Inference of k-Testable Directed Acyclic Graph Languages

Damián López DLOPEZ@DSIC.UPV.ES
Departmento de Sistemas Informàticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
Valencia - 46022, SPAIN

Jorge Calera-Rubio CALERA@DLSI.UA.ES
Antonio-Javier Gallego-Sánchez JGALLEGO@DLSI.UA.ES
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante
Carretera San Vicente del Raspeig s/n
S. Vte. del Raspeig (Alicante) - 03690, SPAIN

Editors: Jeffrey Heinz, Colin de la Higuera and Tim Oates

Abstract

In this paper, we tackle the task of graph language learning. We first extend the well-known classes of *k-testability* and *k-testability in the strict sense* languages to directed graph languages. Second, we propose a graph automata model for directed acyclic graph languages. This graph automata model is used to 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.

Keywords: Graph languages; graph automata; k-testable languages

1. Introduction

Regular string language inference has been widely applied to many tasks, from bioinformatics (Sakakibara, 2005; Peris et al., 2008) to script recognition (Ron et al., 1998). See de la Higuera (2010) for a bibliographic study.

It is important here 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, structural information can be naturally modeled by context-free grammars of various types. In this line of work, Sakakibara 1990; 1992 presented the first algorithms for learning context-free languages with polynomial time complexity. Some other recent results for non-regular language inference study the inference of context-free languages (for instance, Nakamura, 2006; Clark et al., 2008).

Trees are a simple class of graphs with very interesting properties. Thus, several works study the task of tree language inference (García, 1993; López et al., 2004, among others), as well as its application to real tasks (for instance, Rico-Juan et al., 2005; Kosala et al., 2006). 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, the handbook edited by Rozemberg (1997) 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, despite many works study the generating paradigm, there does not exist a recognizing device that properly fits all the different characterized classes of graph languages. Despite this, there have been proposed some graph automata models, for instance Potthoff et al. (1994); Branderburg and Skodinis (2005); Berwanger and Janin (2006).

Some works have studied the inference of graph grammars, but all of them based on the search of general subgraph isomorphism, and therefore, with high time complexity (Jeltsch and Kreowski, 1991; Jonyer et al., 2002; Kukluk et al., 2006)Recently, some works take profit from results on mining in graphs in order to propose graph grammar inference methods (Blockeel and Nijssen, 2008; Florêncio, 2009).

In this paper, we first extend the well-known families of *k-testable* and *k-testable in the strict sense* (k-TSS) languages (McNaughton, 1974) to directed-graph languages. We also prove some lemmas that show that the main features of the k-TSS class of languages are still applicable to graph languages. Second, we take into account the paper by Potthoff et al. (1994) to propose a graph automata model for directed acyclic graph languages, and finally, we propose a polynomial grammatical inference algorithm to learn the same class of directed acyclic k-TSS 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 comparison. Besides, the consideration of directed acyclic graphs also help to further ease the general complexity. We study the time complexity of this algorithm and prove its polynomial behavior.

2. Notation and Definitions

A node-labeled directed graph (from now on referred to as graph if not stated otherwise) can be defined by a tuple $g = (V, E, \mu)$, where V is a finite set of nodes (also called vertexes), $E \subsetneq (V \times V) - id_V$ is the set of edges (where id_V denotes the smallest reflexive relation), and $\mu : V \to \Sigma$ is the node labeling function. We will refer to the components of a graph g as V_g, E_g and μ_g only when necessary. 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*) will be denoted by $idg(v)$ and is 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 $V^n_m(g) = \{v \in V \; : \; idg(v) = n \wedge odg(v) = m\}$, and let ϵ_g denote the empty graph (the graph with no nodes). 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 $V_0(g)$.

Let a typed alphabet Σ^σ_τ be the association of an alphabet Σ with a relation $r \subseteq (\Sigma \times \mathbb{N} \times \mathbb{N})$. This alphabet plays the same role as the plain alphabet in string languages

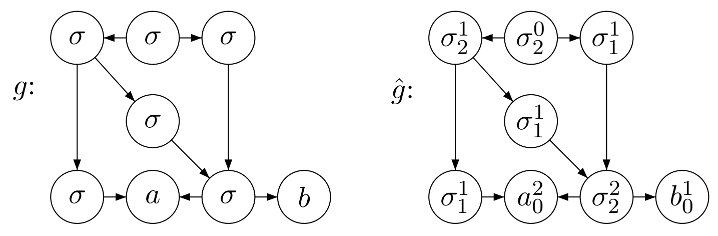

Figure 1: Example of a directed acyclic graph g and its corresponding extended graph $\hat{g}$. In order to give an explicit representation, we label non-frontier nodes (those with outgoing degree greater than 0) with Greek letters.

or the ranked alphabet in tree languages. Since the nodes of the graph may have different incoming and outgoing degrees, it is necessary to establish which symbols can label those nodes. We will denote with Σ_m^n the set: $\{s \in \Sigma \ : \ (s, n, m) \in r\}$, that is, the set of symbols that are able to label the nodes with a given incoming degree n and outgoing degree m.

Once the typed alphabet is defined, the set of all possible consistently labeled graphs can be defined. Formally, let $\mathcal{G}(\Sigma_\tau^\sigma)$ denote the set of graphs over Σ_τ^σ. A *graph language* is any set $L_G \subseteq \mathcal{G}(\Sigma_\tau^\sigma)$.

Given a typed alphabet Σ_τ^σ, 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 labels of the nodes of the extended subgraph $\hat{g}$ explicitly include the incoming and outgoing degrees. Figure 1 shows an example.

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). Nevertheless, our results either support, or can be easily extended, to consider general graphs. In order to give the clearer the better (two-dimensional) graph representations, in the following, 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. A graph is possible to have more than one path between a pair of nodes u and v. Thus, Let us denote the set of shortest paths from u to v by $((u, v))$. Also, let $|((u, v))|$ denote the length of the (possibly multiple) shortest path. Thus, the length of a non-existent path is infinite and $|((u, u))| = 1$. We define the *diameter* of a graph $g = (V, E, \mu)$ as the maximum distance among two connected nodes of the graph. More formally:

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

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 $W = \{u \in V \ : \ |((v, u))| \leq k\}$ 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 $W = \{u \in V \ : \ |((v, u))| < \infty\}$ with the set E' and the labeling function μ' defined as above.

We now recall some definitions from multiset theory which will be used in the transition function of a new graph automata model. In the following, we will denote the set of naturals with $\mathbb{N}$.

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. That is, for any $a \in D$, the function $f(a)$ denotes the number of elements a in the multiset, and we say that a is in A, and write it $a \in A$, if and only if $f(a) \neq 0$. The size of a multiset is defined as the number of elements that a multiset contains. This number can be finite, in which case the multiset is finite. The size of a multiset M will be denoted by $|M|$. We are interested in the class of multisets whose size is equal to a constant n. That is, the class of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$. In the sequel, we will denote this class by $\mathcal{M}_n(D)$.

We say that a multiset $A\langle D, f\rangle$ is *empty* if and only if, for all $a \in D$, $f(a) = 0$. In this way, for any pair of multisets $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$, we say that $A = B$ if and only if, for all $a \in D$, $f(a) = g(a)$, and in the same way, A is a *subset* of B ($A \subseteq B$) if and only if, for all $a \in D$, $f(a) \leq g(a)$. Furthermore, let the sum of two multisets $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ (denoted by $A \oplus B$) be defined as the multiset $C = \langle D, h\rangle$ where for all $a \in D$, $h(a) = f(a) + g(a)$. Finally, we extend in a natural way the definition of *power set* to multisets, thus, for any multiset C, its power set is the set of all possible subsets of C and will be denoted by 2^C.

A very useful concept for dealing with multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \to \mathbb{N}^n$ where $D = \{d_1, d_2, \dots, d_n\}$ and D^* is the set of strings over D. For any $x \in D^*$, this mapping is defined as $\Psi(x) = (\#_{d_1}, \#_{d_2}, \dots, \#_{d_n})$ where $\#_{d_i}$ denotes the number of occurrences of d_j in x. Note that this allows to represent a multiset using whichever string with the correct Parikh mapping. We will do so in the following.

3. k-testable graph languages

Testable and testable in the strict sense languages (McNaughton, 1974) 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 (García and Vidal, 1990; García, 1993). In this section, we extend the definition to consider graph languages.

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

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

Note that $P_k(g) = \emptyset$ if $\mathit{diameter}(g) < k$. Some examples are given below.

Example 1 *Given the graph in Figure 1, Figure 2 shows the components of the* 3*-testability vector. It is worth noting that the nodes of the graphs in each component of the k-testability vector are labeled with the extended function $\hat{\mu}$. 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.*

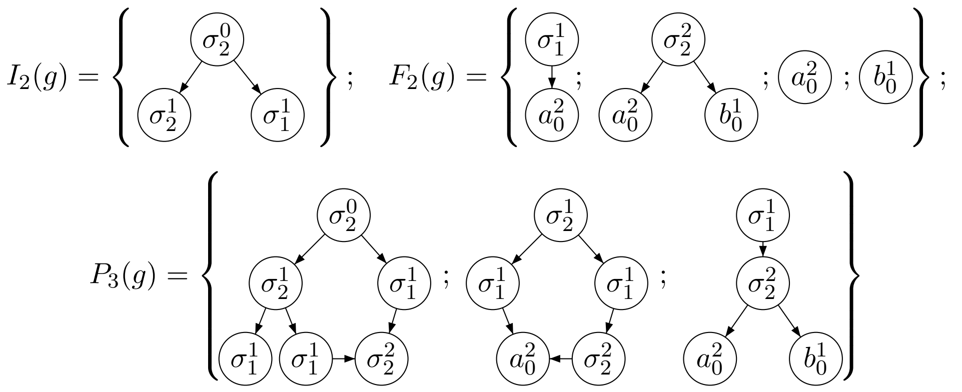

Figure 2: The components of the 3-testability vector for the graph in Figure 1 are shown.

It is not necessary for the graph to be acyclic in order to obtain the k-testability vector. An example is shown in Figure 3. The 2*-testability vector is also shown in the same figure. Note that, in this case, the set $I_1(g)$ is empty because the set $V^0(g)$ is empty.*

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

$$I_k(G) = \{I_k(g) : g \in G\}\,; \quad P_k(G) = \{P_k(g) : g \in G\}\,; \quad F_k(G) = \{F_k(g) : g \in G\}$$

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

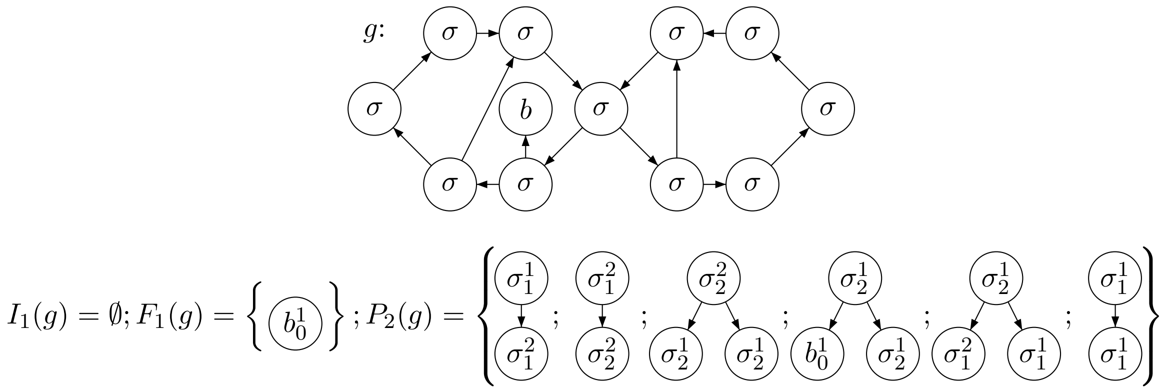

Figure 3: Directed graph and its corresponding 2-testability vector.

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, then, every graph g' such that it has the same k-testability vector than 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 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 $I_{k-1}(g) \subseteq F_{k-1}(g)$.

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 be denoted by $L_k(G)$. We now prove some results related to this class of languages.

Lemma 3 *Let G be a finite set of graphs and $k \geq 2$, then $G \subset L_k(G)$.*
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, are infinite. ■

In string (García and Vidal, 1990) and tree languages (García, 1993), it is proved that, when the value of k is greater than the maximum length (depth in trees) of the elements in a set S, then the k-TSS language obtained from that set (of strings or trees) equals S. This is not generally true when graph languages are taken into account. To enlighten this, please note first that, in this context, it is always possible to obtain new graphs belonging to the k-TSS language by *disconnected joint* of different graphs in the set S. Furthermore, it is possible in some cases to build new graphs taking into account the graphs rooted in the set of nodes with zero incoming arity. As an example, let 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 $L_4(\{g\})$.

Lemma 4 *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 $T \not\subset L_k(G)$.*

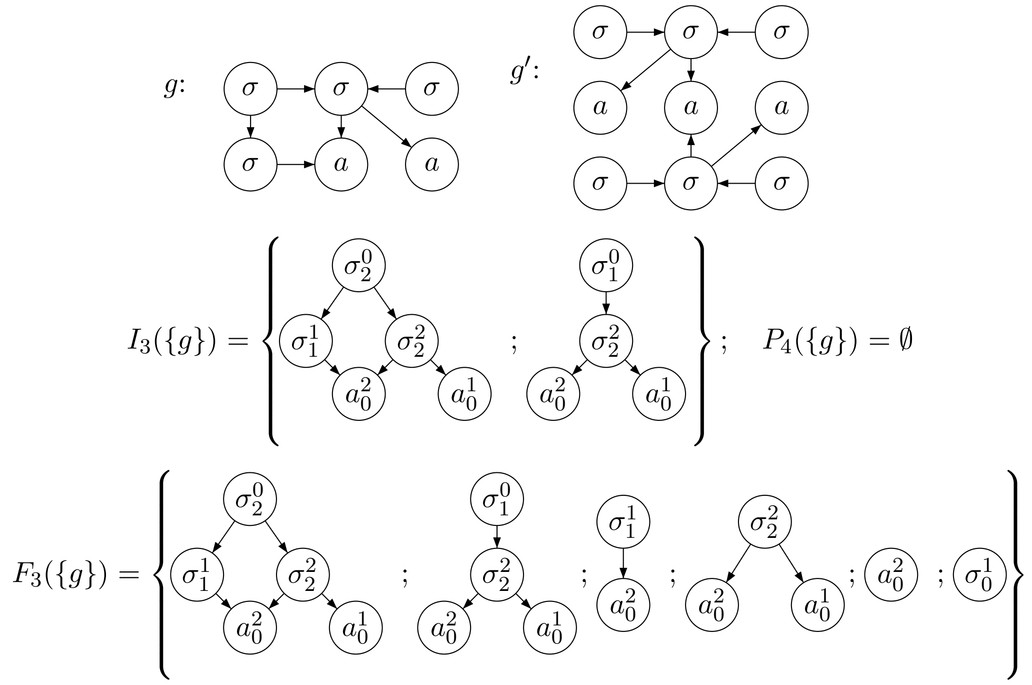

Figure 4: Given the graphs g and g' note that the graph g' belongs to the language $L_4(\{g\})$. The components of the 4-testability vector are also shown.

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 5 *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 6 *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 $L_{k+1}(G) \subseteq L_k(G)$ ■

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

The automata model we propose takes into account the work by Potthoff et al. (1994). We note here that our proposal is not able to process whole class of directed graphs but those without cycles. The transition function of the automata we propose takes into account a multiset which permits, in a natural way, the graphs to be processed without taking into account any order among the nodes except for the partial one induced by the directed edges.

Let us first note that the analysis of any graph is, in essence, similar to the analysis carried out in the context of tree languages, but taking into account that there does not exist any order among the siblings. Second, we also note that this process can be carried out with polynomial complexity using a dynamic programming scheme similar to the proposed in the paper by Zhang (1996). In his paper, Zhang proposes a polynomial algorithm to obtain an edit measure among two unordered trees.

Due to the few graph automata models proposed in the literature, we think it is interesting to provide the more general definition the better, thus, we define our model as non-deterministic.

Definition 7 *Let Σ_τ^σ be a typed alphabet where m denotes the maximum outgoing degree in $\widehat{\Sigma}$. A (non-deterministic) graph automaton for a language over Σ_τ^σ 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_τ^σ, the set $F \subseteq Q$ contains the final states and δ is a set of transition functions defined as follows:*

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

where each δ_j is defined as:

$$\delta_j : \widehat{\Sigma}_j^n \times \mathcal{M}_j(Q) \rightarrow 2^{\mathcal{M}_1(Q)}, \qquad 0 \leq j \leq m$$

and where $\mathcal{M}_j$ represents the class of multisets of size j as defined in Section 2.

We 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 consist of a recursive analysis over each zero-incoming degree node. The multisets returned by these analysis are then summed. Formally, for any given graph g the function δ is extended as follows:

$$\delta : \mathcal{G}(\widehat{\Sigma}) \to 2^{\mathcal{M}(Q)}$$
$$\delta(g) = \delta(\hat{g}) = \bigoplus_{v_i \in V^0(\hat{g})} \delta(R_{\hat{g}}(v_i))$$

where:

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

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.

Let us now define the language accepted by the automaton as follows:

$$L(GA) = \{g \in \mathcal{G}(\Sigma^{\sigma}_{\tau}) \; : \; \forall q \in \delta(\hat{g}), \; q \in F\}$$

Thus, any graph g is accepted by the automaton GA if and only if the extended transition function returns, a multiset such that for every state q with non zero enumeration function, the state q is final. When necessary, we will refer to these multisets as final multisets.

Example 2 *Taking into account the graph shown in Figure 1 and the following automaton, a representation of the analysis is depicted in Figure 5. 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.*

$$\begin{aligned}
\delta(a^2_0, \lambda) &= q_1 \\
\delta(b^1_0, \lambda) &= q_2 \\
\delta(\sigma^2_2, q_1q_2) &= q_1 \\
\delta(\sigma^1_1, q_1) &= q_2 \\
\delta(\sigma^1_2, q_2q_2) &= q_1 \\
\delta(\sigma^0_2, q_1q_2) &= q_1, \; \textit{where } q_1 \in F
\end{aligned}$$

5. 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 (García and Vidal, 1990; García, 1993). 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 $P_k(G)$. Please note that the automaton output by the algorithm is deterministic. An example of run is given below.

$$\begin{tikzcd}
\delta(\sigma_2^1, q_2q_2) = q_1 \arrow[r] & \delta(\sigma_2^0, q_1q_2) = q_1 & \delta(\sigma_1^1, q_1) = q_2 \arrow[l] & \\
 & \delta(\sigma_1^1, q_1) = q_2 \arrow[ul] & & \\
\delta(\sigma_1^1, q_1) = q_2 \arrow[uu] & \delta(a_0^2, \lambda) = q_1 \arrow[l] \arrow[r] & \delta(\sigma_2^2, q_1q_2) = q_1 \arrow[ul] \arrow[uu] & \delta(b_0^1, \lambda) = q_2 \arrow[l]
\end{tikzcd}$$

Figure 5: Example of the parsing of the graph in Figure 1. The graph is recursively traversed to reach those nodes with zero outgoing degree. The arrows show the order of the parsing process once those nodes are reached.

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

```
Require: A set $G$ of graphs. A value $k \geq 2$
Ensure: A graph automaton that recognizes the language $L_k(G)$
Method:
  Compute $(I_{k-1}(G), P_k(G), F_{k-1}(G))$
  Let $\Sigma_\tau^\sigma$ be the typed alphabet from $G$ and $\widehat{\Sigma}_\tau^\sigma$ 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 $g \in F_{k-1}(G)$ with $(v, w_i) \in E_g$, $v \in V^0(g)$, $1 \leq i \leq m$ do
      $\delta_m(\mu(v), Q[R_g(w_1)] \dots Q[R_g(w_m)]) = Q[g]$
  end for
  for all $g \in P_k(G)$ with $v \in V^0(g)$, $(v, w_i) \in E_g$, $1 \leq i \leq m$ do
      $\delta_m(\mu(v), Q[R_g(w_1, k-1)] \dots Q[R_g(w_m, k-1)]) = Q[R_g(v, k-1)]$
  end for
  return $(Q, \widehat{\Sigma}_\tau^\sigma, F, \delta)$
EndMethod:
```

Example 3 *Let us consider $k = 2$, and the set G of graphs shown in Figure 6. The elements of the 2-testability vector are shown in Figure 7.*

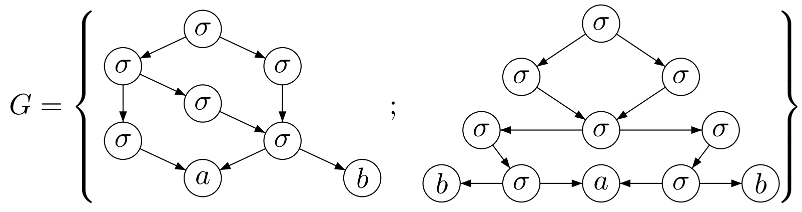

Figure 6: Set of graphs example.

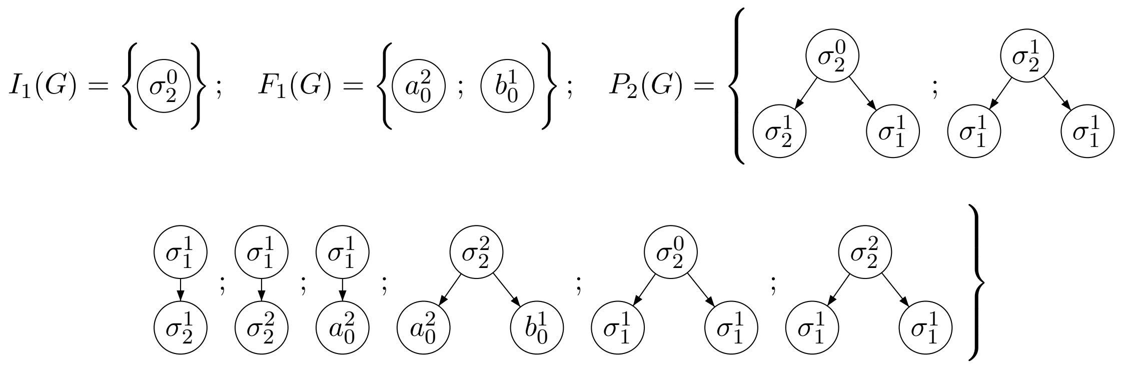

Figure 7: Elements of the 2-testability vector for the graphs example.

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

$$Q\,[\textcircled{a_0^2}] = q_1;\ Q\,[\textcircled{b_0^1}] = q_2;\ Q\,[\textcircled{σ_2^2}] = q_3;\ Q\,[\textcircled{σ_1^1}] = q_4;\ Q\,[\textcircled{σ_2^1}] = q_5;\ Q\,[\textcircled{σ_2^0}] = q_6$$

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 $P_2(G)$:

$$\begin{tikzcd} \sigma_2^1 & \sigma_2^0 \arrow[l] \arrow[r] & \sigma_1^1 \end{tikzcd}$$

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:

$\delta(a_0^2, \lambda) = q_1$	$\delta(\sigma_1^1, q_5) = q_4$
$\delta(b_0^1, \lambda) = q_2$	$\delta(\sigma_1^1, q_1) = q_4$
$\delta(\sigma_2^0, q_5q_4) = q_6$, *where* $q_6 \in F$	$\delta(\sigma_2^2, q_1q_2) = q_3$
$\delta(\sigma_2^1, q_4q_4) = q_5$	$\delta(\sigma_2^0, q_4q_4) = q_6$, *where* $q_6 \in F$
$\delta(\sigma_1^1, q_3) = q_4$	$\delta(\sigma_2^2, q_4q_4) = q_3$

As an example, Figure 8 shows two graphs that were not in the input set and that belong to the 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 8 *Algorithm 5.1 identifies the class of k-TSS graph languages from positive sample.*

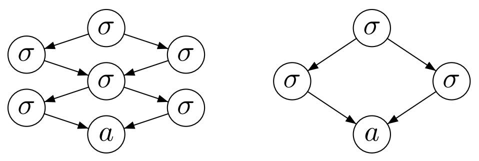

Figure 8: Two graphs not provided in the input set that belong to the example 2-TSS language.

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 the analysis of each of the p nodes with zero incoming arity. In order to accept the graph g, all these analysis 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 $V^0(g)$.

  Now let g be a graph with $v \in V^0(g)$, 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:

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

  Note that there is an edge in g from v to each w_i. Thus, the resulting joint graph with all the $R_{\hat{g}}(w_i, k-1)$, where v is in $P_k(g)$, 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 a final state q ($q \in I_{k-1}(g)$) such that: $\delta(g) = Q[q]$. We will prove the result by induction on the diameter of the graph.

  First, if 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 $diameter(R_g(w_i)) = n \geq k$, it is fulfilled that $F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$.

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

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

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. ■

Finally, we note that our algorithm runs in polynomial time. Let us consider any input set of graphs G over Σ_τ^σ such that m denotes the greatest outgoing degree of the graphs nodes in G. Let also be $k \geq 2$. The time complexity to obtain each transition is bounded by $\mathcal{O}(m \cdot |\widehat{\Sigma_\tau^\sigma}| \cdot m^{k-1})$, 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. The whole inference step implies, at most, the creation of as many transitions as the number of nodes of the graphs in G. Let n denote that number. Thus, the inference process is bounded by $\mathcal{O}(n \cdot |\widehat{\Sigma_\tau^\sigma}| \cdot m^k)$.

6. Conclusions and Future work

In this paper, we extend the well-known families of k-testable and k-TSS languages to directed-graph languages. To our knowledge, this is the first result that characterizes a class of graph languages taking into account the features of the graphs instead of the structure of graph grammar rules. We also propose a model of graph automata that allows us to propose a polynomial time algorithm which identifies the subclass of directed acyclic k-TSS graph languages.

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).

Of course, both the definition of k-TSS graph grammars, and the algorithm to obtain, from any given (k-TSS or not) graph automaton, an equivalent graph grammar, are very interesting problems and should be addressed as future work.

Acknowledgments

Damián López is partially supported by the Spanish Ministerio de Economa y Competitividad under research project TIN2011-28260-C03-01. Jorge Calera-Rubio and Antonio-Javier Gallego-Sánchez thank the Spanish CICyT for partial support of this work through project TIN2009-14205-C04-01, the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778, and the program CONSOLIDER INGENIO 2010 (CSD2007-00018).

References

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.

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.

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

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.

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

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

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

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.

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.

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 71–792, 2002.

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.

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.

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.

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

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.

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

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.

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.

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.

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

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

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

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

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