Exact Learning of Finite Unions of Graph Patterns from Queries

Rika Okada[1,⋆], Satoshi Matsumoto[2], Tomoyuki Uchida[3], Yusuke Suzuki[3], and Takayoshi Shoudai[4]

[1] Dept. of Computer and Media Technologies, Hiroshima City University, Japan
licca_okada@toc.cs.hiroshima-cu.ac.jp
[2] Dept. of Mathematical Sciences, Tokai University, Japan
matumoto@ss.u-tokai.ac.jp
[3] Dept. of Intelligent Systems, Hiroshima City University, Japan
{uchida,y-suzuki}@cs.hiroshima-cu.ac.jp
[4] Dept. of Informatics, Kyushu University, Japan
shoudai@i.kyushu-u.ac.jp

Abstract. A linear graph pattern is a labeled graph such that its vertices have constant labels and its edges have either constant or mutually distinct variable labels. An edge having a variable label is called a variable and can be replaced with an arbitrary labeled graph. Let $\mathcal{GP}(\mathcal{C})$ be the set of all linear graph patterns having a structural feature $\mathcal{C}$ like "having a tree structure", "having a two-terminal series parallel graph structure" and so on. The graph language $GL_{\mathcal{C}}(g)$ of a linear graph pattern g in $\mathcal{GP}(\mathcal{C})$ is the set of all labeled graphs obtained from g by substituting arbitrary labeled graphs having the structural feature $\mathcal{C}$ to all variables in g. In this paper, for any set $\mathcal{T}_*$ of m linear graph patterns in $\mathcal{GP}(\mathcal{C})$, we present a query learning algorithm for finding a set S of linear graph patterns in $\mathcal{GP}(\mathcal{C})$ with $\bigcup_{g\in\mathcal{T}_*} GL_{\mathcal{C}}(g) = \bigcup_{f\in S} GL_{\mathcal{C}}(f)$ in polynomial time using at most $m+1$ equivalence queries and $O(m(n+n^2))$ restricted subset queries, where n is the maximum number of edges of counterexamples, if the number of labels of edges is infinite. Next we show that finite sets of graph languages generated by linear graph patterns having tree structures or two-terminal series parallel graph structures are not learnable in polynomial time using restricted equivalence, membership and subset queries.

1 Introduction

Many electronic data become accessible on Internet. Electronic data such as HTML/XML files, bioinformatics and chemical compounds have graph structures but have no rigid structure. Hence, such data are called *graph structured data*. Especially, graph structured data such as HTML/XML files having tree

⋆ Rika Okada is currently working at Sanyo Girls' Junior and Senior High School, Hiroshima, Japan.

M. Hutter, R.A. Servedio, and E. Takimoto (Eds.): ALT 2007, LNAI 4754, pp. 298–312, 2007.

structures are called *tree structured data*. In the fields of data mining and knowledge discovery, many researchers have developed techniques based on machine learning for analyzing such graph structured data. If we can construct oracles which answer any query in practical time, we can design efficient and effective data mining tools based on query learning algorithms using such oracles. The purpose of our work is to present fundamental learning algorithms for data mining from graph structured data. In this paper, we consider polynomial time learnabilities of finite unions of graph patterns having structured variables, which are knowledge representations for graph structured data, in exact learning model of Angluin [2].

A linear graph pattern is defined as a labeled graph such that its vertices have constant labels and its edges have either constant or mutually distinct variable labels. An edge (u, v) having a variable label is called a *variable*, denoted by $\langle u, v\rangle$, and can be replaced with an arbitrary labeled graph. For example, in Fig. 1, we give a linear graph pattern g having two variables $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ with variable labels x and y, respectively. In the figures of this paper, a variable is represented by a box with lines to its elements. The numbers at these lines indicate the order of the vertices of which a variable consists. The symbol inside a box shows the label of the variable. We can obtain a new linear graph pattern from a linear graph pattern g by substituting an arbitrary linear graph pattern to a variable in g. For example, in Fig. 1, the labeled graph G_3 is obtained from the linear graph pattern g by replacing the variables $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ of g with the labeled graphs G_1 and G_2, respectively.

Web documents like HTML/XML files are expressed by labeled graphs having tree structures. In applications for electrical network and scheduling problems, input data are formalized by labeled graphs having two-terminal series parallel (TTSP for short) graph structures. In order to represent structural features of graph structured data such as "having tree structures" and "having TTSP graph structures", we define *simple* Formal Graph System, which is a restricted class of Formal Graph System (FGS for short) presented by Uchida et al. [15]. FGS is a kind of logic programming systems which directly deals with graph patterns instead of terms in first-order logic. A finite set of clauses on FGS is called an *FGS program*. As examples of simple FGS programs, we give a simple FGS program $\mathcal{OT}$ in Fig. 2 generating all ordered rooted trees and a simple FGS program $\mathcal{TTSP}$ in Fig. 3 generating all TTSP graphs such as F_1, F_2, F_3, F_4, and F_5 in Fig. 3, where TTSP graphs are constructed by recursively applying "series" and "parallel" operations (see [5]). For a simple FGS program Γ, let $\mathcal{GP}(\Gamma)$ be the set of all linear graph patterns obtained from any labeled graph generated by Γ by replacing some edges in it with mutually distinct variables, that is, $\mathcal{GP}(\Gamma)$ contains all linear graph patterns with the graph structural feature "generated by Γ". The graph language $GL_\Gamma(g)$ of a linear graph pattern g in $\mathcal{GP}(\Gamma)$ is the set of all labeled graphs whose graph structures are generated by Γ and which are obtained from g by substituting arbitrary labeled graphs whose graph structures are generated by Γ to all variables in g.

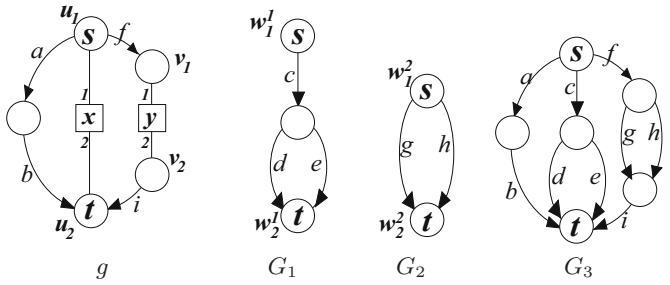

Fig. 1. Linear graph pattern g and labeled graphs $G_1,\ G_2,\ G_3$ over Λ. In the figures in this paper, a variable is represented by a box with lines to its elements. The numbers at these lines indicate the order of the vertices of which a variable consists. The symbol inside a box shows the label of the variable. In this figure, we omit the labels of vertices except two labels $s,\ t$.

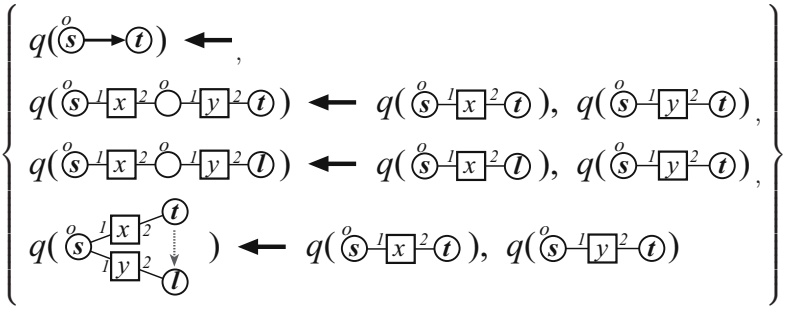

Fig. 2. Simple FGS program $\mathcal{OT}$. The symbol o over internal vertices indicates that the vertex has ordered children. The broken arrow shows that the order of the leaf labeled with t is less than that of the leaf labeled with l.

In exact learning model of Angluin [2], a learning algorithm accesses to oracles, which answer specific kinds of queries, and collects information about a target. Let Γ be a simple FGS program and $\mathcal{T}_*$ a subset of $\mathcal{GP}(\Gamma)$. A learning algorithm is said to *exactly identify* the target set $\mathcal{T}_*$ if it outputs a set of linear graph patterns $S \subseteq \mathcal{GP}(\Gamma)$ such that the union of graph languages of all linear graph patterns in S is equal to that in $\mathcal{T}_*$ and halts, after it asks a certain number of queries to oracles. In this paper, for a simple FGS program Γ and any set $\mathcal{T}_*$ of m linear graph patterns in $\mathcal{GP}(\Gamma)$, we present a query learning algorithm which exactly identifies $\mathcal{T}_*$ in polynomial time using at most $m+1$ equivalence queries and at most $m(n + rn^2)$ restricted subset queries, where n is the maximum number of edges of counterexamples and r is the number of clauses in Γ (i.e., r is a constant), if the number of labels of edges is infinite. Firstly, the algorithm gets a counterexample h_i $(1 \leq i \leq m)$ as an answer of an equivalence query for an empty set, that is, h_i is a labeled graph generated by some linear graph

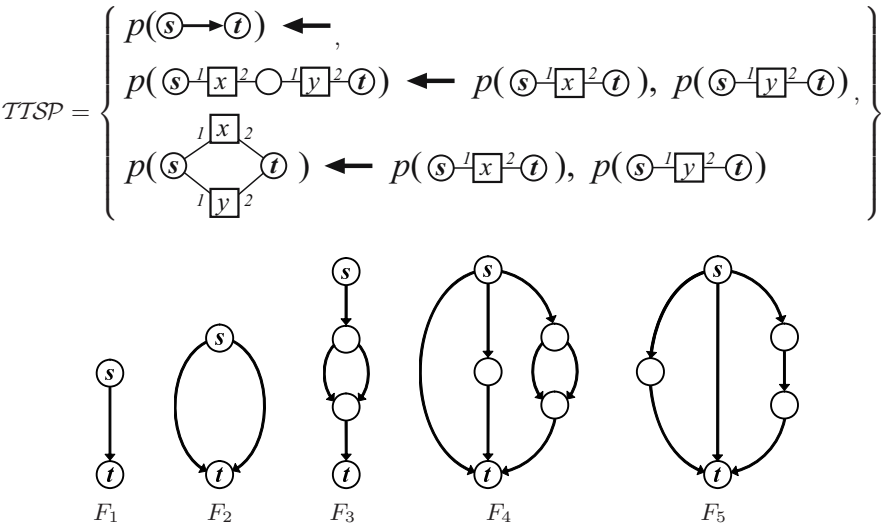

Fig. 3. Simple FGS program $\mathcal{TTSP}$ and TTSP graphs $F_1,\ F_2,\ F_3,\ F_4,\ F_5$. In this figure, we omit the labels of edges and vertices except two labels s and t of vertices.

pattern g_i in $\mathcal{T}_*$. Secondly, the algorithm recursively reconstructs h_i by replacing edges of h_i with variables or subgraphs of h_i generated by Γ with variables and by asking a certain number of restricted subset queries. Next, asking an equivalence query, the algorithm gets a new counterexample h_j $(1 \leq j \neq i \leq m)$ if the equivalence oracle does not answer "yes". Finally, the algorithm halts, if the algorithm exactly identifies $\mathcal{T}_*$.

Next, we show that, by asking a certain number of restricted equivalence, membership and subset queries, finite sets of linear graph patterns in $\mathcal{GP}(\mathcal{OT})$ and $\mathcal{GP}(\mathcal{TTSP})$ are not learnable in polynomial time.

In [10], we already showed that any finite set of m linear graph patterns in $\mathcal{GP}(\mathcal{OT})$ is exactly identifiable at most $m+1$ equivalence queries and using at most $2mn^2$ restricted subset queries, where n is the maximum number of edges in counterexamples, if the number of labels of edges is infinite. The results of this paper are improvements and extensions of the results in [10]. Moreover, in [11], we considered polynomial time learnabilities of finite unions of *nonlinear* graph patterns having ordered tree structures, that is, ordered rooted tree patterns in which variables are allowed to have the same variable labels. In [11], we showed that any finite set of m graph patterns having ordered tree structures is exactly identifiable using $O(m^2n^4+1)$ superset queries and $O(m+1)$ restricted equivalence queries, where n is the maximum number of edges in counterexamples, if the number of labels of edges is infinite.

As for related works, the work [9,16] studied the learnabilities of graph structured patterns in the framework of polynomial time inductive inference. Also the work [13,14] showed the classes of linear graph patterns in $\mathcal{GP}(\mathcal{OT})$ and lin-

ear graph patterns in $\mathcal{GP}(\mathcal{TTSP})$ are polynomial time inductively inferable from positive data, respectively. As an application, the work [12] proposed a tag tree pattern, which is an extension of a linear graph pattern in $\mathcal{GP}(\mathcal{OT})$, and gave a data mining method from tree structured data. As for other related works, the works [1,3] show the exact learnability of tree structured patterns, which are incomparable to linear graph patterns having tree structures, in the exact learning model.

This paper is organized as follows: In Section 2, we formally define a linear graph pattern as a labeled graph having structural variables, and then define its graph language. Moreover, we briefly introduce a exact learning model treated in this paper. In Section 3, we consider the learnabilities of finite unions of graph languages of linear graph patterns in the framework of exact learning model. In Section 4, we consider the insufficiency of learning of finite unions of some graph languages of linear graph patterns in exact learning model. In Section 5, we conclude this work and give future works.

2 Preliminaries

We introduced *term graphs* and *term graph languages* in [15] in order to develop efficient graph algorithms for grammatically defined graph classes. In this section, based on term graphs and term graph languages, we define labeled graph patterns as graphs having structural variables, and then introduce their graph languages. For a set S, $|S|$ denotes the number of elements of S.

2.1 Linear Graph Patterns

Let Λ and $\mathcal{X}$ be infinite alphabets whose elements are called *constant labels* and *variable labels*, respectively. We assume that $\Lambda \cap \mathcal{X} = \emptyset$. Let $G = (V,\ E)$ be a directed labeled graph consisting of a set V of vertices and a set E of edges such that G has no loop but multiple edges are allowed. We denote by ψ_G a vertex labeling assigning a constant label in Λ to each vertex in V and by φ_G an edge labeling assigning either a constant label or a variable label in $\Lambda \cup \mathcal{X}$ to each edge in E. A *graph pattern over* $\Lambda \cup \mathcal{X}$ *obtained from* G is defined as a triplet $g = (V_g,\ E_g,\ H_g)$ where $V_g = V$, $E_g = \{e \in E \mid \varphi_G(e) \in \Lambda\}$ and $H_g = E - E_g$. An element of H_g is called a *variable*. We note that $\psi_g(u) = \psi_G(u)$ for each vertex $u \in V_g$, $\varphi_g(e) = \varphi_G(e) \in \Lambda$ for each edge $e \in E_g$ and $\varphi_g(h) = \varphi_G(h) \in \mathcal{X}$ for each variable $h \in H_g$. We use notations (u, v) and $\langle s, t\rangle$ to represent an edge in E_g and a variable in H_g consisting of two vertices u, v and s, t in V_g, respectively. Here after, since the background graph G can be easily found from a triplet $g = (V_g,\ E_g,\ H_g)$, we omit the description of the background graph G. A graph pattern g over $\Lambda \cup \mathcal{X}$ is said to be *linear* if all variables in g have mutually distinct variable labels in $\mathcal{X}$. In particular, a graph pattern over $\Lambda \cup \mathcal{X}$ with no variable is regarded as a (standard) labeled graph over Λ. We denote the set of all linear graph patterns over $\Lambda \cup \mathcal{X}$ by $\mathcal{GP}_{\Lambda \cup \mathcal{X}}$ and the set of all labeled graphs over Λ by $\mathcal{G}_\Lambda$. In this paper, we deal with only linear graph patterns over

$\Lambda \cup \mathcal{X}$, and then we call a linear graph pattern over $\Lambda \cup \mathcal{X}$ a *graph pattern*, simply. A graph pattern having no edge is said to be *simple*. A graph pattern g is said to be *primitive* if g is a simple graph pattern consisting of two vertices and only one variable, (i.e. $|V_g| = 2$, $|E_g| = 0$ and $|H_g| = 1$, where $g = (V_g,\ E_g,\ H_g)$).

Two graph patterns $f = (V_f,\ E_f,\ H_f)$ and $g = (V_g,\ E_g,\ H_g)$ are said to be *isomorphic*, denoted by $f \equiv g$, if there is a bijection π from V_f to V_g, such that (1) $(u, v) \in E_f$ if and only if $(\pi(u), \pi(v)) \in E_g$, (2) $\psi_f(u) = \psi_g(\pi(u))$ for each vertex $u \in V_f$ and $\varphi_f((u, v)) = \varphi_g((\pi(u), \pi(v)))$ for each edge $(u, v) \in E_f$, and (3) $\langle u, v\rangle \in H_f$ if and only if $\langle \pi(u), \pi(v)\rangle \in H_g$. A bijection π satisfying (1)–(3) is called an *isomorphism* from f to g. Two isomorphic graph patterns are considered to be identical.

Let f and g be graph patterns having at least two vertices. Let $\sigma = [u, v]$ be a pair of distinct vertices in g. The form $x := [g, \sigma]$ is called a *binding* for a variable label x in $\mathcal{X}$. A new graph pattern, denoted by $f\{x := [g, \sigma]\}$, is obtained by applying the binding $x := [g, \sigma]$ to f in the following way: Let $e = \langle s, t\rangle$ be a variable in f with the variable label x, i.e., $\varphi_f(e) = x$. Let g' be a copy of g. And let u' and v' be the vertices of g' corresponding to u and v of g, respectively. For the variable $e = \langle s, t\rangle$, we attach g' to f by removing the variable e from f and identifying the vertices s and t with the vertices u' and v' of g', respectively. For two bindings $x := [g, [u_g, v_g]]$ and $x := [f, [u_f, v_f]]$, we write $(x := [g, [u_g, v_g]]) \equiv (x := [f, [u_f, v_f]])$ if there exists an isomorphism π from g to f such that $\pi(u_g) = u_f$ and $\pi(v_g) = v_f$. A *substitution* θ is a finite set of bindings $\{x_1 := [g_1, \sigma_1],\ x_2 := [g_2, \sigma_2], \ldots,\ x_n := [g_n, \sigma_n]\}$, where x_i's are mutually distinct variable labels in $\mathcal{X}$. For a graph pattern f and a substitution θ, we denote by $f\theta$ the graph pattern obtained from f and θ by applying all bindings in θ to f simultaneously. For example, for the graph pattern g in Fig. 1 and labeled graphs $G_1,\ G_2,\ G_3$ in Fig. 1, G_3 is isomorphic to the graph pattern $g\theta$ obtained by applying $\theta = \{x := [G_1,\ [w_1^1,\ w_2^1]],\ y := [G_2,\ [w_1^2,\ w_2^2]]\}$ to g (i.e., $G_3 \equiv g\theta$).

For graph patterns f and g, we write $f \preceq g$ if there exists a substitution θ such that $f \equiv g\theta$. Especially, we write $f \prec g$ if $f \preceq g$ and $g \not\preceq f$. For example, for the graph patterns G_3 and g given in Fig. 1, we can see that $G_3 \prec g$ because of $G_3 \equiv g\{x := [G_1,\ [w_1^1,\ w_2^1]],\ y := [G_2,\ [w_1^2,\ w_2^2]]\}$ and $g \not\preceq G_3$.

2.2 Graph Languages over Λ

The purpose of this subsection is to define graph languages over an alphabet Λ of infinitely many constant labels (i.e., $|\Lambda| = \infty$). First of all, in order to represent structural features of graph structured data like "having tree structures", "having TTSP graph structures" and so on, we introduce *simple* Formal Graph System, which is a restricted class of Formal Graph System (FGS for short) presented by Uchida et al. [15]. FGS is a kind of logic programming systems which directly deals with graph patterns instead of terms in first-order logic.

Let Π be a set of unary predicate symbols and Σ a finite subset of Λ. An *atom* is an expression of the form $p(g)$, where p is a unary predicate symbol in Π and g is a graph pattern over $\Sigma \cup \mathcal{X}$. For two atoms $p(g)$ and $q(f)$, we write

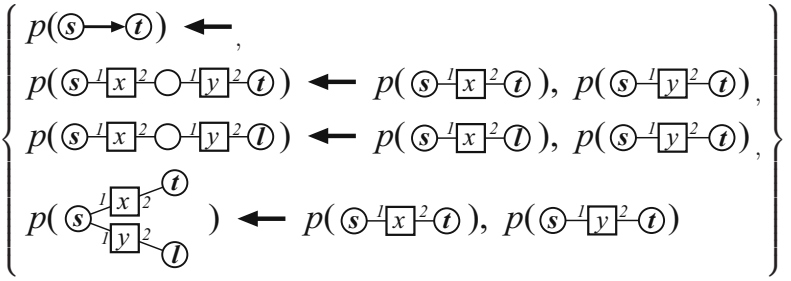

Fig. 4. Simple FGS program $\mathcal{UT}$

$p(g) \equiv q(f)$ if $p = q$ and $g \equiv f$ hold. Let $A, B_1, B_2, \ldots, B_n$ be atoms, where $n \geq 0$. Then, a *graph rewriting rule* is a clause of the form $A \leftarrow B_1, B_2, \ldots, B_n$. We call the atom A the *head* and the right part $B_1, B_2, \ldots, B_n$ the *body* of the graph rewriting rule. For a graph pattern $g = (V_g,\ E_g,\ H_g)$ and a variable label $x \in \mathcal{X}$, the number of variables of g labeled with x is denoted by $o(g, x)$ (i.e., $o(g, x) = |\{h \in H_g \mid \varphi_g(h) = x\}|$). Because any graph pattern is assumed to be linear in this paper, we have $o(g, x) = 1$ if x appears in g, otherwise $o(g, x) = 0$. A graph rewriting rule $p(g) \leftarrow q_1(f_1),\ q_2(f_2), \ldots,\ q_n(f_n)$ is said to be *simple* if the following conditions (1)-(3) hold: (1) f_i is primitive for any $i = 1, 2, \ldots, n$, (2) g consists of two vertices and the edge between them if $n = 0$, otherwise g is simple, and (3) for any variable $x \in X$, $o(g, x) = 1$ if and only if $o(f_1, x) + o(f_2, x) + \cdots + o(f_n, x) = 1$. A *FGS program* is a finite set of graph rewriting rules. An FGS program Γ is said to be *simple* if any graph rewriting rule in Γ is simple. For example, we give some simple FGS programs in Figs. 2–5.

We define substitutions for graph rewriting rules in a similar way to those in logic programming [7]. For an atom $p(g)$, a graph rewriting rule $A \leftarrow B_1, \ldots, B_n$ and a substitution θ, we define $p(g)\theta = p(g\theta)$ and $(A \leftarrow B_1, \ldots, B_n)\theta = A\theta \leftarrow B_1\theta, \ldots, B_n\theta$. Let Γ be an FGS program. The relation $\Gamma \vdash C$ for a graph rewriting rule C is inductively defined as follows.

(1) If $C \in \Gamma$, then $\Gamma \vdash C$.
(2) If $\Gamma \vdash C$, then $\Gamma \vdash C\theta$ for any substitution θ.
(3) If $\Gamma \vdash A \leftarrow B_1, \ldots, B_i, \ldots, B_n$ and $\Gamma \vdash B_i \leftarrow C_1, \ldots, C_m$, then $\Gamma \vdash A \leftarrow B_1, \ldots, B_{i-1}, C_1, \ldots, C_m, B_{i+1}, \ldots, B_n$.

For an FGS program Γ and its predicate symbol p in Π, $GL(\Gamma, p)$ denotes the subset $\{g \in \mathcal{G}_\Sigma \mid \Gamma \vdash p(g) \leftarrow\}$ of $\mathcal{G}_\Sigma$. We say that a subset $L \subseteq \mathcal{G}_\Sigma$ is an *FGS language* if there exists an FGS program Γ and its predicate symbol p such that $L = GL(\Gamma, p)$. The FGS language $GL(\Gamma, p)$ is simply denoted by $GL(\Gamma)$ if we need not clarify the predicate symbol p. For example, for the simple FGS programs $\mathcal{OT}$ in Fig. 2, $\mathcal{TTSP}$ in Fig. 3, $\mathcal{UT}$ in Fig. 4 and $\mathcal{MT} = \mathcal{UT} \cup \mathcal{OT} \cup \mathcal{R}$ (here, $\mathcal{R}$ in Fig. 5), the FGS languages $GL(\mathcal{OT}, q)$, $GL(\mathcal{TTSP}, p)$, $GL(\mathcal{UT}, p)$ and $GL(\mathcal{MT}, r)$ of $\mathcal{OT}$, $\mathcal{TTSP}$, $\mathcal{UT}$ and $\mathcal{MT}$ are the sets of all rooted ordered trees, all TTSP graphs (see [5]), all rooted unordered trees, and all rooted mixed

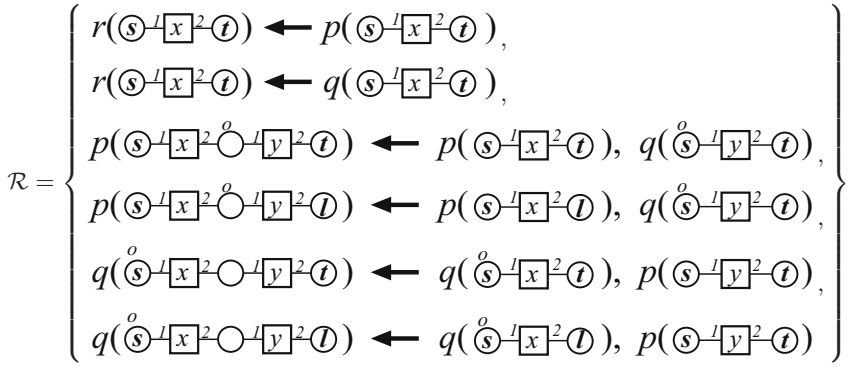

Fig. 5. Simple FGS program $\mathcal{R}$

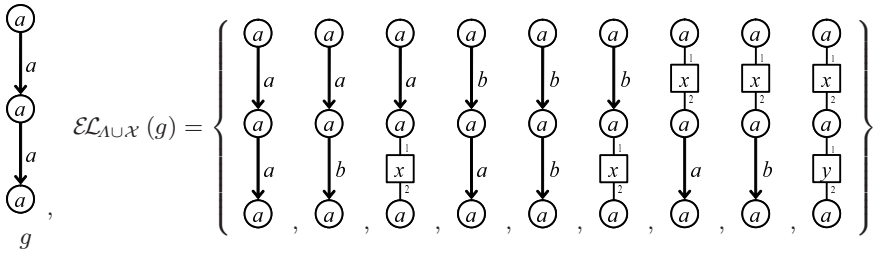

Fig. 6. The set $\mathcal{EL}_{\Lambda\cup\mathcal{X}}(g)$ of all graph patterns over $\Lambda \cup \mathcal{X}$ obtained from the labeled graph g over $\{a\}$ in case of $\Lambda = \{a, b\}$

trees each of whose internal vertices has ordered or unordered children (see [14]), respectively.

Next, we construct graph languages over an infinite alphabet Λ from FGS languages over Σ. For a labeled graph $g \in \mathcal{G}_\Sigma$, $\mathcal{EL}_{\Lambda\cup\mathcal{X}}(g)$ and $\mathcal{EL}_\Lambda(g)$ denote the sets of all graph patterns over $\Lambda \cup \mathcal{X}$ and all labeled graphs over Λ which are obtained from g by ignoring all edge labels of g and relabeling all edges with arbitrary labels in $\Lambda \cup \mathcal{X}$ and Λ, respectively. When $|\Lambda| = \infty$, this indicates that $\mathcal{EL}_{\Lambda\cup\mathcal{X}}(g)$ and $\mathcal{EL}_\Lambda(g)$ are infinite subsets of $\mathcal{GP}_{\Lambda\cup\mathcal{X}}$ and $\mathcal{G}_\Lambda$, respectively. In Fig. 6, as an example in case of $\Lambda = \{a, b\}$, we give the set $\mathcal{EL}_{\Lambda\cup\mathcal{X}}(g)$ of all graph patterns over $\Lambda\cup\mathcal{X}$ obtained from the labeled graph g over $\{a\}$. For a simple FGS program Γ, let $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma) = \bigcup_{g\in GL(\Gamma)} \mathcal{EL}_{\Lambda\cup\mathcal{X}}(g)$ and $\mathcal{G}_\Lambda(\Gamma) = \bigcup_{g\in GL(\Gamma)} \mathcal{EL}_\Lambda(g)$. We denote all the finite subsets of $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ by $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$. For a simple FGS program Γ and a graph pattern $g \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, let $\mathcal{L}_\Lambda(\Gamma, g) = \{f \in \mathcal{G}_\Lambda(\Gamma) \mid f \preceq g\} \subseteq \mathcal{G}_\Lambda$ and we call $\mathcal{L}_\Lambda(\Gamma, g)$ the *graph language of* Γ *and* g. For a simple FGS program Γ and a finite subset S of $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, we define $\mathcal{L}_\Lambda(\Gamma, S)$ as the union of graph languages of Γ and $g \in S$ with respect to S (i.e., $\mathcal{L}_\Lambda(\Gamma, S) = \bigcup_{g\in S} \mathcal{L}_\Lambda(\Gamma, g)$) and we call it the *graph language over* Γ *and* S. In particular, we assume that $\mathcal{L}_\Lambda(\Gamma, \phi) = \phi$.

Let Γ be a simple FGS program, g a graph pattern in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ and S in $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$. Then, we consider the following property: $\mathcal{L}_\Lambda(\Gamma, g) \subseteq \mathcal{L}_\Lambda(\Gamma, f)$ for some $f \in S$ if and only if $\mathcal{L}_\Lambda(\Gamma, g) \subseteq \mathcal{L}_\Lambda(\Gamma, S)$. This property is important in the learning of unions of graph languages and called *compactness*, which was proposed in [4]. The following lemma shows that the graph language over a simple FGS program Γ and $S \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ has compactness. We can prove the following lemma by slightly modifying the proof of Lemma 1 in [10].

Lemma 1. *Let Γ be a simple FGS program, S in $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ and $|\Lambda|$ infinite. Then, for a graph pattern g in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, $\mathcal{L}_\Lambda(\Gamma, g) \subseteq \mathcal{L}_\Lambda(\Gamma, S)$ if and only if there exists a graph pattern f in S with $g \preceq f$.*

In this paper, we consider polynomial time learnabilities of the class of graph languages for a simple FGS program Γ and a finite subset S of $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$. We remark that we do not consider the learnabilities of FGS languages.

2.3 Learning Model

Let Γ be a simple FGS program. In what follows, let $\mathcal{T}_* \subseteq \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ (i.e., $\mathcal{T}_* \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$) denotes a finite set of graph patterns to be identified, and we say that $\mathcal{T}_*$ is a *target*. In the exact learning model via queries due to Angluin [2], learning algorithms can access to *oracles* that will answer queries about the target $\mathcal{T}_*$. In this paper, we consider the following queries.

1. **Membership query:** The input is a labeled graph $g \in \mathcal{G}_\Lambda(\Gamma)$. The output is `yes` if $g \in \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, otherwise `no`. The oracle which answers the membership query is called a *membership oralce*.
2. **Subset query and Restricted subset query:** The input of both queries is a finite subset S of $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$. The output of a subset query is `yes` if $\mathcal{L}_\Lambda(\Gamma, S) \subseteq \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, otherwise a labeled graph, called a *counterexample*, in $(\mathcal{L}_\Lambda(\Gamma, S) - \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*))$. The oracle which answers the subset query is called a *subset oracle*. The output of a restricted subset query is `yes` if $\mathcal{L}_\Lambda(\Gamma, S) \subseteq \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, otherwise `no`. The oracle which answers the restricted subset query is called a *restricted subset oracle*.
3. **Equivalence query and Restricted equivalence query:** The input of both queries is a finite subset S of $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$. The output of a equivalence query is `yes` if $\mathcal{L}_\Lambda(\Gamma, S) = \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, otherwise a labeled graph, called a *counterexample*, in $(\mathcal{L}_\Lambda(\Gamma, S) \cup \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)) - (\mathcal{L}_\Lambda(\Gamma, S) \cap \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*))$. The oracle which answers the equivalence query is called a *equivalence oracle*. The output of a restricted equivalence query is `yes` if $\mathcal{L}_\Lambda(\Gamma, S) = \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, otherwise `no`. The oracle which answers the restricted equivalence query is called a *restricted equivalence oracle*.

A learning algorithm $\mathcal{A}$ is said to *exactly identify a target $\mathcal{T}_*$ in polynomial time* if $\mathcal{A}$ outputs a set $S \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ in polynomial time with $\mathcal{L}_\Lambda(\Gamma, S) = \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$.

Algorithm LEARN_UNION

Assumption: A simple FGS program Γ and a target $\mathcal{T}_* \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$.
Given: Oracles for $Equiv_{\mathcal{T}_*}$ and $rSub_{\mathcal{T}_*}$ for $\mathcal{T}_*$.
Output: A set $S \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ with $\mathcal{L}_\Lambda(\Gamma, S) = \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$.

begin
1. $S := \emptyset$;
2. **while** $Equiv_{\mathcal{T}_*}(S) \neq$ `yes` **do**
3. **begin**
4. Let g be a counterexample;
5. **foreach** edge e of g **do**
6. **if** $rSub_{\mathcal{T}_*}(\{g/\{e\}\}) =$ `yes` **then** $g := g/\{e\}$;
7. **repeat**
8. **foreach** $f \in \{g' \mid g \dashv_\Gamma g'\}$ **do**
9. **if** $rSub_{\mathcal{T}_*}(\{f\}) =$ `yes` **then begin** $g := f$; **break end**
10. **until** g does not change;
11. $S := S \cup \{g\}$
12. **end**;
13. **output** S

end.

Fig. 7. Algorithm LEARN_UNION

3 Learning Finite Unions of Graph Languages

In this section, for a fixed simple FGS program Γ, we consider the learnabilities of finite unions of graph languages over Γ and $S \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ in the framework of exact learning model. For a simple FGS program Γ and a target $\mathcal{T}_*$ in $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, we present a polynomial time learning algorithm LEARN_UNION in Fig. 7 which outputs a set S in $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ such that $\mathcal{L}_\Lambda(\Gamma, S) = \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$ holds, by asking several queries to a restricted subset oracle, denoted by $rSub_{\mathcal{T}_*}$, and an equivalence oracle, denoted by $Equiv_{\mathcal{T}_*}$. The formal definitions of notations used in LEARN_UNION are stated later. We assume that $|\Lambda|$ is infinite.

First, we consider the internal foreach-loop at lines 5 and 6 in the algorithm LEARN_UNION. For two graph patterns $g = (V_g,\ E_g,\ H_g)$ and f in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}$, we write $g \lhd f$ if f is isomorphic to a graph pattern g' obtained from g by replacing an edge $(u, v) \in E_g$ with a new variable $\langle u, v\rangle$ labeled with a new variable label in $\mathcal{X}$ (i.e., $g' = (V_g,\ E_g - \{(u, v)\},\ H_g \cup \{\langle u, v\rangle\})$). That is, f is a generalized graph pattern of g such that $g \preceq f$. In order to show the replaced edge (u, v) explicitly, f is denoted by $g/\{(u, v)\}$. Let $\lhd^*$ be the reflexive and transitive closure of $\lhd$ on $\mathcal{GP}_{\Lambda\cup\mathcal{X}}$. Then, we have the following lemma.

Lemma 2. *For graph patterns $g,\ g_1,\ g_2 \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}$, if $g \lhd^* g_1$ and $g \lhd^* g_2$, then there exists a graph pattern $g' \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}$ such that $g_1 \lhd^* g'$ and $g_2 \lhd^* g'$ hold.*

Proof. Since $g \lhd^* g_1$, there exists a subset $I_1 = \{e_{g_1,1}, e_{g_1,2}, \ldots, e_{g_1,k}\}$ of E_g such that $g/\{e_{g_1,1}\}/\{e_{g_1,2}\}/\ldots/\{e_{g_1,k}\} \equiv g_1$ holds, where E_g is the set of edges in

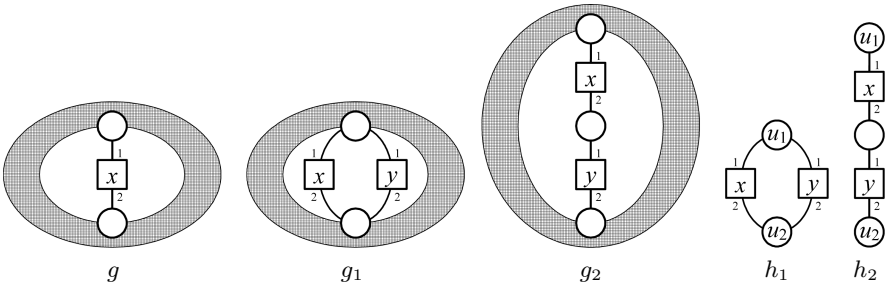

Fig. 8. Graph patterns $g,\ g_1,\ g_2,\ h_1,\ h_2$

g. Moreover, there exists also a subset $I_2 = \{e_{g_2,1}, e_{g_2,2}, \ldots, e_{g_2,r}\}$ of E_g such that $g/\{e_{g_2,1}\}/\{e_{g_2,2}\}/\ldots/\{e_{g_2,r}\} \equiv g_2$ holds. For a set $I_1 \cup I_2 = \{e_1, e_2, \ldots, e_s\}$ $(s \leq k + r)$ and $g' \equiv g/\{e_1\}/\{e_2\}/\ldots/\{e_s\}$, we have $g_1 \lhd^* g'$ and $g_2 \lhd^* g'$. □

This lemma shows that the binary relation $\lhd$ over $\mathcal{GP}_{\Lambda\cup\mathcal{X}}$ has the Church-Rosser property. We can easily prove the following lemma.

Lemma 3. *For two graph patterns $g,\ f$ in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}$, we have $g \preceq f$ if $g \lhd^* f$.*

For a graph pattern g given after executing the internal foreach-loop at lines 5 and 6 in the algorithm Learn_Union, from Lemma 2, we can see that $\mathcal{L}_\Lambda(\Gamma, g) \subseteq \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$ and $\mathcal{L}_\Lambda(\Gamma, g/\{e\}) \not\subseteq \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$ for any edge e in g. Then, by modifying the proof of Lemma 3 in [10], we can prove the following lemma.

Lemma 4. *Let Γ be a simple FGS program, $g = (V_g,\ E_g,\ H_g)$ a graph pattern in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ and S in $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$. If $\mathcal{L}_\Lambda(\Gamma, g) \subseteq \mathcal{L}_\Lambda(\Gamma, S)$ and $\mathcal{L}_\Lambda(\Gamma, g/\{e\}) \not\subseteq \mathcal{L}_\Lambda(\Gamma, S)$ for any edge $e \in E_g$, then there exists a graph pattern $g' = (V_{g'},\ E_{g'},\ H_{g'})$ in S such that $g \preceq g'$ and $|E_g| = |E_{g'}|$ hold.*

Second, we consider the internal repeat-loop between lines 7 and 10 in the algorithm Learn_Union. Let Γ be a simple FGS program. Let g be a graph pattern in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, g' a graph pattern in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}$ and x a variable label appearing in g'. We write $g \dashv_\Gamma g'$ if there exists a graph rewriting rule D in Γ such that $g \equiv g'\{x := [h, \sigma]\}$, that is, if g is a graph pattern obtained from g' by replacing the variable having the variable label x with h, where h is a simple graph pattern appearing in the head of D. For example, for graph patterns $g,\ g_1,\ g_2$ given in Fig. 8, we have $g_1 \dashv_{\mathcal{TTSP}} g$ and $g_2 \dashv_{\mathcal{TTSP}} g$ (i.e., $g_1 \equiv g\{x := [h_1, (u_1, u_2)]\}$ and $g_2 \equiv g\{x := [h_2, (u_1, u_2)]\}$), from the second and the third graph rewriting rules in $\mathcal{TTSP}$, where $h_1,\ h_2$ are simple graph patterns given in Fig. 8 and $\mathcal{TTSP}$ is the simple FGS program in Fig. 3.

For graph patterns $g,\ g' \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, if $g \dashv_\Gamma g'$ then g' is a generalized graph pattern of g such that $g \preceq g'$. Therefore we have the following lemma.

Lemma 5. *Let Γ be a simple FGS program. For two graph patterns $g,\ g'$ in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, if $g \dashv_\Gamma g'$ holds then $g \preceq g'$ holds.*

For a graph pattern g given after executing the internal repeat-loop between lines 7 and 10 in the algorithm LEARN_UNION, we can see that $\mathcal{L}_\Lambda(\Gamma, f) \not\sqsubseteq \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$ for any graph pattern $f \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ with $g \dashv_\Gamma f$.

Lemma 6. *Let Γ be a simple FGS program. Let $g = (V_g,\ E_g,\ H_g)$ be a graph pattern in $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ and S a set in $\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ such that there exists a graph pattern $g' = (V_{g'},\ E_{g'},\ H_{g'})$ in S with $g \preceq g'$ and $|E_g| = |E_{g'}|$. Then, if $\mathcal{L}_\Lambda(\Gamma, f) \not\sqsubseteq \mathcal{L}_\Lambda(\Gamma, S)$ for any graph pattern $f \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ with $g \dashv_\Gamma f$, then $g \equiv g'$ holds.*

Proof. Since $g \preceq g'$ holds, there exists a substitution $\theta = \{x_1 := [f_1, \sigma_1],\ x_2 := [f_2, \sigma_2], \ldots,\ x_n := [f_n, \sigma_n]\}$ such that $g \equiv g'\theta$ holds. Since $|E_g| = |E_{g'}|$, $|H_g| \geq |H_{g'}|$ holds. We assume that $|H_g| > |H_{g'}|$ holds. Then, we can see that there exists a binding $x_\ell := [f_\ell, \sigma_\ell]$ in θ such that $|V_{f_\ell}| \geq 2$, $|E_{f_\ell}| = 0$ and $|H_{f_\ell}| \geq 2$ hold, where $f_\ell = (V_{f_\ell}, E_{f_\ell}, H_{f_\ell})$. Hence, since $g' \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, there exists a substitution θ' such that $g \equiv g'\theta \dashv_\Gamma g'\theta' \in \mathcal{GP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ holds. Since from Lemma 5, $g \preceq g'\theta' \preceq g' \in S$ holds, we have $\mathcal{L}_\Lambda(L, g) \subseteq \mathcal{L}_\Lambda(L, g'\theta') \subseteq \mathcal{L}_\Lambda(L, g')$. This is a contradiction. Thus, we can see that $|H_g| = |H_{g'}|$. We have $g \equiv g'$. □

Let Γ be a simple FGS program. For two sets $P, Q \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$, if there exists a graph pattern $f \in Q$ such that $f \preceq g$ for any $g \in P$, we write $P \sqsubseteq Q$. If $P \sqsubseteq Q$ and $Q \not\sqsubseteq P$, then we write $P \sqsubset Q$. Then, from the above lemmas, the following theorem holds.

Theorem 1. *Let Γ be a simple FGS program. The algorithm* LEARN_UNION *in Fig. 7 exactly identifies any set $\mathcal{T}_* \in \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$ in polynomial time using at most $m+1$ equivalence queries and at most $m(n+rn^2)$ restricted subset queries, where $m = |\mathcal{T}_*|$, n is the maximum number of edges of counterexamples and $r = |\Gamma|$, if the number of labels of edges is infinite.*

Proof. We consider the i-th iteration from the line 2 to the line 12 of the algorithm LEARN_UNION, where $i \geq 1$. Let S_i be a hypothesis given to $Equiv_{\mathcal{T}_*}$ at the line 2 of LEARN_UNION and $g_i = (V_i, E_i, H_i)$ a counterexample given at the line 4 of LEARN_UNION. Assume that $S_0 = \emptyset$. In a similar way to Lemmas 6 and 7 in [10], from Lemmas 1, 3, 4 and 6, we can prove that for every $i \geq 1$, $g_i \in \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, $S_{i-1} \sqsubseteq \mathcal{T}_*$ and $S_{i-1} \sqsubset S_i$ hold. Hence, we can see that the algorithm LEARN_UNION correctly outputs a set S such that $\mathcal{L}_\Lambda(\Gamma, S) = \mathcal{L}_\Lambda(\Gamma, \mathcal{T}_*)$, and that LEARN_UNION terminates in polynomial time.

Next, we consider the numbers of restricted subset queries and equivalence queries. In the loop of the lines 5-6, LEARN_UNION uses at most $|E_i|$ restricted subset queries. Moreover, the loop of the lines 7-10 uses at most $|\Gamma| \times |E_i|^2$ restricted subset queries. The while-loop from the line 2 to the line 12 is repeated at most $|\mathcal{T}_*|$ times. Therefore, LEARN_UNION uses at most $m(n+rn^2)$ restricted subset queries and at most $m+1$ equivalence queries, where $m = |\mathcal{T}_*|$, n the maximum number of edges of counterexamples and $r = |\Gamma|$, if the number of labels of edges is infinite. □

Let $\mathcal{FMT} = \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\mathcal{MT})$, $\mathcal{FUT} = \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\mathcal{UT})$, $\mathcal{FOT} = \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\mathcal{OT})$ and $\mathcal{FTTSP} = \mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\mathcal{TTSP})$. From the definitions of $\mathcal{MT}$ and $\dashv_{\mathcal{MT}}$, we can reduce the number of restricted subset queries as the following corollary.

Corollary 1. *Any set $\mathcal{T}_* \in \mathcal{FMT}$ is exactly identified in polynomial time using at most $m+1$ equivalence queries and at most $m(n+3n^2)$ restricted subset queries, where $m = |\mathcal{T}_*|$ and n is the maximum number of edges of counterexamples, if the number of labels of edges is infinite.*

Moreover, since $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\mathcal{UT})$, $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\mathcal{OT})$ and $\mathcal{GP}_{\Lambda\cup\mathcal{X}}(\mathcal{TTSP})$ are closed with respect to the binary relations $\dashv_{\mathcal{UT}}$, $\dashv_{\mathcal{OT}}$ and $\dashv_{\mathcal{TTSP}}$, respectively, we have the following corollary.

Corollary 2. *The algorithm* Learn_Union *in Fig. 7 exactly identifies any finite set $\mathcal{T}_*$ of either $\mathcal{FUT}$, $\mathcal{FOT}$ or $\mathcal{FTTSP}$ in polynomial time using at most $m+1$ equivalence queries and at most $m(n+n^2)$ restricted subset queries, where $m = |\mathcal{T}_*|$ and n is the maximum number of edges of counterexamples, if the number of labels of edges is infinite.*

4 Hardness Results on the Learnability

In this section, we show the insufficiency of learning of $\mathcal{FUT}$, $\mathcal{FOT}$, $\mathcal{FMT}$ and $\mathcal{FTTSP}$ in exact learning model. For a graph pattern $g = (V_g, E_g, H_g)$, the sum of numbers of edges and variables in g is called the *size* of g, i.e., $|g| = |E_g|+|H_g|$.

Lemma 7. *(László Lovász [8]) Let W_n be the number of all rooted unordered unlabeled trees of size n. Then, $2^{n+1} < W_n < 4^{n+1}$, where $n \geq 5$.*

From the above lemma, if $|\Lambda| \geq 1$, then the number of rooted unordered (ordered, mixed) trees of size n is greater than 2^{n+1}. The following lemma is known to show the insufficiency of learning in exact learning model.

Lemma 8. *(Angluin [2]) Suppose the hypothesis space contains a class of distinct sets $L_1, \ldots, L_N$. If there exists a set $L_\cap$ in the hypothesis space such that for any pair of distinct indices i, j $(1 \leq i, j \leq N)$, $L_\cap = L_i \cap L_j$, then any algorithm that exactly identifies each of the hypotheses L_i using restricted equivalence, membership, and subset queries must make at least $N-1$ queries in the worst case.*

By Lemmas 7 and 8, we have the following Theorems 2 and 3.

Theorem 2. *Let $\mathcal{F}$ be either $\mathcal{FUT}$, $\mathcal{FOT}$ or $\mathcal{FMT}$ and $\mathcal{F}_n$ the collection of all sets in $\mathcal{F}$ each of which contains only graph patterns of size n. Then, any learning algorithm that exactly identifies all sets in $\mathcal{F}_n$ using restricted equivalence, membership and subset queries must make greater than 2^{n+1} queries in the worst case, where $|\Lambda| \geq 1$ and $n \geq 5$.*

Proof. We prove the insufficiently of learning for $\mathcal{FUT}$. In a similar way to it, we can prove the insufficiently of learning for $\mathcal{FOT}$ and $\mathcal{FMT}$. We denote by

$\mathcal{S}_n$ the class of singleton sets of rooted unordered trees of size n. The class $\mathcal{S}_n$ is a subclass of $\mathcal{FUT}$ and for any L and L' in $\mathcal{S}_n$, $L \cap L' = \emptyset$. Since the empty set $\mathcal{L}_\Lambda(\mathcal{UT}, \emptyset) = \emptyset$ is a hypothesis in $\mathcal{FUT}$, by Lemmas 7 and 8, any learning algorithm that exactly identifies all the finite sets of rooted unordered term trees of size n using restricted equivalence, membership and subset queries must make more than 2^{n+1} queries in the worst case, even when $|\Lambda| = 1$. □

Theorem 3. *Any learning algorithm that exactly identifies all sets in $\mathcal{FTTSP}$ each of which contains only graph patterns of size n, using restricted equivalence, membership and subset queries, must make greater than $2^{\frac{n}{2}}$ queries in the worst case, where $|\Lambda| \geq 1$ and $n \geq 10$.*

5 Conclusion

We have considered polynomial time learnabilities of finite unions of graph structured datasets in exact learning model of Angluin [2]. In order to represent structural features of graph structured data, we have given a linear graph pattern with structural features such as "having tree structures" and "having TTSP graph structures" by using Formal Graph System given in [15]. Then, for a simple FGS program Γ, we have shown that any set $\mathcal{T}_*$ of m linear graph patterns is exactly identified in polynomial time using at most $m+1$ equivalence queries and at most $m(n + rn^2)$ restricted subset queries, where n is the maximum number of edges of counterexamples and $r = |\Gamma|$, if the number of labels of edges is infinite. Next, as a negative result, we show that finite sets of linear graph patterns having tree structures and two-terminal series parallel graph structures are not learnable in polynomial time using restricted equivalence, membership and subset queries.

As future works, we will consider polynomial time learnabilities of finite unions of graph patterns having structural features generated by non-simple FGS programs such as planar graphs, balanced binary trees, complete graphs. We conclude by summarizing our results and remained open problems in Table 1.

Table 1. Our results and remained open problems

	polynomial time exact learning	polynomial time inductive inference from positive data
$\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Gamma)$		
$\mathcal{FTTSP}$		
$\mathcal{FMT}$	*Yes*[This Work]	*Open*
$\mathcal{FUT}$		
$\mathcal{FOT}$	*Yes*[10]	*Yes*[6] (for 2 unions)
$\mathcal{FGP}_{\Lambda\cup\mathcal{X}}(\Delta)$	*Open*	*Open*
$\mathcal{FEXGP}_{\Lambda\cup\mathcal{X}}(\mathcal{OT})$	*Yes*[11]	*Open*
$\mathcal{FEXGP}_{\Lambda\cup\mathcal{X}}(\Delta)$	*Open*	*Open*

Here, Γ and Δ are a simple FGS program and a (non-simple) FGS program, respectively. $\mathcal{FEXGP}_{\Lambda\cup\mathcal{X}}(\mathcal{OT})$ denotes the finite sets of (non-linear) graph patterns with ordered tree structures generated by the simple FGS program $\mathcal{OT}$.

References

1. Amoth, T.R., Cull, P., Tadepalli, P.: On exact learning of unordered tree patterns. Machine Learning 44, 211–243 (2001)
2. Angluin, D.: Queries and concept learning. Machine Learning 2, 319–342 (1988)
3. Arimura, H., Sakamoto, H., Arikawa, S.: Efficient learning of semi-structured data from queries. In: Abe, N., Khardon, R., Zeugmann, T. (eds.) ALT 2001. LNCS (LNAI), vol. 2225, pp. 315–331. Springer, Heidelberg (2001)
4. Arimura, H., Shinohara, T., Otsuki, S.: Polynomial time algorithm for finding finite unions of tree pattern languages. In: Proc. NIL-91. LNCS (LNAI), vol. 659, pp. 118–131. Springer, Heidelberg (1993)
5. Duffin, R.J.: Topology of series parallel networks. J. Math. Anal. Appl. 10, 303–318 (1965)
6. Hirashima, H., Suzuki, Y., Matsumoto, S., Uchida, T., Nakamura, Y.: Polynomial time inductive inference of unions of two term tree languages. In: Proc. ILP'06, pp. 92–94 (2006) (short papers)
7. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987)
8. Lovász, L.: Combinatorial Problems and Exercises. ch. Two classical enumeration problems in graph theory. North-Holland Publishing Company (1979)
9. Matsumoto, S., Hayashi, Y., Shoudai, T.: Polynomial time inductive inference of regular term tree languages from positive data. In: ALT 1997. LNCS (LNAI), vol. 1316, pp. 212–227. Springer, Heidelberg (1997)
10. Matsumoto, S., Shoudai, T., Miyahara, T., Uchida, T.: Learning of finite unions of tree patterns with internal structured variables from queries. In: McKay, B., Slaney, J.K. (eds.) AI 2002: Advances in Artificial Intelligence. LNCS (LNAI), vol. 2557, pp. 523–534. Springer, Heidelberg (2002)
11. Matsumoto, S., Suzuki, Y., Shoudai, T., Miyahara, T., Uchida, T.: Learning of finite unions of tree patterns with repeated internal structured variables from queries. In: Gavaldá, R., Jantke, K.P., Takimoto, E. (eds.) ALT 2003. LNCS (LNAI), vol. 2842, pp. 144–158. Springer, Heidelberg (2003)
12. Miyahara, T., Suzuki, Y., Shoudai, T., Uchida, T., Takahashi, K., Ueda, H.: Discovery of frequent tag tree patterns in semistructured web documents. In: Chen, M.-S., Yu, P.S., Liu, B. (eds.) PAKDD 2002. LNCS (LNAI), vol. 2336, pp. 341–355. Springer, Heidelberg (2002)
13. Suzuki, Y., Akanuma, R., Shoudai, T., Miyahara, T., Uchida, T.: Polynomial time inductive inference of ordered tree patterns with internal structured variables from positive data. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS (LNAI), vol. 2375, pp. 169–184. Springer, Heidelberg (2002)
14. Takami, R., Suzuki, Y., Uchida, T., Shoudai, T., Nakamura, Y.: Polynomial time inductive inference of TTSP graph languages from positive data. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 366–383. Springer, Heidelberg (2005)
15. Uchida, T., Shoudai, T., Miyano, S.: Parallel algorithm for refutation tree problem on formal graph systems. IEICE Transactions on Information and Systems E78-D(2), 99–112 (1995)
16. Yamasaki, H., Shoudai, T.: A polynomial time algorithm for finding linear interval graph patterns. In: Proc. TAMC-2007. LNCS, vol. 4484, pp. 67–78. Springer, Heidelberg (2007)