subseteq P_k(G')$ and $F_{k-1}(G) \subseteq F_{k-1}(G')$. Therefore, $L_k(G) \subseteq L_k(G')$. □

Lemma 4. *For any set of graphs G and $k \geq 2$, $L_{k+1}(G) \subseteq L_k(G)$.*

Proof. We need to prove that, for every $g \in L_{k+1}(G)$, g is also in $L_k(G)$, in other words, we need to prove that for any $g \in L_{k+1}(G)$, $I_{k-1}(g) \subseteq I_{k-1}(G)$, $F_{k-1}(g) \subseteq F_{k-1}(G)$ and $P_k(g) \subseteq P_k(G)$ hold.

Note that the sets $I_{k-1}(G)$ and $F_{k-1}(G)$ can be obtained from the sets $I_k(G)$ and $F_k(G)$ as follows:

$$I_{k-1}(G) = I_{k-1}(I_k(G))$$
$$F_{k-1}(G) = F_{k-1}(F_k(G))$$

Concerning $P_k(G)$ and $P_{k+1}(G)$, for every graph $g \in L_{k+1}(G)$, we distinguish two cases:

- if $diameter(g) \leq k$, then $P_{k+1}(g) = \emptyset$, which is a subset of $P_k(G)$
- if $diameter(g) > k$, then $P_k(g) = P_k(P_{k+1}(g)) \subseteq P_k(P_{k+1}(G)) = P_k(G)$

Thus, as mentioned above, any graph fulfilling the conditions fixed by the $(k+1)$-testability vector also fulfills those fixed by the k-testability vector. Therefore, we conclude that $L_{k+1}(G) \subseteq L_k(G)$ □

We remark here the important role the incoming and outgoing degrees have in order to obtain different graphs with the same diameter over a given typed alphabet. We also remark that, for any given typed alphabet and k values, the number of distinct graphs that can appear in a k-testability vector depends on the symbols that are in the typed alphabet.

We recall that every k-TSS graph language (and every k-testable graph language as well) is related to a k-testability vector. Taking into account the (finite) number of different elements that can appear in a k-testability vector, it follows that the class of k-TSS graph languages is finite, although the number of languages in the class can be dramatically high (in fact, exponentially high, because every subset of such elements potentially defines a different k-TSS language).

This should neither be considered as a drawback nor as an advantage. It is a shared feature with k-TSS string and tree languages, for which there exist results that support their interest in many fields.

In the next section, we propose a new model of graph automata for directed acyclic graphs and use this model to propose a grammatical inference algorithm.

4. Graph automata model

In [22] a non-deterministic automata model for directed and acyclic graph languages is presented. That model takes into account the work by Potthoff et al. in [35], where the authors present three models of finite graph automata. All three models consider node-labeled graphs together with a coloring over the edges. The authors restrict this class of graphs to those graphs where, on the incoming (resp. outgoing) edges of a node, each color appears at most once. Thus, as the authors state, they are dealing with some kind of *ordered* acyclic graph class. The order induced by the edge coloring helps in the processing of the transitions.

The main difference of the automata model in [22] with respect to the previous models commented above consist in the definition of the transition function of the automata, which takes into account a multiset which permits the graphs to be processed without taking into account any order among the nodes except for the partial one induced by the directed edges. Another difference is the consideration of a set of *final states*. Previous automata models consider the existence of a *run* over the graph as the criterion to accept it. In [22], the authors present a model that traverses the graph with a bottom-up scheme (in a similar way a bottom-up tree automaton analyzes a tree), but taking into account that there does not exist any order among the siblings. A state is assigned to each node v taking into account a multiset of states and the extended label of the node. The multiset of states is the result of processing the nodes reached by the outgoing edges from v. The use of the extended labels allow graph-like structures to be considered in a straightforward way. Once the analysis ends, a state has been assigned to each node with zero incoming degree. The graph belongs to the language accepted by the automaton if all the states assigned to the nodes with zero incoming degree are final.

In this paper we consider a deterministic version of the model in [22], i.e. the automaton has (at most) only one transition for any symbol in $\widehat{\Sigma}$ and any multiset over the set of states. For the sake of simplicity, multisets will be represented in the following using whichever string with the correct Parikh mapping. Definition 3 describes the automata model.

Definition 3. *Let Σ_r be a typed alphabet where w denotes the maximum outgoing degree in $\widehat{\Sigma}$. A (non-deterministic) graph automaton for a language over Σ_r is defined as the tuple $GA = (Q, \widehat{\Sigma}, \delta, F)$, where Q is a finite set of states, $\widehat{\Sigma}$ is the extended alphabet of Σ_r, the set $F \subseteq Q$ contains the final states and δ is a set of transition functions defined as follows:*

$$\delta = \bigcup_{\substack{0 \leq j \leq w \\ j:\ \exists n > 0, \Sigma_j^n \neq \emptyset}} \delta_j$$

where each δ_j is defined as:

$$\delta_j : \widehat{\Sigma}_j^n \times \mathcal{M}_j(Q) \rightarrow Q, \qquad 0 \leq j \leq w$$

where, as defined above, $\mathcal{M}_j$ represents the class of multisets of size j.

Please, note that the definition of the domain of the transition function considers the extended alphabet instead of the original one. This allows the graph to be processed taking into account both the outgoing degree (the size of the multiset) and the incoming degree of the node (captured in the symbol of the extended alphabet).

In order to extend the transition function to operate on graphs, the intuitive idea consists of a recursive analysis over each zero-incoming degree node. The multisets returned by these analyses are then summed ($\oplus$ operator). Formally, for any given graph g the function δ is extended as follows:

$$\delta : \mathcal{G}(\Sigma) \cup \mathcal{G}(\widehat{\Sigma}) \to \mathcal{M}(Q)$$
$$\delta(g) = \delta(\hat{g}) = \bigoplus_{v_i \in V^0(\hat{g})} \delta(R_{\hat{g}}(v_i))$$

where, assuming that the outgoing degree of node v_i is m:

$$\delta(R_g(v_i)) = \delta_m(\mu_{\hat{g}}(v_i), M_{i1} \oplus \ldots \oplus M_{im}) :$$
$$M_{ij} = \delta(R_g(w_j)),\ (v_i, w_j) \in E$$

For any graph g the extended version $\hat{g}$ is isomorphic, thus, there is no problem in reducing the parsing of a graph to its extended version. The language accepted by the automaton is defined as follows:

$$L(GA) = \{g \in \mathcal{G}(\Sigma_r)\ :\ \forall q \in \delta(\hat{g}),\ q \in F\}$$

Thus, any graph g is accepted by the automaton GA if and only if the extended transition function returns a multiset that contains only final states. When necessary, we will refer to these multisets as final multisets.

The model of graph automata we propose is not able to process general directed graph languages, that is, languages with graphs that may contain cycles. The main problems to extend this model to those general graphs is the need to establish a processing order, since it is possible for the graph to have no node with zero incoming degree. Another, closely related problem is to establish an acceptance criterion for general graphs because, again, the graph may have no node with zero outgoing degree.

$$\begin{aligned}
\delta(a_0^2,\lambda) &= q_1\\
\delta(b_0^1,\lambda) &= q_2\\
\delta(\sigma_2^2,q_1q_2) &= q_1\\
\delta(\sigma_1^1,q_1) &= q_2\\
\delta(\sigma_2^1,q_2q_2) &= q_1\\
\delta(\sigma_2^0,q_1q_2) &= q_1, \text{ where } q_1\in F
\end{aligned}$$

Figure 5: An example of graph automaton.

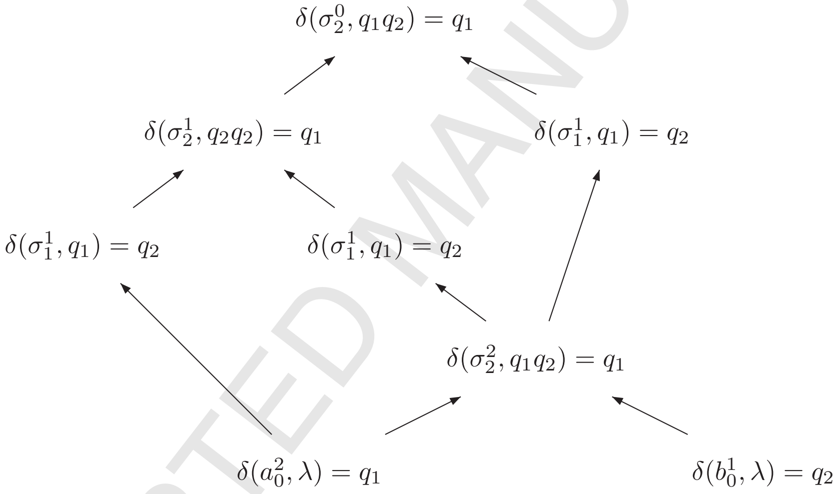

$\delta(\sigma_2^0, q_1q_2) = q_1$
$\delta(\sigma_2^1, q_2q_2) = q_1$
$\delta(\sigma_1^1, q_1) = q_2$
$\delta(\sigma_1^1, q_1) = q_2$
$\delta(\sigma_1^1, q_1) = q_2$
$\delta(\sigma_2^2, q_1q_2) = q_1$
$\delta(a_0^2, \lambda) = q_1$
$\delta(b_0^1, \lambda) = q_2$

Figure 6: Example of the parsing of the graph in Figure 1.

Example 3. *Taking into account the graph shown in Figure 1 and the automaton in Figure 5, a representation of the analysis of the graph is depicted in Figure 6.*

We recall that the strings in the transition function denote multisets according the Parikh function. Thus, the string q_1q_2 represents the multiset with one element q_1 and one element q_2. The graph is recursively traversed to reach those nodes with zero outgoing degree, therefore, the first nodes that are related to a state are those for which no recursive call is needed (nodes with zero outgoing degree). Figure 6 shows the order of the parsing process once those nodes are *reached.*

As we mentioned above, the processing of a graph reminds the processing of a tree in a bottom-up tree automaton. Please note that, in this case, the use of multisets permits not to take into account the order of the siblings.

5. *k*-TSS inference algorithm

We now propose Algorithm 5.1 to infer the class of k-TSS graph languages from positive presentation. The algorithm follows the same scheme used previously to infer k-TSS string or tree languages [13, 14]. The algorithm first establishes the set of states taking into account the graph structures of diameter $k-1$ in the k-testability vector of the input sample. The set of final states is also established. Then, the algorithm creates the transitions using the graphs in $F_{k-1}(G)$ and $P_k(G)$. Let us recall here that the definition of the transition function considers a symbol of the extended alphabet and a multiset of states, and, for the sake of simplicity, multisets are represented by a string (of states) with the correct Parikh mapping (lines 8 and 11 in Algorithm 5.1). An example of run is given below.

Example 4. *Let us consider $k = 2$, and the set G of graphs shown in Figure 7. The extracted elements of the 2-testability vector are shown in Figure 8.*

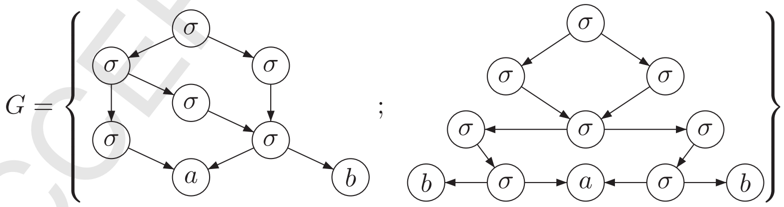


Figure 7: Set of graphs example.

Algorithm 5.1 Grammatical inference algorithm from positive sample for the class of k-TSS graph languages.

```
INPUT: A set $G$ of graphs. A value $k \geq 2$
OUTPUT: A graph automaton that recognizes the language $L_k(G)$
Method:
Compute $(I_{k-1}(G), P_k(G), F_{k-1}(G))$
Let $\Sigma_r$ be the typed alphabet from $G$ and $\widehat{\Sigma}_r$ be the extended one
for $g \in \{I_{k-1}(G) \cup F_{k-1}(G) \cup I_{k-1}(P_k(G))\}$ do
    Let $Q[g]$ be a new state related to $g$
end for
$F = \{Q[g] \ : \ g \in I_{k-1}(G)\}$
for all $g \in F_{k-1}(G)$, $v \in V_m^0(g)$, where $(v, w_i) \in E_g$, $1 \leq i \leq m$ do
    $\delta_m(\mu(v), Q[R_g(w_1)] \ldots Q[R_g(w_m)]) = Q[g]$
end for
for all $g \in P_k(G)$, $v \in V_m^0(g)$, where $(v, w_i) \in E_g$, $1 \leq i \leq m$ do
    $\delta_m(\mu(v), Q[R_g(w_1, k-1)] \ldots Q[R_g(w_m, k-1)]) = Q[R_g(v, k-1)]$
end for
return $(Q, \widehat{\Sigma}_r, F, \delta)$
EndMethod:
```

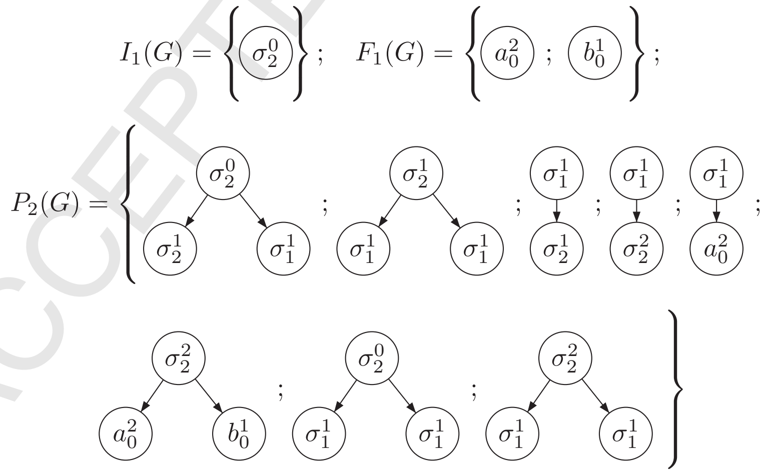


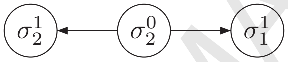
Figure 8: Elements of the 2-testability vector for the graphs example.

First, the algorithm constructs the set of states taking into account $I_1(G)$, $F_1(G)$ and $I_1(P_2(G))$:

$$Q\left[\enclose{circle}{a_0^2}\right] = q_1; \quad Q\left[\enclose{circle}{b_0^1}\right] = q_2; \quad Q\left[\enclose{circle}{\sigma_2^2}\right] = q_3;$$

$$Q\left[\enclose{circle}{\sigma_1^1}\right] = q_4; \quad Q\left[\enclose{circle}{\sigma_2^1}\right] = q_5; \quad Q\left[\enclose{circle}{\sigma_2^0}\right] = q_6$$

The algorithm obtains the set of final states, which is $F = \{q_6\}$. Then, the algorithm considers the graphs in $F_{k-1}(G)$. Note that the diameter of the graphs is 1. Therefore, the transitions $\delta(a_0^2, \lambda) = q_1$ and $\delta(b_0^1, \lambda) = q_2$ are added to the automaton. Note that λ denotes the empty string. The algorithm now processes the graphs in $P_k(G)$. As an example, let us consider the following graph in $P_2(G)$:



The algorithm takes into account the subgraphs of diameter $k-1$ rooted at the nodes below *the node σ_2^0 and the graph of diameter $k-1$ rooted at the node σ_2^0. Thus, the algorithm adds the transition $\delta(\sigma_2^0, q_5q_4) = q_6$.*

Once all the structures in the k-testability vector have been processed, the following automaton is obtained:

$$\begin{aligned}
\delta(a_0^2, \lambda) &= q_1 \\
\delta(b_0^1, \lambda) &= q_2 \\
\delta(\sigma_2^0, q_5q_4) &= q_6, \text{ where } q_6 \in F \\
\delta(\sigma_2^1, q_4q_4) &= q_5 \\
\delta(\sigma_1^1, q_3) &= q_4 \\
\delta(\sigma_1^1, q_5) &= q_4 \\
\delta(\sigma_1^1, q_1) &= q_4 \\
\delta(\sigma_2^2, q_1q_2) &= q_3 \\
\delta(\sigma_2^0, q_4q_4) &= q_6, \text{ where } q_6 \in F \\
\delta(\sigma_2^2, q_4q_4) &= q_3
\end{aligned}$$

As an example, Figure 9 shows two graphs that were not in the input set and that belong to the 2-TSS graph language.

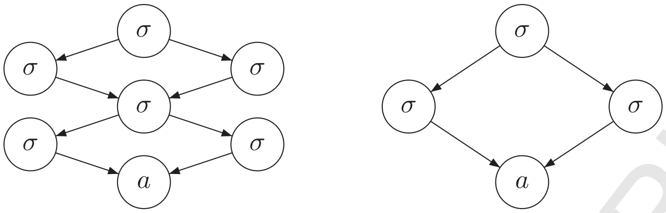


Figure 9: Two graphs not provided in the input set that belong to the example 2-TSS graph language.

We now prove that the algorithm identifies in the limit the class of k-TSS directed acyclic graph languages from positive presentation.

Theorem 1. *Algorithm 5.1 identifies the class of k-TSS graph languages from positive sample.*

Proof. We first prove that, given a set of graphs G, the algorithm returns a graph automaton GA that accepts the language $L_k(G)$. Note that, on the one hand, for any graph g, its membership to the language of the automaton output by Algorithm 5.1 implies to analyze each of the p nodes with zero incoming degree. In order to accept the graph g, all these analyses should return a final multiset. On the other hand, the membership of any graph g to $L_k(G)$ implies that, among other criteria, for every node v in $V^0(g)$, $R_{\hat{g}}(v, k-1) \in I_{k-1}(G)$. Thus, for the sake of clarity, and without loss of generality, we will consider graphs with just one node with zero incoming degree.

$L_k(G) \subseteq L(GA)$: We will prove by induction on the diameter of the graphs, that, if $g \in L_k(G)$, then $\delta(g)$ returns a final state. First, if $diameter(g) < k$ and $v \in V^0(g)$, then g is isomorphic to $\hat{g} = R_{\hat{g}}(v) \in I_{k-1} \cap F_{k-1}$, and the algorithm sets $\delta(R_{\hat{g}}(v)) = Q[R_{\hat{g}}(v, k-1)] = Q[R_{\hat{g}}(v)]$, which is a final state. Let us suppose that, for any graph g such that $diameter(g) = n$, it is fulfilled that $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$ where v is in $V^0(g)$. Note that $Q[R_{\hat{g}}(v, k-1)]$ is a final state because of the definition of I_{k-1} and the construction of the set of final states (line 6 in Algorithm 5.1). Now let g be a graph with $v \in V^0_m(g)$, such that, for $1 \leq i \leq m$, there exists $(v, w_i) \in E$ and $diameter(R_g(w_i)) \leq n$, and where at least one of the graphs $R_g(w_i)$ has diameter n. Then, $\delta(R_{\hat{g}}(w_i)) = Q[R_{\hat{g}}(w_i, k-1)]$ for each i, and therefore:

$$\begin{aligned}\delta(g) = \delta(\hat{g}) =& \delta_m(\mu_{\hat{g}}(v), \delta(R_{\hat{g}}(w_1)) \oplus \delta(R_{\hat{g}}(w_2)) \oplus \ldots \oplus \delta(R_{\hat{g}}(w_m))) \\ =& \delta_m(\mu_{\hat{g}}(v), Q[R_{\hat{g}}(w_1, k-1)]Q[R_{\hat{g}}(w_2, k-1)] \ldots \\ & \ldots Q[R_{\hat{g}}(w_m, k-1)])\end{aligned}$$

Note that there is an edge in g from v to each w_i. Thus, the resulting joint graph with all the $R_{\hat{g}}(w_i, k-1)$ is such that $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$. Note also that the state is final, because $R_{\hat{g}}(v, k-1)$ is in $I_{k-1}(g)$.

$L(GA) \subseteq L_k(G)$: We will prove that, for any graph $g \in L(GA)$, it is fulfilled that $F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and there is graph $q \in I_{k-1}(g)$ such that $\delta(g) = Q[q]$ (a final state). We will prove the result by induction on the diameter of the graph.

First, if $\mathit{diameter}(g) < k$ with $v \in V^0(g)$, then $\hat{g} \in F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) = \emptyset$ and $R_{\hat{g}}(v, k-1)$ is in $I_{k-1}(G)$.

Let us suppose by induction hypothesis that, for any graph $g \in L(GA)$ such that $\mathit{diameter}(g) = n \geq k$, it is fulfilled that $F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$, where $v \in V^0(g)$.

Now let g be a graph such that $v \in V^0_m(g)$, with $(v, w_i) \in E$ and where $\mathit{diameter}(R_g(w_i)) \leq n$ for all $1 \leq i \leq m$, with at least one of the graphs $R_g(w_i)$ of diameter n. Therefore:

$$\begin{aligned}\delta(g) = \delta(\hat{g}) =& \delta_m(\mu_{\hat{g}}(v), \delta(R_{\hat{g}}(w_1)) \oplus \delta(R_{\hat{g}}(w_2)) \oplus \ldots \oplus \delta(R_{\hat{g}}(w_m))) = \\ =& \delta_m(\mu_{\hat{g}}(v), Q[R_{\hat{g}}(w_1, k-1)]Q[R_{\hat{g}}(w_2, k-1)] \ldots \\ & \ldots Q[R_{\hat{g}}(w_m, k-1)]) = \\ =& Q[R_{\hat{g}}(v, k-1)]\end{aligned}$$

where $R_{\hat{g}}(v, k) \in P_k(g)$ because $(v, w_i) \in E$ for all $1 \leq i \leq m$. Besides,

$Q[R_{\hat{g}}(v, k-1)] = q$. Moreover:

$$F_{k-1}(g) = \bigcup_{1 \leq i \leq m} F_{k-1}(R_{\hat{g}}(w_i)) \subseteq F_{k-1}(G)$$

$$P_k(g) = \left(\{R_{\hat{g}}(v, k)\} \cup \bigcup_{1 \leq i \leq m} P_k(R_{\hat{g}}(w_i)) \right) \subseteq P_k(G).$$

Also, if $g \in L(GA)$, then $Q[R_{\hat{g}}(v, k-1)]$ is a final state. Therefore $R_{\hat{g}}(v, k-1)$ is in $I_{k-1}(G)$ and $g \in L_k(G)$.

Given the fact that, for any k given, the elements in the components of the k-testable vector are finite, we conclude that the proposed algorithm identifies the class of k-TSS directed acyclic graph languages. □

6. Efficiency of the algorithm

Gold's seminal learnability success criterion states that a class of languages is *identifiable in the limit* with respect to a learning algorithm if, for any language in the class there is a training set S such that, when supplied to the algorithm, it outputs a correct hypothesis that does not change when the algorithm considers supersets of S [15].

As we remark at the end of Section 3, each k-TSS graph language is related to a k-testability vector, and, therefore, for every language in the class, there always exists a set of graphs that can be processed to obtain the k-testability vector. From this fact, it follows that the class of k-TSS graph languages is identifiable in the limit.

From the seminal Gold's success criterion, many time efficiency conditions have been proposed in the literature. It is worth to be noted here that efficiency criteria usually consider an incremental framework, that is, they assume that the inference algorithms iterate on the input set, and modify a hypothesis to adapt it to the new sample. We note that Algorithm 5.1 processes the input set of graphs and outputs the graph automaton that accepts the smallest k-TSS graph language that contains the input set. Nevertheless, it seems rather natural to think of an incremental version of this algorithm that forgets the input set of graphs and keeps the current hypothesis. In the following, we will consider such an incremental version of the algorithm in order to study the behavior of the algorithm in this (incremental) framework.

Angluin and Smith [1], among some other criteria, define the *mind changes errors* for evaluating inference algorithms. Given x as an admissible presentation and M an identification method, let $DH(M, x)$ be the number of *distinct hypotheses* output by M, and $MC(M, x)$ be the number of *mind changes*, that is, times when the hypothesis that is output changes.

Yokomori [49] also says that, in order to be useful in applied tasks, a given inference method should run using positive presentation. Thus, motivated by a question posed by Angluin and recalling a previous definition by Pitt [34], Yokomori proposes a definition of polynomial time identification in the limit from text as follows:

Definition 4. *(Yokomori [49]) A class of grammars (languages) $\mathcal{L}$ is polynomial-time identifiable in the limit from text if and only if there is an algorithm M which, given $l \in \mathcal{L}$, identifies $l' \in \mathcal{L}$ equivalent to l in the limit from positive presentation, with the property that there exist polynomials p and q such that for any n, for any l of size n, the number of times M makes a wrong conjecture is at most $p(n)$, and the time for updating a conjecture is at most $q(n, N)$ where N is the sum of lengths of data provided.*

We note that, when a new sample is processed, our algorithm modifies the automaton only when the current sample is not accepted. Therefore, we can identify a wrong conjecture and a *mind change*. We also note that, in the worst case, each input sample modifies just one transition rule in the automaton, therefore, it can be concluded that the number of mind changes of Algorithm 5.1 is bounded by the size of the automaton that represents the target language.

In order to prove that the proposed algorithm is efficient *in the sense of Yokomori*, we prove that the time for updating a conjecture is polynomial.

Lemma 5. *The time our algorithm takes to updating a conjecture is polynomial.*

Proof. Let G denote the input set of graphs. Also, let w denote the greatest outgoing degree of the graph nodes in G. Finally, let Σ_r be the typed alphabet of the graphs in G.

For a given value of k, the time complexity to obtain each transition is bounded by $\mathcal{O}(w \cdot |\widehat{\Sigma}_r| \cdot w^{k-1})$ (that is, the biggest outgoing degree times the size of the alphabet times the size of the greatest subgraph that can be reduced to a state).

Each inference step implies at most the creation of as many transitions as the number of nodes of the sample graph. Let n denote that number. Thus, the time for updating a conjecture is bounded by $\mathcal{O}(n \cdot |\widehat{\Sigma}_r| \cdot w^k)$. □

Thus, we have the following:

Theorem 2. *The proposed algorithm identifies in the limit, in polynomial time from text the class of k-TSS graph languages.*

7. Experimental results

In this section we show some experimental results to illustrate the suitability of the k-TSS graph language class to model data. To do so, we have tested the proposed inference algorithm with four datasets: the first one is synthetic generated according to the properties of RNA hairpin loops; the second dataset is a collection of architectonic and electronic symbols; the third dataset contains information related to chemical compounds; and the fourth is related to handwritten symbols. We have used six different algorithms in order to model these data using graphs.

In this section, we first describe the datasets and parameters used to assess the behavior of the learning algorithm. Then we perform an experiment to study the inter-class discrimination capabilities (ie., each class from the rest) of the k-TSS automaton. The good results obtained show that the learning of

k-TSS graph languages is useful to efficiently distinguish between the different classes in each dataset.

7.1. Datasets

7.1.1. Hairpin RNA molecules dataset

The function of nucleic acid molecules is usually determined by their secondary structure. This secondary structure is due to bonds between some elements in an RNA strand (nucleotides) with other elements that are not adjacent in the strand. The creation of these bonds is possible because the RNA bases (adenine, guanine, uracil, and cytosine) are complementary two by two (adenine is complementary to uracil as well as cytosine to guanine). The simplest secondary structures that are present in RNA molecules are called hairpin loops or hairpins.

A hairpin loop is an unpaired loop of (messenger) RNA that is created when an RNA strand folds because some nucleotide bases of the strand form pairs. Hairpins are a common type of secondary structure in RNA molecules and are of importance in the cell processes (from the interaction of the RNA molecule with a ribosome or the protection of the RNA molecule from degradation, to its role in enzymatic reactions). In Figure 10 we show a graphical representation of a (synthetic) RNA molecule with two hairpin loops.

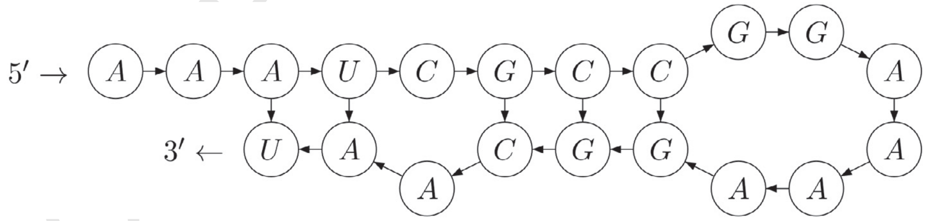


Figure 10: Example of RNA hairpin graph representation. The RNA sequence AAAUCGC-CGGAAAAGGCAAU with two hairpin loops is shown. The $5' \rightarrow 3'$ order has been used to orient the edges of the graph.

A dataset of 10,000 samples of hairpin graphs was generated synthetically following a series of simple rules modeling the formation of hairpin RNA molecules and taking into account the following criteria:

- The length of the loops was randomly chosen taking into account a normal distribution with a mean of 6 bases and a standard deviation of 1.5 bases.
- The length of the RNA molecules was randomly chosen with a minimum length of 10 and a maximum of 60.
- The position of the hairpin was also randomly generated. Non-hairpin sequences were allowed (they occur with a probability of 0.2%).

Biologically, the molecular structure of the RNA provides an order on the sequence (from the so-called 5′ end to the 3′ end). We take this ordering into account to obtain directed acyclic graph representations of hairpin RNA molecules. We note that, when this biological feature of the RNA strands is considered the graphs obtained are acyclic.

It is worth noting that some of the biological structures (for instance, pseudoknots in RNA molecules) are beyond the class of context-free languages [25]. Nevertheless, these structures can be easily modeled using directed acyclic graphs.

7.1.2. IAM graph datasets

The IAM-Graph DB is a publicly accessible repository freely available for non-commercial research purposes [37]. From the set of databases available in this repository we have chosen two that are appropriate for the task at hand: GREC and Mutagenicity datasets. The rest were discarded well for being synthetic or similar to those already tested.

GREC dataset. The GREC dataset consists of graphs representing symbols from architectural and electronic drawings [9]. The images occur at five different distortion levels. Figure 11 shows some examples of drawings for each distortion level. Depending on the distortion level, either erosion, dilation, or other morphological operations are applied. The result is thinned to obtain lines of one pixel width. Finally, graphs are extracted from the resulting denoised images by tracing the lines from end to end and detecting intersections as well as corners. A depth-first traversal of the graphs were used to orientate the edges of the representation. The selection of the initial node (node with zero incoming degree) was made randomly.

Nodes are labeled representing three possible classes: intersections, corners and ending points. This dataset contains 1,110 graphs uniformly distributed in 22 classes.

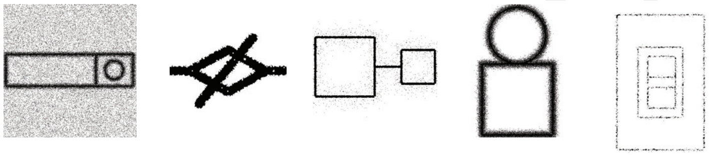


Figure 11: Examples of drawings from the GREC dataset for each of the distortion levels.

Mutagenicity dataset. Mutagenicity is a dataset of molecular structures with several chemical compounds classified as mutagenic or not mutagenic [19]. The term "mutagenicity" refers to one of the numerous adverse properties of a compound that hampers its potential to become a marketable drug. Toxic properties can often be related to chemical structure, more specifically, to particular substructures, which are generally identified as toxicophores. Figure 12 shows some examples of different compounds classified as mutagenic and non-mutagenic. In this dataset, molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. As for the GREC dataset, the orientation of the edges was set using the depth-first traversal of the graphs, and the initial node was randomly chosen.

Nodes are labeled with the number of the corresponding chemical symbol, which can be one of the following fourteen classes: Br, C, Ca, Cl, F, H, I, K, Li, N, Na, O, P, and S. This dataset contains 4,337 samples divided into two classes: 2,401 mutagen elements and 1,936 non-mutagen elements.

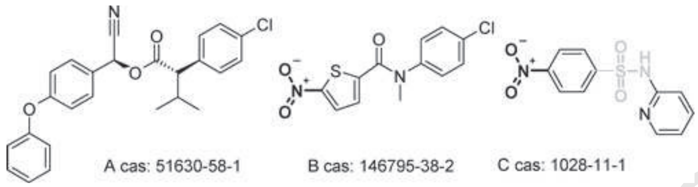


Figure 12: Examples of chemical compounds classified as mutagenic (compound B) and non-mutagenic (compounds A and C).

7.1.3. NIST dataset

The *NIST SPECIAL DATABASE* (NIST) [47] consists of a huge dataset of images of isolated handwritten characters, including upper and lower case letters and digits. For the extraction of graphs we have considered a subset of randomly chosen samples from the digits' section of the dataset. Three of the algorithms proposed in [12], namely: neighborhood, grid, and skeleton graphs were used to model the images. These methods are a natural generalization of similar tree-representation methods and have shown good performance in graph-representation of images.

Briefly speaking, the neighborhood method considers the quadtree representation of the image and calculates the neighborhood of regions up to a certain level of depth 'd' using the obtained q-tree to generate the graphs. The grid algorithm reduces the image to a structural grid of a certain size 'g' and creates the graph by assigning as nodes the regions with the foreground color and as edges the adjacency between the foreground regions. Only four plane directions (north, east, south and southeast) were considered. The skeleton method takes into account the skeleton of the binarized image to build the graph. Pixels of the obtained contour are followed according to their neighborhood to create the nodes and the arcs of the graph every 'ws' pixels (window size). Figure 13 shows an example of the extraction of each of these type of graphs from an image with the digit 6.

Using these three methods and the images from the NIST dataset at a resolution of 64 × 64 pixels, we generated three datasets of graphs with 60,000

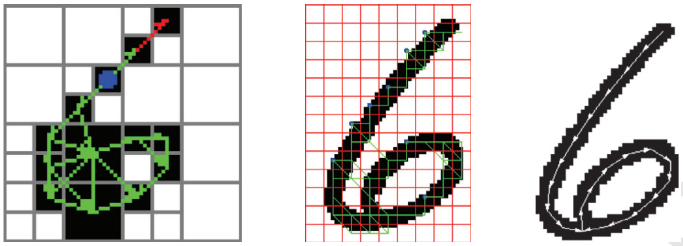


Figure 13: Examples of the three types of graphs extracted from the NIST dataset. From left to right it can be seen the result obtained for the digit 6 using neighborhood graphs, grid graphs and skeleton graphs.

samples each one (6,000 samples of each digit). The tuning parameter of each algorithm were fixed to $d = 4$, $g = 8$, and $ws = 15$. Other parameter values were also tested obtaining a similar performance but obtaining bigger models due to the increased size of the generated graphs.

7.2. Experimental results

In this section we summarize the results of the experiments carried out. These experiments show the good behavior of the learning algorithm we propose [1]. For each dataset we used Algorithm 5.1 to obtain a graph automaton to model each class. The models are then used to classify a set of non-overlapping test samples using the strategy *one-vs.-rest*. That is, each automaton must correctly classify the samples of the class that the automaton models (that we will name *positive class* from now on) and reject the samples from other (*negative*) classes as well.

In order to evaluate the performance of the experiment we use the F-measure (F-m). This is a common metric widely used for two-class classification prob-

[1] For the sake of reproducible research, the code of the experiments, the Hairpin RNA molecules dataset, and the collection of graphs extracted from the NIST dataset are available at `http://github.com/ajgallego/grammatical-inference-of-graphs` under the conditions of the GNU General Public License version 3. The rest of the datasets are publicly available for download.

lems. F-m can be defined by means of Precision and Recall as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{F-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

where TP (True Positives) denotes the number of correctly classified samples from the positive class, FN (False Negatives) the number of misclassified samples from the positive class, and FP (False Positives) the number of negative samples classified as positive.

We note that the Hairpin dataset contains only one class. Thus, we added another heterogeneous class using a selection of 10,000 samples from the other datasets (1,110 samples from the GREC dataset, 4,337 from Mutagenicity, and 4,553 randomly selected samples from the NIST datasets).

We applied a three-fold cross-validation scheme using the datasets previously described. Each dataset was divided into two non-overlapping sets, considering 80% of the samples in each class as training set, using the remaining 20% of the samples in each class as test set. We generate a k-TSS graph automaton per class taking into account k values from 2 to 4. The precision and recall measures were obtained according the description above.

Structural information is sometimes enough to successfully solve applied tasks (for instance, document recognition [31]). In order to illustrate that our approach is also suitable in this situation, we modified the samples of Hairpin, GREC and Mutagenicity datasets in order to erase the labels of the internal nodes. The NIST dataset was not considered because the internal nodes of the graphs obtained were not labeled with discrete values. We used these processed datasets and followed the same scheme explained above.

Table 1 shows the results in terms of Precision, Recall, and F-m (%) obtained for each of the values of k considered together with the number of rules learned by the automaton. The information shown averages the results for all the classes in the dataset and the three folds.

	Dataset	**Hairpin**		**GREC**		**Mutagen.**		**Neighbour**	**Grid**	**Skeleton**
	Node labels?	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**No**	**No**	**No**
$k = 2$	**#rules**	314.51	152.43	34.77	30.82	840.58	321.75	113.89	105.74	97.83
	Prec.	85.96	82.79	87.55	81.25	81.4	78.68	87.13	85.8	89.49
	Recall	90.44	91.64	90.91	91.82	85.52	86.44	92.5	91.83	92.92
	F-m	88.14	86.99	89.2	86.21	83.41	82.38	89.73	88.71	91.17
$k = 3$	**#rules**	3527.15	1278.41	42.68	37.64	3016.24	1815.75	237.19	234.48	204.81
	Prec.	88.96	85.45	89.48	85.17	84.29	81.74	93.87	93.85	95.79
	Recall	88.9	90.69	89.09	90.91	85.29	86.21	90.67	90	91.75
	F-m	88.93	87.99	89.29	87.95	84.79	83.91	92.24	91.89	93.73
$k = 4$	**#rules**	5487.23	3421.65	43.91	40.82	4873.35	3474.62	354.65	376.12	246.34
	Prec.	91.77	86.56	89.48	85.17	86.79	85.43	95.58	94.18	96.81
	Recall	88.75	90.59	89.09	90.91	84.83	85.75	90.17	89.83	91.33
	F-m	90.23	88.53	89.29	87.95	85.8	85.59	92.79	91.96	93.99

Table 1: Average results in terms of Precision, Recall, and F-m (%) obtained by the algorithm of inference using the one-vs.-rest strategy. For each dataset, the results for each value of k next to the number of rules learned by the automaton are included.

As it can be checked, in all cases a similar behavior is observed: when increasing the value of k, the automaton has to learn a larger number of rules. This is due to the fact that the elements of the k-testability vector that represent the graph language have greater variability. Nevertheless, we note that this particularity allows the automaton to better discriminate among the different classes. We note that, in the case of the GREC dataset, similar results were obtained for $k = 3$ and $k = 4$. This is because the average graph diameter is close to 3, and, therefore, few rules related to graphs of diameter 4 are generated.

It can also be checked that, the use of structural information lead to obtain automata with a fewer number of rules. This is due to the number of possible elements in the k-testability vector is more reduced. This also causes Precision and F-m to worsen, and Recall to increase, which is due to the fact that the automaton is giving more false positives.

This experiment shows how the proposed inference algorithm is able to model, at the class level, the different types of graphs considered and to distinguish them from the rest of the samples in the same dataset. Evaluating the quality of the classification by class or the probability of belonging to a class is out of the scope of this paper and is left as future work.

8. Conclusions and Future work

Graphs are used in many applied tasks because of their power of representation. For instance, trees, as the simpler class of graphs, have been widely applied in pattern recognition. Statistical tree language inference is also the basis of a good compression algorithm [36]. Nevertheless, when general graphs are used, they are usually reduced or represented by simpler formalisms because of their high operational complexity.

Stochastic models based on *k-grams* output the probability of the next element in a sequence taking into account the last $k-1$ elements already observed. This approach to pattern recognition has proved to be fruitful in many tasks. From a theoretical point of view, these models can be regarded as a probabilistic extension of strictly locally testable string languages [26].

In this paper, we extend the well-known families of k-testable and k-TSS languages to directed-graph languages. As far as we know, this is the first result that characterizes a graph language class taking into account features of the elements in the language, instead of the properties of the language generating machine (graph grammar).

The definition of k-testable and k-TSS languages support general directed graph languages (those that may contain cycles), nevertheless, the automata model proposed, as well as the inference algorithm do not so, and are focused to directed acyclic graphs. The main problems to extend the results to general directed graphs are the need to establish a processing order and the accepting criterion (because both the zero-incoming and zero-outgoing sets of nodes may be empty).

The experimentation carried out shows the suitability of the proposed inference algorithm to model different types of graphs. In this experimentation, a synthetic datatset and three real-data datasets were processed to model with graphs the information contained in them. Further improvements of the graph model (such as an error correcting parser, a smooth procedure or the addition of probabilities) would indeed be very useful in applied tasks.

In this paper we do not tackle the task of generating graphs of a given k-TSS graph language. Of course, both the definition of k-TSS graph grammars, as well as the task of obtaining, from a given k-TSS graph automaton, an equivalent graph grammar, are quite interesting, and would help to propose a proper stochastic extension.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] D. Angluin and C.H. Smith. Inductive inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.

[2] D. Berwanger and D. Janin. Automata on directed graphs: edge versus vertex marking. *LNCS*, 4178:46–60, 2006. Proceedings of 3rd International Workshop on Software Evolution Through Transformations.

[3] H. Blockeel and S. Nijssen. Induction of node label controlled graph grammar rules. In *Proceedings of 6th International Workshop on Mining and Learning with Graphs*, 2008.

[4] H. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

[5] F. J. Branderburg and K. Skodinis. Finite graph automata for linear and boundary graph languages. *Theoretical Computer Science*, 332:199–232, 2005.

[6] R. C. Carrasco, J. Oncina, and J. Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning*, 44(1-2):185–197, 2001.

[7] A. Clark and R. Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745, 2007.

[8] A. Clark, R. Eyraud, and A. Habrard. A polynomial algorithm for the inference of context free languages. *LNAI*, 5278:29–42, 2008. Proceedings of ICGI-08.

[9] Ph. Dosch and E. Valveny. Report on the Second Symbol Recognition Contest. *LNCS*, 3926:381–397, 2005. Proc. 6th Int. Workshop on Graphics Recognition, GREC'05.

[10] C. Costa Florêncio. Learning node controlled graph grammars. *LNAI*, 5278:286–288, 2008. Proceedings of ICGI-08.

[11] C. Costa Florêncio. Identification of hyperedge-replacement graph grammars. In *Proceedings of 7th International Workshop on Mining and Learning with Graphs*, 2009.

[12] A. Javier Gallego-Sánchez, J. Calera-Rubio, and Damián López. Structural Graph Extraction from Images. Int. Symp. on Distributed Computing and Artificial Intelligence (DCAI 2012), Salamanca, Spain, 2012.

[13] P. García and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:920–925, 1990.

[14] P. García. Learning k-testable tree sets from positive data. Technical Report DSIC/II/46/1993, Dpto. Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 1993. Available on: `http://www.dsic.upv.es/users/tlcc/tlcc.html`.

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

[16] C. de la Higuera. Grammatical inference. Learning automata and grammars. *Cambridge University Press*, 2010.

[17] E. Jeltsch and H. J. Kreowski. Grammatical inference based on hyperedge replacement. *LNCS*, 532:461–474, 1991. 4th International workshop on graph grammars and their application to computer science.

[18] I. Jonyer, L. B. Holder, and D. J. Cook. Concept formation using graph grammars. In *Proceedings of the KDD Workshop on Multi-Relational Data Mining*, pages 71–79, 2002.

[19] J. Kazius, R. McGuire and R. Bursi. Derivation and Validation of Toxicophores for Mutagenicity Prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005.

[20] R. Kosala, H. Blockeel, M. Bruynooghe, and J. Van den Bussche. Information extraction from documents using k-testable tree automaton inference. *Data & Knowledge Engineering*, 58:129–158, 2006.

[21] J. P. Kukluk, L. B. Holder, and D. J. Cook. Inference of node replacement recursive graph grammars. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, 2006.

[22] D. López, J. Calera-Rubio, and A. Javier Gallego-Sánchez. Inference of k-testable directed acyclic graph languages. *Journal of Machine Learning Research: Workshop and Conference Proceedings*, 21:149–163, 2012.

[23] D. López, J. M. Sempere, and P. García. Inference of reversible tree languages. *IEEE Transactions on System Man. and Cybernetics, Part B: Cybernetics*, 34(4):1658–1665, 2004.

[24] E. M. Lucks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

[25] H. Matsui, K. Sato, and Y. Sakakibara. Pair stochastic tree adjoining grammars for aligning and predicting pseudoknot RNA structures. *Bioinformatics*, 21(11):2611–2617, 2005.

[26] R. McNaughton and S. Papert. Counter-free automata. *The MIT Press*, 1971.

[27] R. McNaughton. Algebraic decision procedures for local testability. *Math. Sysr. Theory*, 8(1):60–76, 1974.

[28] G.L. Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 225–235, 1980.

[29] Sourav Mukherjee and Tim Oates. Estimating graph parameters using graph grammars. *LNAI*, 5278:292–294, 2008. 9th Intl. Colloquium, ICGI'08.

[30] K. Nakamura. Incremental learning of context free grammars by bridging rule generation and search for semi-optimum rule sets. *LNAI*, 4201:72–83, 2006. Proceedings of ICGI-06.

[31] I. Perea and D. López. Syntactic modelling and recognition of document images. *LNCS*, 3138:416–424, 2004. Proceedings of S+SSPR 2004.

[32] P. Peris, D. López, M. Campos, and J.M. Sempere. Protein motif prediction by grammatical inference. *LNAI*, 4201:175–187, 2006. 8th Intl. Colloquium, ICGI'06.

[33] P. Peris, D. López, and M. Campos. IgTM: An algorithm to predict transmembrane domains and topology in proteins. *BMC - Bioinformatics*, 9:367, 2008.

[34] L. Pitt. Inductive inference, DFAs, and computational complexity. *LNAI*, 397:18–44, 1989. Proc. 2nd Workshop on Analogical and Inductive Inference.

[35] A. Potthoff, S. Seibert, and W. Thomas. Nondeterminism versus determinism on finite automata over directed acyclic graphs. *Bull. Belg. Math. Soc.*, 1:285–298, 1994.

[36] J. R. Rico-Juan, J. Calera-Rubio, and R. C. Carrasco. Smoothing and compression with stochastic k-testable tree languages. *Pattern Recognition*, 38:1420–1430, 2005.

[37] K. Riesen and H. Bunke. *IAM* Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. SSPR, 2008.

[38] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56: 133–152, 1998.

[39] G. Rozemberg, editor. *Handbook of graph grammars and computing by graph transformation*, volume 1. Word Scientific, 1997.

[40] Y. Sakakibara. Learning Context-Free Grammars from Structural Data in Polynomial Time. *Theoretical Computer Science*, 76:223–242, 1990.

[41] Y. Sakakibara. Efficient Learning of Context-Free Grammars from Positive Structural Examples. *Information and Computation*, 97:23–60, 1992.

[42] Y. Sakakibara. Grammatical inference in bioinformatics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1051–1062, 2005.

[43] Y. Sakakibara. Learning context-free grammars using tabular representations. *Pattern Recognition*, 38:1372–1383, 2005.

[44] J. M. Sempere. Learning context-sensitive languages from linear structural information. *LNAI*, 5278:175–186, 2008. Proceedings of ICGI-08.

[45] A. Syropoulos. Mathematics of Multisets. *Multiset Processing. LNCS*, 2235:347–358. 2001. Workshop on Membrane Computing 2000.

[46] J. L. Verdú-Mas, R.C. Carrasco, and J. Calera-Rubio. Parsing with probabilistic strictly locally testable tree languages. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1040–1047, 2005.

[47] R.-A. Wilkinson, J. Geist, S. Janet, P.-J. Grother, and others. The first census optical character recognition system conference. US Department of Commerce, 1992.

[48] T. Yokomori and S. Kobayashi. Learning local languages and their application to DNA sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1067–1079, 1998.

[49] T. Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 298:179–206, 2003.

[50] K. Zhang. A constrained edit distance between unordered labelled trees. *Algorithmica*, 15:205–222, 1996.