Discovering New Knowledge from Graph Data Using Inductive Logic Programming

Tetsuhiro Miyahara[1], Takayoshi Shoudai[2], Tomoyuki Uchida[1], Tetsuji Kuboyama[3], Kenichi Takahashi[1], and Hiroaki Ueda[1]

[1] Faculty of Information Sciences,
Hiroshima City University, Hiroshima 731-3194, Japan
{miyahara@its, uchida@cs, takahasi@its, ueda@its}.hiroshima-cu.ac.jp
[2] Department of Informatics, Kyushu University 39, Kasuga 816-8580, Japan
shoudai@i.kyushu-u.ac.jp
[3] Center for Collaborative Research, University of Tokyo, Tokyo 153-0041, Japan
kuboyama@ccr.u-tokyo.ac.jp

Abstract. We present a method for discovering new knowledge from structural data which are represented by graphs in the framework of inductive logic programming. A graph, or network, is widely used for representing relations between various data and expressing a small and easily understandable hypothesis. Formal Graph System (FGS) is a kind of logic programming system which directly deals with graphs just like first order terms. By employing refutably inductive inference algorithms and graph algorithmic techniques, we are developing a knowledge discovery system KD-FGS, which acquires knowledge directly from graph data by using FGS as a knowledge representation language.
In this paper we develop a logical foundation of our knowledge discovery system. A term tree is a pattern which consists of variables and tree-like structures. We give a polynomial-time algorithm for finding a unifier of a term tree and a tree in order to make consistency checks efficiently. Moreover we give experimental results on some graph theoretical notions with the system. The experiments show that the system is useful for finding new knowledge.

1 Introduction

The aim of knowledge discovery is to find a small and easily understandable hypothesis explaining given data. Many machine learning and data mining technologies for discovering knowledge have been proposed in many fields. Especially Inductive Logic Programming (ILP) techniques have been applied to discover knowledge from "real-world" data [4]. A graph is one of the most common abstract structures and is widely used for representing relations between various data. In many "real-world" domains such as vision, pattern recognition and organic chemistry, data are naturally represented by graphs.

Formal Graph System (FGS, [12]) is a kind of logic programming system which uses graphs, called term graphs, instead of terms in first-order logic. FGS

S. Dzeroski and P. Flach (Eds.): ILP-99, LNAI 1634, pp. 222–233, 1999.

can represent naturally logical knowledge explaining data represented by graphs. When we try to discover new knowledge from given data, we can not assume that a given hypothesis space contains a hypothesis explaining given data from the beginning. Hence, when we know that the hypothesis is not in the hypothesis space, it is necessary to change the hypothesis space to another space. In [9], the method of refutably inductive inference is proposed. If a correct hypothesis dose not exist in a hypothesis space, we can refute the hypothesis space and change it to another one by using this method. Refuting a hypothesis space is a quite important suggestion for us.

With the above motivations, in [8], we implemented a prototype of a knowledge discovery system KD-FGS (see Fig. 1). As inputs, the system receives positive and negative examples of graph data. As an output, the system produces an FGS program which is consistent with the positive and negative examples if such a hypothesis exists. Otherwise, the system refutes the hypothesis space. KD-FGS consists of an FGS interpreter and a refutably inductive inference algorithm of FGS programs. The FGS interpreter is used to check whether a hypothesis is consistent with the given graph data or not. The refutably inductive inference algorithm is a special type of inductive inference algorithm with refutability of hypothesis spaces and is based on [9]. When the hypothesis space is refuted, KD-FGS chooses another hypothesis space and tries to make a discovery in the new hypothesis space. By refuting the hypothesis space, the algorithm gives important suggestions to achieve the goal of knowledge discovery. Thus, KD-FGS is useful for knowledge discovery from graph data.

In this paper, we also consider a restricted term graph g, called a term tree, such that the term graph obtained by applying any substitution θ to g is a tree, where each graph in θ is a tree. A term tree can represent a tree structure which has variables at internal nodes. But we can not represent such a tree structure in the standard representation of a first order term. In [1, 3], a tree pattern was considered, and learning algorithms for tree patterns from queries were presented, where a tree pattern has constants at its internal nodes, but only its leaves may be variables. Since KD-FGS is a system directly dealing with graphs, the running time is long, in general. Especially, KD-FGS must solve the subgraph isomorphism problem, which is NP-complete, in the component of the FGS interpreter. However, a polynomial-time algorithm solving the subgraph isomorphism problem for trees was proposed in [10]. The FGS interpreter must find a unifier of an input graph and a term graph. Since there exists no mgu (most general unifier) of two term trees in general, we can not apply the standard term algorithms to finding a unifier of a term tree and a tree. Then we give a polynomial-time algorithm for finding a unifier of a term tree and a tree by using graph theoretical techniques. By employing this algorithm, if input data have tree structures, KD-FGS may output a hypothesis within a practical time. There are many "real-world" data having tree structures [13]. This algorithm enables the application of KD-FGS for those data.

This paper is organized as follows. In Section 2, we introduce FGS as a new knowledge representation language for graph data. In Section 3, by giving a

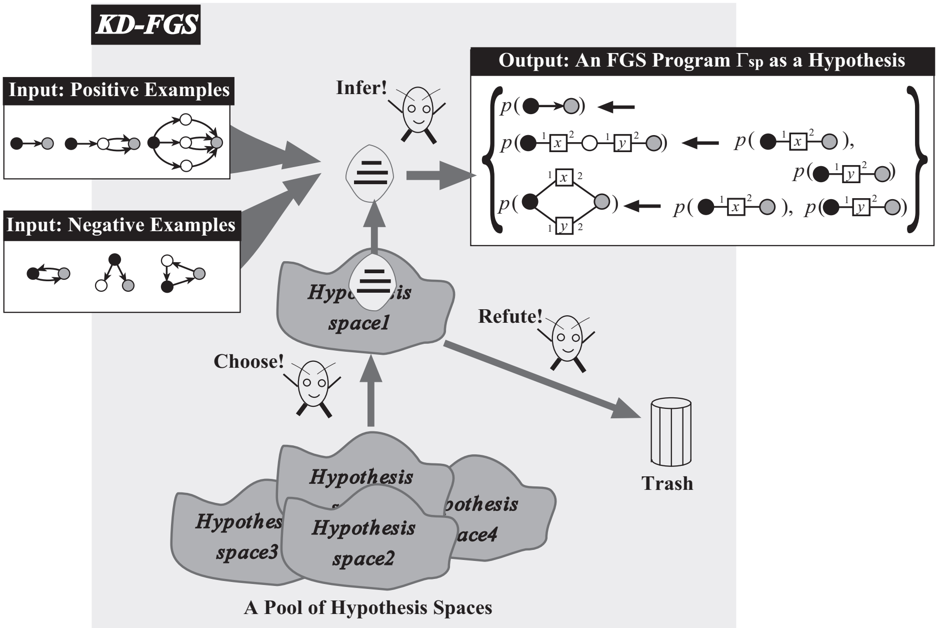


Fig. 1. KD-FGS: a knowledge discovery system from graph data using FGS.

framework of refutably inductive inference of FGS programs, we develop a logical foundation of KD-FGS. In Section 4, we give a polynomial-time algorithm for finding a unifier of a term tree and a tree. In Section 5, we give some examples for graph theoretical notions to our system in order to show the usefulness of our system.

2 FGS as a New Knowledge Representation Language

Formal Graph System (FGS, [12]) is a kind of logic programming system which directly deals with graphs just like first order terms. In [11, 12], we have shown that a class of graphs is generated by a hyperedge replacement grammar (HRG) [5] if and only if it is defined by an FGS of a special form called a regular FGS, and that for a node-label controlled graph grammar (NLC grammar) G introduced in [6], there exists an FGS Γ such that the language generated by G can be definable by Γ. These show that FGS is more powerful than HRG or NLC grammar.

Let Σ and Λ be finite alphabets, and let X be an alphabet, whose element is called a *variable label*. Assume that $(\Sigma \cup \Lambda) \cap X = \emptyset$. A *term graph* $g = (V, E, H)$ consists of a vertex set V, an edge set E and a multi-set H where each element is a list of distinct vertices in V and is called a *variable*. And a term graph g has a vertex labeling $\varphi_g : V \to \Sigma$, an edge labeling $\psi_g : E \to \Lambda$ and a variable

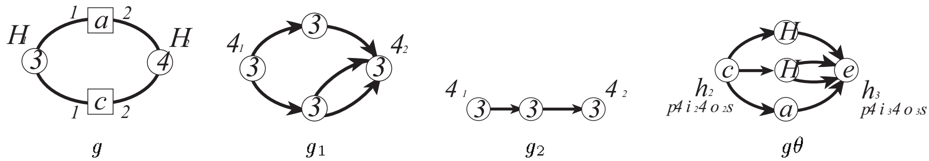


Fig. 2. Term graphs g and $g\theta$ obtained by applying a substitution $\theta = \{x := [g_1, (v_1, v_2)], y := [g_2, (w_1, w_2)]\}$ to g.

labeling $\lambda_g : H \rightarrow X$. A term graph $g = (V, E, H)$ is called *ground* and simply denoted by $g = (V, E)$ if $H = \emptyset$. For example, a term graph $g = (V, E, H)$ is shown in Fig. 2, where $V = \{u_1, u_2\}$, $E = \emptyset$, $H = \{e_1 = (u_1, u_2), e_2 = (u_1, u_2)\}$, $\varphi_g(u_1) = s$, $\varphi_g(u_2) = t$, $\lambda_g(e_1) = x$, and $\lambda_g(e_2) = y$. A variable is represented by a box with lines to its elements and the order of its elements is indicated by the numbers at these lines. An *atom* is an expression of the form $p(g_1, \ldots, g_n)$, where p is a predicate symbol with arity n and $g_1, \ldots, g_n$ are term graphs. Let $A, B_1, \ldots, B_m$ be atoms with $m \geq 0$. Then, a *graph rewriting rule* is a clause of the form $A \leftarrow B_1, \ldots, B_m$. An *FGS program* is a finite set of graph rewriting rules. For example, the FGS program Γ_{SP} in Fig. 1 generates the family of all two-terminal series parallel (TTSP) graphs.

Let g be a term graph and σ be a list of distinct vertices in g. We call the form $x := [g, \sigma]$ a *binding* for a variable label $x \in X$. A *substitution* θ is a finite collection of bindings $\{x_1 := [g_1, \sigma_1], \ldots, x_n := [g_n, \sigma_n]\}$, where x_i's are mutually distinct variable labels in X and each g_i $(1 \leq i \leq n)$ has no variable labeled with an element in $\{x_1, \ldots, x_n\}$. For a set or a list S, the number of elements in S is denoted by $|S|$. In the same way as logic programming system, we obtain a new term graph f by applying a substitution $\theta = \{x_1 := [g_1, \sigma_1], \cdots, x_n := [g_n, \sigma_n]\}$ to a term graph $g = (V, E, H)$ in the following way. For each binding $x_i := [g_i, \sigma_i] \in \theta$ $(1 \leq i \leq n)$ in parallel, we attach g_i to g by removing the all variables $t_1, \cdots, t_k$ labeled with x_i from H, and by identifying the m-th element t_j^m of t_j and the m-th element σ_i^m of σ_i for each $1 \leq j \leq k$ and each $1 \leq m \leq |t_j| = |\sigma_i|$, respectively. We remark that the label of each vertex t_j^m of g is used for the resulting term graph which is denoted by $g\theta$. Namely, the label of σ_i^m is ignored in $g\theta$. In Fig. 2, for example, we draw the term graph $g\theta$ which is obtained by applying a substitution $\theta = \{x := [g_1, (v_1, v_2)], y := [g_2, (w_1, w_2)]\}$ to the term graph g. A *unifier* of two term graphs g_1 and g_2 is a substitution θ such that $g_1\theta$ and $g_2\theta$ are isomorphic. In general, there exists no mgu (most general unifier) of two term graphs. Therefore, in FGS a derivation is based on an enumeration of unifiers and only ground goal is considered in this paper. A graph rewriting rule C is *provable from* an FGS program Γ if C is obtained from Γ by finitely many applications of graph rewriting rules and modus ponens. An FGS interpreter as a component of KD-FGS is used to check whether a hypothesis, which is an FGS program, is consistent with the given graph data or not.

3 Refutably Inductive Inference of FGS Programs

In this section we introduce refutably inductive inference of FGS programs. And we give two interesting hypothesis spaces of FGS programs, weakly reducing and size-bounded FGS programs, which are refutably inferable. Moreover, we present refutably inductive inference algorithms for the hypothesis spaces. We give our framework of refutably inductive inference of FGS programs according to [2, 9, 14]. Mukouchi and Arikawa [9] originated a computational learning theory of machine discovery from facts. They showed that refutably inductive inference is essential in machine discovery from facts and the sufficiently large hypothesis spaces for language learning are refutably inferable.

We give our hypothesis spaces of FGS programs. Let $g = (V, E, H)$ be a term graph. Then we denote the *size* of g by $|g|$ and define $|g| = |V| + |E| + |H|$. For example, $|g| = |V|+|E|+|H| = 2+0+2 = 4$ for the term graph $g = (V, E, H)$ in Fig. 2. For an atom $p(g_1, \ldots, g_n)$, we define $||p(g_1, \ldots, g_n)|| = |g_1|+\cdots+|g_n|$. An *erasing binding* is a binding $x := [g, \sigma]$ such that g consists of all vertices in σ, no edge and no variable. An *erasing substitution* is a substitution which contains an erasing binding. In this paper, we disallow an erasing substitution. Then $||g\theta|| \geq ||g||$ for any term graph g and any substitution θ (Size Non-decreasing Property). A graph rewriting rule $A \leftarrow B_1, \ldots, B_m$ is said to be *weakly reducing* (resp., *size-bounded*) if $||A\theta|| \geq ||B_i\theta||$ for any $i = 1, \ldots, m$ and any substitution θ (resp., $||A\theta|| \geq ||B_1\theta|| + \cdots + ||B_m\theta||$ for any substitution θ). An FGS program Γ is *weakly-reducing* (resp., *size-bounded*) if every graph rewriting rule in Γ is weakly reducing (resp., size-bounded). A size-bounded FGS program is also weakly reducing. For example, the FGS program Γ_{SP} in Fig. 1 is weakly reducing but not size-bounded. Let $g = (V, E, H)$ be a term graph. For a variable label $x \in X$, the number of variables in H labeled with x is denoted by $o(x, g)$. For example, $o(x, g) = 1$ and $o(y, g) = 1$ for the term graph $g = (V, E, H)$ in Fig. 2. For an atom $p(g_1, \ldots, g_n)$ and a variable label $x \in X$, we define $o(x, p(g_1, \ldots, g_n)) = o(x, g_1) + \cdots + o(x, g_n)$.

We consider the two properties of hypothesis spaces for machine discovery from facts. Firstly, the hypothesis space for machine discovery must be recursively enumerable. Secondly, whether a hypothesis is consistent with examples or not must be recursively decidable. The following Lemma 1 and 2 show that our target hypothesis spaces have the first and second properties, respectively. The proofs of Lemma 1 and 2 are based on [2, 14]. In case a hypothesis space dose not have Size Non-decreasing Property, Lemma 1 does not hold. The set of all ground atoms with ground term graphs as arguments is called the Herbrand base and denoted by HB. For an FGS program Γ, M_Γ denotes the least Herbrand model of Γ.

Lemma 1. *A graph rewriting rule $A \leftarrow B_1, \ldots, B_m$ is weakly reducing (resp., size-bounded) if and only if $||A|| \geq ||B_i||$ and $o(x, A) \geq o(x, B_i)$ for any $i = 1, \ldots, m$ and any variable label x (resp., $||A|| \geq ||B_1||+\cdots+||B_m||$ and $o(x, A) \geq o(x, B_1) + \cdots + o(x, B_m)$ for any variable label x).*

Lemma 2. *Let Γ be a weakly reducing or size-bounded FGS program. Then the least Herbrand model M_Γ of Γ is a recursively decidable set.*

We explain the refutably inductive inference of FGS programs. Let Π be a finite set of predicate symbols. For an atom A, pred(A) denotes the predicate symbol of A. For a set $\Pi_0 \subseteq \Pi$ and a set S of atoms, $S \mid_{\Pi_0}$ denotes the set of all atoms in S whose predicate symbols are in Π_0. That is $S \mid_{\Pi_0} = \{A \in S \mid pred(A) \in \Pi_0\}$. A *predicate-restricted complete presentation* of a set $I \subseteq HB$ w.r.t. $\Pi_0 \subseteq \Pi$ is an infinite sequence $(A_1, t_1), (A_2, t_2), \ldots$ of elements in $HB \mid_{\Pi_0} \times \{+, -\}$ such that $\{A_i \mid t_i = +,\ i \geq 1\} = I \mid_{\Pi_0}$ and $\{A_i \mid t_i = -,\ i \geq 1\} = HB \mid_{\Pi_0} \setminus I \mid_{\Pi_0}$. A *refutably inductive inference algorithm* (*RIIA*) is a special type of algorithm that receives a predicate-restricted complete presentation as an input. An RIIA $\mathcal{A}$ is said to *refute* a hypothesis space, if $\mathcal{A}$ produces the sign"refute"as an output and stops. An RIIA either produces infinitely many FGS programs as outputs or refutes a hypothesis space. For an RIIA $\mathcal{A}$ and a presentation δ, $\mathcal{A}(\delta[n])$ denotes the last output produced by $\mathcal{A}$ which is successively presented the first n elements in δ. An RIIA $\mathcal{A}$ is said to *converge* to an FGS program Γ for a presentation δ, if there is a positive integer m_0 such that for any $m \geq m_0$, $\mathcal{A}(\delta[m])$ is defined and equal to Γ. Let $\mathcal{HS}$ be a hypothesis space of FGS programs. For an FGS program $\Gamma \in \mathcal{HS}$ and a predicate-restricted complete presentation δ of M_Γ w.r.t. $\Pi_0 \subseteq \Pi$, an RIIA $\mathcal{A}$ is said to be *infer* the FGS program Γ w.r.t. $\mathcal{HS}$ *in the limit from* δ, if $\mathcal{A}$ converges to an FGS program $\Gamma' \in \mathcal{HS}$ with $M_{\Gamma'} \mid_{\Pi_0} = M_\Gamma \mid_{\Pi_0}$ for δ.

A hypothesis space $\mathcal{HS}$ is said to be *theoretical-term-freely and refutably inferable from complete data*, if for any nonempty finite subset Π_0 of Π, there is an RIIA $\mathcal{A}$ which satisfies the following condition: For any set $I \subseteq HB$ and any predicate-restricted complete presentation δ of I w.r.t. Π_0, (i) if there is an FGS program $\Gamma \in \mathcal{HS}$ such that $M_\Gamma \mid_{\Pi_0} = I \mid_{\Pi_0}$, then $\mathcal{A}$ infers Γ w.r.t. $\mathcal{HS}$ in the limit from δ, (ii) otherwise $\mathcal{A}$ refutes the hypothesis space $\mathcal{HS}$ from δ.

Theoretical terms are supllementary predicates that are necessary for defining some goal predicates. In the above definition, the phrase "theoretical-term-freely inferable" means that using only facts on the goal predicates an RIIA can generates some suppllementary predicates. $\mathcal{WR}^{[\leq n]}$ (resp., $\mathcal{SB}^{[\leq n]}$) denotes the set of all weakly reducing (resp., size-bounded) FGS programs with at most n graph rewriting rules. There are many FGS programs which have the same least Herbrand model. We can assume a canonical form of such FGS programs by fixing predicate symbols in $\Pi \setminus \Pi_0$ and variable labels. $\mathcal{CWR}^{[m]}[\Pi_0]$ denotes the set of all such canonical weakly reducing FGS programs with just m graph rewriting rules. We define $\mathcal{CWR}^{[m]}[\Pi_0](s) = \{\Gamma \in \mathcal{CWR}^{[m]}[\Pi_0] \mid$ the head's size of each rule of Γ is not greater than $s\}$. The proof of Theorem 1 is based on [9].

Theorem 1. *For any $n \geq 1$, the hypothesis space $\mathcal{WR}^{[\leq n]}$ (resp., $\mathcal{SB}^{[\leq n]}$) of all weakly reducing (resp., size-bounded) FGS programs with at most n graph rewriting rules has infinitely many hypotheses. And $\mathcal{WR}^{[\leq n]}$ (resp., $\mathcal{SB}^{[\leq n]}$) is theoretical-term-freely and refutably inferable from complete data.*

```
procedure RIIA_WR(integer n, set of predicate symbols Π_0 ⊆ Π);
begin
    T := ∅; F := ∅;
    read_store(T, F);
    while T = ∅ do begin
        output the empty FGS program;
        read_store(T, F);
    end;
    T_0 := T; F_0 := F;
    for m = 1 to n do begin
        s_m := max{||A|| | A ∈ T_{m-1}};
        recursively generate CWR^[m][Π_0](s_m), and set it to S;
        for each Γ ∈ S do
            while (T, F) is consistent with M_Γ do begin
                output Γ;
                read_store(T, F);
            end;
        T_m := T; F_m := F;
    end;
    output "refute" and stop;
end;

procedure read_store(T, F);
begin
    read the next fact (w, t);
    if t =' +' then T := T ∪ {w} else F := F ∪ {w};
end.
```

Fig. 3. RIIA_WR: a refutably inductive inference algorithm for the hypothesis space $\mathcal{WR}^{[\leq n]}$ of all weakly reducing FGS programs with at most n graph rewriting rules.

Proof. (Sketch of proof) We feed a predicate-restricted complete presentation of a set $I \subseteq HB$ w.r.t. Π_0 to the procedure RIIA_WR in Fig. 3. (i) In case there is an FGS program $\Gamma \in \mathcal{WR}^{[\leq n]}$ such that $M_\Gamma \mid_{\Pi_0} = I \mid_{\Pi_0}$. It follows by Size Non-decreasing Property that a graph rewriting rule whose head has greater size than a ground atom A is not used to derive the atom A. Thus, in the procedure, for any $0 \leq m \leq n$, if T_m and F_m are defined, then $M(\Gamma) \mid_{\Pi_0}$ is not consistent with T_m and F_m for any $\Gamma \in \mathcal{CWR}^{[m]}[\Pi_0]$. Therefore T_n and F_n are never defined and the procedure never terminates the first or second while-loop. (ii) Otherwise. For any $1 \leq m \leq n$, all FGS programs in $\mathcal{CWR}^{[m]}[\Pi_0](s_m)$ are discarded.

By simple enumeration of hypotheses, the hypothesis spaces $\mathcal{WR}^{[\leq n]}$ and $\mathcal{SB}^{[\leq n]}$ are inferable but not refutably inferable. If the number of graph rewriting rules is not bounded by a constant, then these hypothesis spaces are not refutably inferable. We can construct a machine discovery system for a refutably inferable hypothesis space. Thus Theorem 1 gives a theoretical foundation of KD-FGS.

procedure Unification(regular term tree t_1, tree T_2);
begin
Let r_1 be one of leaves of t_1;
Construct the set of all labeling rules R_{r_1};
foreach leaf r_2 of T_2 **do begin**
Label each leaf of T_2 except r_2 with the set of all leaves of T_1 except r_1;
while there exists a vertex v of T_2
such that v is not labeled and all children of v are labeled
do Labeling(v, R_{r_1});
if the label of r_2 includes r_1 **then** t_1 and T_2 are unifiable **and exit**
end;
t_1 and T_2 are not unifiable
end.

Fig. 4. Unification: an algorithm for deciding whether t_1 and T_2 are unifiable or not.

4 An Efficient Algorithm for Finding a Unifier of a Term Tree and a Tree

In this section, we give a polynomial-time algorithm for finding a unifier of a term tree and a tree in order to achieve speedup of KD-FGS.

A term graph g is called a *term tree* if each variable in g is a list of two distinct vertices and, for any substitution $\theta = \{x_1 := [g_1, \sigma_1], \cdots, x_n := [g_n, \sigma_n]\}$ such that each term graph g_i is a tree, $g\theta$ is also a tree. A term tree g is called *regular* if each variable label in g occurs exactly once [7]. For example, a term tree $g = (\{r, s, t, u, v, w\}, \{\{r, s\}, \{u, v\}\}, \{(s, t), (s, u), (u, w)\})$ is shown in Fig. 6. As stated in the section 2, in general, there exists no mgu of two regular term trees. Therefore, even if the input data for KD-FGS is restricted to trees, a derivation in FGS is based on an enumeration of unifiers and only ground goal is considered. From a simple observation we can show that the FGS interpreter must solve the subgraph isomorphism problem, which is NP-complete. For certain special subclasses of graphs, the subgraph isomorphism problem is efficiently solvable [10]. But we should note that even if a subclass of graphs has an efficient algorithm for the subgraph isomorphism problem, we can not construct a unification algorithm straightforwardly from the algorithm.

In this section, we assume that a tree which is an input to our unification algorithm is an unrooted tree without a vertex label and an edge label, since we can easily construct a unification algorithm for a tree having a vertex label and an edge label. Let $t_1 = (V_1, E_1, H_1)$ and $T_2 = (V_2, E_2)$ be a regular term tree and a tree, respectively. Then, we give the algorithm Unification (Fig. 4) for finding a unifier of a regular term tree and a tree. First we specify one of leaves of t_1. Let the leaf be r_1. We define the rooted tree T_1 as $T_1 = (V_1, E_1 \cup \{\{u_1, u_2\} \mid (u_1, u_2) \in H_1 \text{ or } (u_2, u_1) \in H_1\})$ with the root r_1. For a vertex $u \in V_1$, let $w_1, \cdots, w_k$ be all children of u in T_1 such that each $\{u, w_i\}$ is an edge in t_1 for $i = 1, \ldots, k$ and let $w_{k+1}, \cdots, w_m$ be all children of u in T_1 such that either (u, w_i) or (w_i, u) is

procedure Labeling(vertex $v \in V_2$, set of labeling rules R);
begin
    $L := \emptyset$;
    Let d be the number of children of v and $L_1, \cdots, L_d$ be labels of the children;
    foreach $u \leftarrow w_1, \cdots, w_d$ in R **do begin**
        Let $E := \{\{w_i, L_j\} \mid w_i \in L_j (1 \le i \le d, 1 \le j \le d)\}$
        if there is a perfect matching
            for the bipartite graph $(\{w_1, \cdots, w_d\}, \{L_1, \cdots, L_d\}, E)$
        then $L := L \cup \{u\}$
    end;
    foreach $u \Leftarrow w_1, \cdots, w_k, (w_{k+1}), \cdots, (w_m)$ in R with $m \le d$ **do begin**
        Let $E_1 := \{\{w_i, L_j\} \mid w_i \in L_j \quad (1 \le i \le k, 1 \le j \le d)\}$,
        $E_2 := \{\{w_i, L_j\} \mid w_i \in L_j$ or $(w_i) \in L_j \quad (k+1 \le i \le m, 1 \le j \le d)\}$ and
        if for the bipartite graph $(\{w_1, \cdots, w_m\}, \{L_1, \cdots, L_d\}, E_1 \cup E_2)$
            there is a maximum matching which contains all vertices $w_1, \cdots, w_m$
        then $L := L \cup \{u\}$
    end;
    foreach $(w) \Leftarrow (w)$ in R **do begin**
        if there is a set among $L_1, \cdots, L_d$ which includes w or (w) **then**
            $L := L \cup \{(w)\}$
    end;
    Label v with L
end.

Fig. 5. Labeling: a procedure for labeling a vertex in T_2 with a set of vertices in t_1.

a variable in t_1 for $i = k+1, \ldots, m$. We let v be the parent of u in T_1 if u is not a root of T_1. We define labeling rules for u as follows: If there is no variable which has u as a its element, i.e. $k = m$ and both (v, u) and (u, v) are not variables in t_1, then we simply add the following rule to the set of labeling rules:

$$u \leftarrow w_1, w_2, \cdots, w_m.$$

If $k = m$ but either (v, u) or (u, v) is a variable in t_1, we add the following rule:

$$u \Leftarrow w_1, w_2, \cdots, w_m.$$

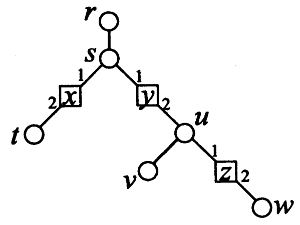
Otherwise, we add the following rules:

$$u \Leftarrow w_1, \cdots, w_k, (w_{k+1}), \cdots, (w_m)$$

and for $k+1 \le i \le m$,

$$(w_i) \Leftarrow (w_i).$$

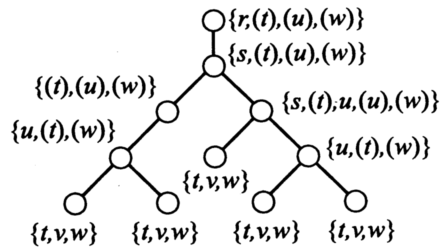
Let R_{r_1} be the set of all labeling rules obtained by applying the above process to all vertices in V_1. We specify one of leaves r_2 of T_2 and consider T_2 as the rooted tree with root r_2. Then, we label all vertices of T_2 with sets of vertices of T_1 using the procedure Labeling (Fig. 5). First we label each leaf of T_2 except r_2 with the set of all leaves of T_1 except r_1. For each vertex u in T_2 such that u itself is not labeled yet but all children of u have been already labeled, we repeat the procedure Labeling until r_2 is labeled. After the procedure Labeling for a vertex $v \in V_2$ terminates , if v has u as an element of the label of v, it shows that v possibly corresponds to u. If v has (u) as an element of the label



Term Tree g

$r \leftarrow s$
$s \Leftarrow (t), (u)$
$u \Leftarrow v, (w)$
$(t) \Leftarrow (t)$
$(u) \Leftarrow (u)$
$(w) \Leftarrow (w)$

Labeling Rules R_r



Tree T

Fig. 6. An example: the labeling rule constructed from a term tree g and the labels of a tree T after the Unification algorithm terminates.

of v, it shows that v possibly corresponds to u or v has a descendant which possibly corresponds to u. In the procedure Labeling, a labeling rule of the form $u \leftarrow w_1, w_2, \cdots, w_m$ can be applied to $v \in V_2$ only when v has exactly m children $c_1, c_2, \cdots, c_m$ such that each child c_i has w_{ℓ_i} as an element of the label of c_i for $i = 1, \ldots, m$, where $\{w_{\ell_1}, w_{\ell_2}, \cdots, w_{\ell_m}\} = \{w_1, w_2, \cdots, w_m\}$. On the other hand, $u \Leftarrow w_1, \cdots, w_k, (w_{k+1}), \cdots, (w_m)$ can be applied to $v \in V_2$ when v has at least m children $c_1, \cdots, c_k, c_{k+1}, \cdots, c_m$ such that for $i = 1, \ldots, k$, c_i has w_{ℓ_i} as an element of the label of c_i where $\{w_{\ell_1}, w_{\ell_2}, \cdots, w_{\ell_k}\} = \{w_1, w_2, \cdots, w_k\}$ and for $i = k+1, \ldots, m$, c_i has w_{ℓ_i} or (w_{ℓ_i}) as an element of the label of c_i where $\{w_{\ell_{k+1}}, \cdots, w_{\ell_m}\} = \{w_{k+1}, \cdots, w_m\}$. The rules of the form $(w) \Leftarrow (w)$ are used to define the descendant relation. If r_2 is labeled with a set including r_1, the Unification algorithm reports the fact that there is a unifier of t_1 and T_2, and terminates. Otherwise, the Unification algorithm applies the above process to the other leaves of T_2. In Fig. 6, for example, we give the labeling rules R_r constructing in the Unification algorithm and show the label assigned to each vertex of T in the Labeling procedure when the term tree g and the tree T shown in Fig. 6 are given as inputs.

If the algorithm declares that t_1 and T_2 are unifiable, we can easily find a unifier from labels of T_2. Since the number of vertices contained in each label is $O(|V_1|)$, we show the following theorem:

Table 1. Experimental results on the KD-FGS system.

No.	Examples	Hypothesis Space	Result
1	TTSP graph	weakly reducing, #atom$\leq$ 2, #rule$\leq$ 2	refute
2		weakly reducing, #atom$\leq$ 2, #rule$\leq$ 3	infer
3		size-bounded, #atom$\leq$ 6, #rule$\leq$ 2	refute
4		size-bounded, #atom$\leq$ 6, #rule$\leq$ 3	refute
5	undirected tree	weakly reducing, #atom$\leq$ 1, #rule$\leq$ 2	refute
6		weakly reducing, #atom$\leq$ 1, #rule$\leq$ 3	infer
7		size-bounded, #atom$\leq$ 6, #rule$\leq$ 2	refute
8		size-bounded, #atom$\leq$ 6, #rule$\leq$ 3	infer

Theorem 2. *A unifier of a regular term tree and a tree can be found in polynomial time.*

5 Experimental Results: Obtaining Some New Knowledge about Graph Theoretical Notions

In order to show that the KD-FGS system is useful for knowledge discovery from graph data, we have preparatory experiments of running the system (see Table 1). We give examples for graph theoretical notions to the system and obtain some new knowledge about representability in FGS programs. For example, in Exp. 2 and 4, input data are positive and negative examples of TTSP graphs (see Fig. 1). In Exp. 2 (resp., 4), the hypothesis space $\mathcal{C}_2$ (resp., $\mathcal{C}_4$) is the set of all restricted weakly reducing (resp., size-bounded) FGS programs with at most 2 (resp., 6) atoms in each body and at most 3 (resp., 3) rules in each program. After the system receives some positive and negative examples, it infers a correct FGS program in $\mathcal{C}_2$ for TTSP graphs in Exp. 2 (resp., it refutes $\mathcal{C}_4$ in Exp. 4). No one knows whether there exists a size-bounded FGS program for TTSP graphs. So we have interests in the experiment of finding such an FGS program. The new results of inferring an FGS program or refuting a hypothesis space are new knowledge about graph theoretical notions. Thus, we confirm that the system is useful for knowledge discovery from graph data.

6 Concluding Remarks

We have given a logical foundation for discovering new knowledge from graph data by employing a refutably inductive inference algorithm, which is one of ILP methods. And we have presented a polynomial-time algorithm for finding a unifier of a term tree and a tree. This algorithm leads us to discover new knowledge from "real-world" data having tree structures.

In order to apply our system to huge "real-world" data, we must achieve practical speedup of the KD-FGS system. We are implementing another FGS interpreter, which is based on a bottom-up theorem proving method, in a parallel logic programming language KLIC.

Acknowledgments

We are grateful to anonymous referees for their valuable comments. The first author would like to thank Akihiro Yamamoto and Hiroki Arimura for their helpful suggestions. This work is partly supported by Grant-in-Aid for Scientific Research No.09780356 from the Ministry of Education, Science, Sports and Culture, Japan and Grant for Special Academic Research No.9870 from Hiroshima City University.

References

1. T. R. Amoth, P. Cull, and P. Tadepalli. Exact learning of tree patterns from queries and counterexamples. *Proc. COLT-98, ACM Press*, pages 175–186, 1998.
2. S. Arikawa, T. Shinohara, and A. Yamamoto. Learning elementary formal systems. *Theoretical Computer Science*, 95:97–113, 1992.
3. H. Arimura, H. Ishizaka, and T. Shinohara. Learning unions of tree patterns using queries. *Proc. ALT-95, Springer-Verlag, LNAI 997*, pages 66–79, 1995.
4. S. Džeroski, N. Jacobs, M. Molina, C. Moure, S. Muggleton, and W. V. Laer. Detecting traffic problems with ILP. *Proc. ILP-98, Springer-Verlag, LNAI 1446*, pages 281–290, 1998.
5. A. Habel and H.-J. Kreowski. May we introduce to you: hyperedge replacement. *Proc. 3rd Graph-Grammars and Their Application to Computer Science, Springer-Verlag, LNCS 291*, pages 15–26, 1987.
6. D. Janssens and G. Rozenberg. On the structure of node-label-controlled graph languages. *Information Sciences*, 20:191–216, 1980.
7. S. Matsumoto, Y. Hayashi, and T. Shoudai. Polynomial time inductive inference of regular term tree languages from positive dat. *Proc. ALT-97, Springer-Verlag, LNAI 1316*, pages 212–227, 1997.
8. T. Miyahara, T. Uchida, T. Kuboyama, T. Yamamoto, K. Takahashi, and H. Ueda. KD-FGS: a knowledge discovery system from graph data using formal graph system. *Proc. PAKDD-99, Springer-Verlag, LNAI 1574, (to appear)*, 1999.
9. Y. Mukouchi and S. Arikawa. Towards a mathematical theory of machine discovery from facts. *Theoretical Computer Science*, 137:53–84, 1995.
10. S. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal on Computing*, 6(4):730–732, 1977.
11. T. Uchida, T. Miyahara, and Y. Nakamura. Formal graph systems and node-label controlled graph grammars. *Proc. 41th Inst. Syst. Control and Inf. Eng.*, pages 105–106, 1997.
12. T. Uchida, T. Shoudai, and S. Miyano. Parallel algorithm for refutation tree problem on formal graph systems. *IEICE Trans. Inf. Syst.*, E78-D(2):99–112, 1995.
13. J. T.-L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Trans. on Knowledge and Data Engineering*, 6(4):559–571, 1994.
14. A. Yamamoto. Procedural semantics and negative information of elementary formal system. *Journal of Logic Programming*, 13:89–98, 1992.