Learning Logic Programs with Unary Partial Function Graph Background Knowledge (extended abstract)

Tamás Horváth[1⋆], Robert H. Sloan[2], and György Turán[3]

[1] Institute of Computer Science III, University of Bonn and Fraunhofer Institute for Autonomous intelligent Systems, tamas.horvath@ais.fhg.de
[2] Department of Computer Science, University of Illinois at Chicago, sloan@uic.edu
[3] Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago and Research Group on Artificial Intelligence, Hungarian Academy of Sciences, Szeged, gyt@uic.edu

Abstract. The product homomorphism method is a combinatorial tool that can be used to develop polynomial PAC-learning algorithms in predicate logic. Using the product homomorphism method, we show that a single nonrecursive definite Horn clause is polynomially PAC-learnable if the background knowledge is a function-free extensional database over a single binary predicate and the ground atoms in the background knowledge form a unary partial function. That is, the background knowledge corresponds to a directed graph, where each node has outdegree at most 1. The proof is based on a detailed analysis of products and homomorphisms of the class of digraphs corresponding to unary partial functions.

1 Introduction

Attribute-value languages are often not suitable for representing complex real-world machine learning problems. Therefore, one of the research challenges in machine learning is to study learning in other representation languages. Among such approaches, inductive logic programming (ILP) [8] is concerned with learning in predicate logic, in particular, with learning logic programs.

The general ILP learning problem is computationally intractable. Therefore, one of the challenging problems in ILP is to show positive and negative theoretical results about the efficient learnability of different fragments of predicate logic in the formal models of computational learning theory. Most of such positive results have been obtained by restricting the hypothesis language. In particular, the most frequently used restrictions are *determinateness* and learning with *constant depth bound* [5].

In contrast to these approaches, in this work we present a positive learnability result by restricting the background knowledge. We assume that there is a single binary background predicate R, and that the ground R-atoms in the background knowledge represent a unary partial function. This structural assumption implies that the background knowledge corresponds to a digraph where each vertex has outdegree at most

⋆ Partially supported by the DFG project (WR 40/2-1) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume.*

1. To prove polynomial learnability for such family of learning problems in the PAC model of learning [9], we use the product homomorphism method [4], a general combinatorial method specific to deriving polynomial learning algorithms in predicate logic. The method is based on finding a combinatorial characterization for the existence of a certain homomorphism from products of relational structures. From the structural assumption on the background knowledge it follows that we have to study products and homomorphisms related to unary partial function graphs.

Using the product homomorphism method, we obtained positive PAC result for the cases when the ground atoms in the background knowledge form a forest [4] or a unary function graph [3] (i.e., when each vertex has outdegree 1). The result of this paper generalizes these results, as unary partial function graphs include both cases; a connected component of a unary partial function graph is always either a tree or a function graph consisting of a single connected component. Though the structural difference between unary function and unary partial function graphs may seem to be insignificant, it turns out that the presence of both types of components requires a careful revision of the results in [3].

The paper is organized as follows. In Section 2, we first give the necessary concepts related to unary partial function graphs, and in Section 3 we then formulate our learning problem. In Section 4, we briefly describe the product homomorphism method, and in Section 5, we derive a polynomial PAC-learning algorithm by using the product homomorphism method. Finally, in Section 6, we give some concluding remarks along with some open problems. Due to space limitation, we omit the proofs in this extended abstract.

2 Graphs and unary partial function graphs

We assume the reader is familiar with the basic concepts of graph theory (see, e.g., [2]). Throughout this paper, by graphs we always mean directed graphs. For a graph G, we denote by $V(G)$ (resp. $E(G)$) the set of vertices (resp. edges) of G.

Let G_i be a graph for $1 \leq i \leq t$. The *product* $G = \prod_{i=1}^{t} G_i$ is a graph with $V(G) = \prod_{i=1}^{t} V(G_i)$ such that for all $\vec{u} = (u_1, \ldots, u_t)$, $\vec{v} = (v_1, \ldots, v_t) \in V(G)$ it holds that $(\vec{u}, \vec{v}) \in E(G)$ iff $(u_i, v_i) \in E(G_i)$ for every $i = 1, \ldots, t$. The t-th power of G, denoted G^t, is the product of t copies of G.

A *homomorphism* from a graph G_1 to a graph G_2 is a map $\varphi : V(G_1) \to V(G_2)$ such that $(\varphi(u), \varphi(v)) \in E(G_2)$ whenever $(u, v) \in E(G_1)$. We call a homomorphism *singly rooted* if we specify the image of one vertex in $V(G_1)$ in advance, and we call a homomorphism *multiply rooted* if we specify images of multiple vertices in $V(G_1)$ in advance. A homomorphism from G_1 to G_2 mapping u_i to v_i for $i = 1, \ldots, k$ is denoted by

$$G_1 \xrightarrow[\{u_1/v_1, \ldots, u_k/v_k\}]{} G_2 \ .$$

We note that a homomorphism always maps one connected component into one connected component.

2.1 Unary partial function graphs

A graph G is a *unary partial function graph* if every vertex of G has outdegree at most 1. The name is justified by viewing G as a graph representing a *unary partial function* $f : V(G) \hookrightarrow V(G)$ such that $f(u) = v$ iff $(u, v) \in E(G)$ for every $u, v \in V(G)$. As an example, the graph given in Fig. 1 is a unary partial function graph consisting of three connected components. Except a_5, each vertex has outdegree 1.

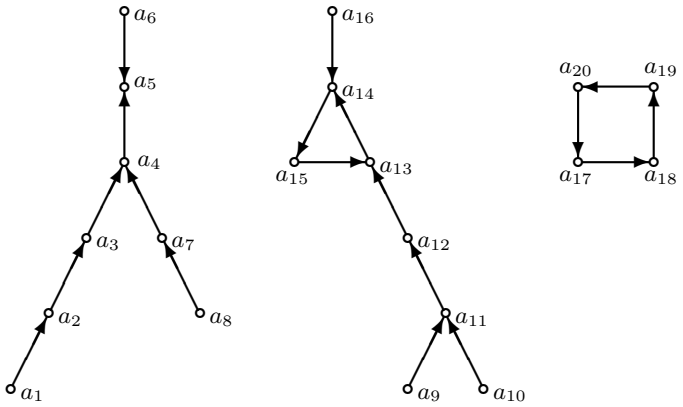

Fig. 1. The directed graph representing the unary function f.

For the rest of this section, let G denote a unary partial function graph consisting of a single connected component. Then it holds that G has at most one vertex of outdegree 0. If G has such a vertex then G is *acyclic*, as in this case it is a directed tree such that the edges are directed towards the root (which is the vertex of outdegree 0). Otherwise (i.e., when each of the vertices of the connected component has outdegree 1) G is *cyclic* and it may be viewed as a directed cycle with directed trees "hanging" from some vertices of the cycle. The edges of the trees are directed towards the cycle. We note that the directed cycle may be a *loop* (i.e., a cycle may have length 1). *Cyclic* vertices are those on the cycle (e.g., a_{15} on Fig. 1). The other vertices are called *noncyclic*.

For a vertex $v \in V(G)$, we denote by $f(v)$ the successor of v. We define $f^{(0)}(v) = v$ and $f^{(k)}(v)$ denotes $f(f^{(k-1)}(v))$ for every $k > 0$. Note that $f^{(k)}(v)$ may be undefined. For instance, for the graph on Fig. 1 it holds that $f^{(2)}(a_3) = a_5$ and $f^{(3)}(a_3)$ is undefined.

Let $k \geq 0$ be an integer. We define $h(f^{(k)}(v))$, the *height* of $f^{(k)}(v)$, by

$$h(f^{(k)}(v)) = \begin{cases} \max\{d : f^{(k)}(v) = f^{(d)}(u) \text{ for some } u \in V(G)\} & \text{if } f^{(k)}(v) \text{ is noncyclic} \\ \infty & \text{if } f^{(k)}(v) \text{ is cyclic} \\ 0 & \text{otherwise .} \end{cases}$$

For the graph on Fig. 1, we have $h(f^{(2)}(a_3)) = h(a_5) = 4$, $h(f^{(3)}(a_3)) = 0$, and $h(a_{15}) = \infty$.

If G is cyclic then $\delta(v)$ denotes the length of the unique directed path connecting v to the cycle, otherwise (i.e., if G is a tree) it denotes the length of the unique directed path connecting v to the root. In both cases, the other endpoint $f^{(\delta(v))}(v)$ of the path is denoted by $r(v)$ and is referred to as the *root* of v. In our example on Fig. 1, $\delta(a_3) = 2$, as $r(a_3) = a_5$, and $\delta(a_{11}) = 2$, as $r(a_{11}) = a_{13}$.

If G is cyclic then we denote by $L_{\text{cycle}}(G)$ the length of its cycle; if G is a tree then $L_{\text{cycle}}(G) = 0$. If v is a vertex of a general unary partial function graph (i.e., one which may consist of more than one connected component) then $L_{\text{cycle}}(v)$ denotes $L_{\text{cycle}}(G')$, where G' is the connected component containing v. In Fig. 1, $L_{\text{cycle}}(a_{11}) = 3$.

Let $u, v \in V(G)$ such that $u = r(u)$ and $v = r(v)$. Then $\sigma(u, v)$ is the smallest nonnegative integer d satisfying $f^{(d)}(u) = v$. Note that by the definition of u and v, both of them are either cyclic or noncyclic. In the first case, $\sigma(u, v)$ denotes the length of the (smallest) directed path leading from u to v on the cycle of G. In the second case, both u and v must be roots of a tree. Since G consists of a single connected component, $u = v$ and hence, $\sigma(u, v) = 0$ always holds for this case.

Now let $u, v \in V(G)$ such that $r(u) = r(v)$. Then there is a unique maximal integer d, $0 \leq d \leq \min\{\delta(u), \delta(v)\}$, such that $f^{(\delta(u)-d)}(u) = f^{(\delta(v)-d)}(v)$. This node is called the *least common ancestor* of u and v, and is denoted by $\text{lca}(u, v)$.

We are ready to define the *distance* between two vertices. Let $u, v \in V(G)$. Then their distance is an ordered pair of nonnegative integers defined by

$$d_{pf}(u, v) = \begin{cases} (d_1, d_2) \text{ such that } f^{(d_1)}(u) = \text{lca}(u, v) = f^{(d_2)}(v) & \text{if } r(u) = r(v) \\ (\delta(u) + \sigma(r(u), r(v)), \delta(v)) & \text{otherwise .} \end{cases}$$

On Fig. 1, $d_{pf}(a_3, a_8) = (1, 2)$, as $\text{lca}(a_3, a_8) = a_4$, and $d_{pf}(a_{11}, a_{16}) = (2 + 1, 1)$.

In the following proposition we formulate some properties of products of unary partial function graphs, that will be used many times in what follows.

Proposition 1. *Let G_i be unary partial function graphs and $b_i \in V(G_i)$ for $i = 1, \ldots, t$. Let $G = \prod_{i=1}^{t} G_i$ be the product of the G_i's and $\vec{b} = (b_1, \ldots, b_t)$ be the product vertex obtained from the b_i's. Then for G and $\vec{b}$ the following properties hold.*

(i) *G is a unary partial function graph.*
(ii) *$\vec{b}$ is cyclic iff all the b_i's are cyclic.*
(iii) *If $\vec{b}$ is cyclic then $L_{cycle}(\vec{b}) = \text{lcm}(L_{cycle}(b_1), \ldots, L_{cycle}(b_t))$, where $\text{lcm}(n_1, \ldots, n_t)$ denotes the least common multiple of $n_1, \ldots, n_t$.*

3 Learning simple logic programs

In this section we define a special class of logic programs [7] that will be discussed from the point of view of learnability.

3.1 Simple logic programs

Throughout this paper we consider (relational) *vocabularies* consisting of a *target predicate* P of arity m, a binary *background predicate* R, and constants $a_1, \ldots, a_n$. Thus, a *term* is either a variable or a constant, and an *atom* is of the form $P(t_1, \ldots, t_m)$ or $R(t_1, t_2)$, where the t's are terms. Depending on its predicate symbol, an atom is said to be a *P-atom* or an *R-atom*. A *literal* is an atom or its negation. An atom is *ground* if it contains no variables. A *basic clause* is a first-order Horn clause of the form $L_0 \leftarrow L_1, \ldots, L_l$ where L_0 is a P-atom and L_i is an R-atom for $i = 1, \ldots, l$. It is also viewed as the set of literals it contains. A *simple logic program* consists of a basic clause and a set $\mathcal{B}$ of ground R-atoms. Since R is binary, the ground R-atoms in $\mathcal{B}$ form a directed graph with vertices $a_1, \ldots, a_n$.

A *substitution* $\theta = \{x_1/t_1, \ldots, x_s/t_s\}$ is a mapping of variables to terms such that $x_i \neq t_i$ for $i = 1, \ldots, s$. Let W be a literal (respectively a clause). Then $W\theta$ is the literal (respectively clause) obtained from W by rewriting simultaneously each variable x_i to t_i in W for $i = 1, \ldots, s$. A clause C *subsumes* a clause D, denoted $C \leq_\theta D$, if there exists a substitution θ such that $C\theta \subseteq D$.

To close this subsection, let C be a basic clause, $\mathcal{B}$ be a set of ground R-atoms, and A be a ground P-atom. We say that C *subsumes A with respect to $\mathcal{B}$*, denoted $C \leq_{\theta,\mathcal{B}} A$, if C subsumes the basic clause $A \leftarrow \mathcal{B}$, i.e., $C \leq_\theta (A \leftarrow \mathcal{B})$. It holds that $C \leq_{\theta,\mathcal{B}} A$ iff A is *implied* by the simple logic program consisting of C and $\mathcal{B}$.

3.2 The learning problem

In this section we give a formal description of the family of learning problems considered in this paper. We assume that the reader is familiar with the basic notions of the PAC-model of learning [9].

Let $\mathcal{B}$ be a set of ground R-atoms. In what follows, $\mathcal{B}$ is referred to as *background knowledge*, and its elements are called *background atoms*. As R is a binary predicate, $\mathcal{B}$ can be viewed as a graph with vertices $a_1, \ldots, a_n$.

The *instance space* of the learning problem is the set of all ground P-atoms. Let C be a basic clause. Then the *concept* $C_\mathcal{B}$ represented by C wrt. $\mathcal{B}$ is the set of ground P-atoms implied by the simple logic program consisting of C and $\mathcal{B}$, i.e.,

$$C_\mathcal{B} = \{A : A \text{ is a ground } P\text{-atom and } C \leq_{\theta,\mathcal{B}} A\} \ .$$

The *concept class* $\mathcal{C}_{\mathcal{B},m}$, corresponding to $\mathcal{B}$ is the family of concepts $C_\mathcal{B}$, where C is a basic clause. (For the next definition, we recall that m is the arity of the target predicate P.) Throughout this paper, we consider the family $\mathcal{F}_{m,n}$ $(m, n > 0)$ of learning problems defined by

$$\mathcal{F}_{m,n} = \{\mathcal{C}_{\mathcal{B},m} : \mathcal{B} \text{ corresponds to a unary partial function graph over } n \text{ vertices}\} \ .$$

That is, a concept class $\mathcal{C}_{\mathcal{B},m}$ belongs to $\mathcal{F}_{m,n}$ iff for every a_i there is at most one a_j such that $R(a_i, a_j) \in \mathcal{B}$ $(1 \leq i, j \leq n)$. The *parameters* measuring the size of a learning problem in $\mathcal{F}_{m,n}$ are m and n.[1]

[1] In Section 5.4, we shall show that the size of the target basic clause as parameter can be omitted by extending the standard representation language.

4 The product homomorphism method

In order to prove polynomial PAC-learnability for $\mathcal{F}_{m,n}$, we shall apply the following basic result [1] from computational learning theory.

Theorem 1. *A family of learning problems is polynomially PAC-learnable if*

(i) the hypothesis finding task can be solved in time polynomial in the parameters,
(ii) the VC-dimension of the concept classes is bounded by a polynomial of the parameters.

According to the first step of the above theorem, we have to show that the hypothesis finding problem for the concept classes in $\mathcal{F}_{m,n}$ can be solved in time polynomial in m and n. More precisely, we consider the following *single clause hypothesis finding problem*: *Given* $\mathcal{C}_{\mathcal{B},m} \in \mathcal{F}_{m,n}$ and disjoint sets E^+ and E^- of ground P-atoms, *find* a basic clause C such that $E^+ \subseteq C_{\mathcal{B}}$ and $E^- \cap C_{\mathcal{B}} = \emptyset$, if such a basic clause exists, and *output* "no", otherwise.

In [4], we have shown that $\mathcal{C}_{\mathcal{B},m}$ is *closed under nonempty intersection* for every $\mathcal{C}_{\mathcal{B},m} \in \mathcal{F}_{m,n}$. That is, for every subset $\mathcal{C} \subseteq \mathcal{C}_{\mathcal{B},m}$ satisfying $\bigcap_{c\in\mathcal{C}} c \neq \emptyset$ it holds that $\bigcap_{c\in\mathcal{C}} c \in \mathcal{C}_{\mathcal{B},m}$. This implies that for a set S of ground P-atoms, the intersection of all concepts containing S, denoted $G_{\mathcal{B}}(S)$, is also a concept in $\mathcal{C}_{\mathcal{B},m}$, i.e., $G_{\mathcal{B}}(S) \in \mathcal{C}_{\mathcal{B},m}$, where

$$G_{\mathcal{B}}(S) = \bigcap\{C_{\mathcal{B}} \in \mathcal{C}_{\mathcal{B},m} : S \subseteq C_{\mathcal{B}}\} \ .$$

In other words, $G_{\mathcal{B}}(S)$, also referred to as the *concept generated by* S, is the smallest concept in $\mathcal{C}_{\mathcal{B},m}$ that contains S. But this means that a consistent clause for the above defined single clause hypothesis finding problem exists iff $G_{\mathcal{B}}(S)$ and E^- are disjoint. Thus, the single clause hypothesis finding problem can be solved by computing first an *efficiently evaluable* basic clause representing the concept $G_{\mathcal{B}}(E^+)$ and then testing whether $G_{\mathcal{B}}(E^+) \cap E^- = \emptyset$ holds.

The following theorem, a special case of the product homomorphism theorem in [4], gives a combinatorial characterization of the concept generated by a set of ground P-atoms.

Theorem 2. *Let* $\mathcal{C}_{\mathcal{B},m} \in \mathcal{F}_{m,n}$, $S = \{P(b_{1,1},\ldots,b_{1,m}),\ldots,P(b_{t,1},\ldots,b_{t,m})\}$ *for some* $t > 0$, *and let* $\vec{b}_j$ *denote* $(b_{1,j},\ldots,b_{t,j})$ *for* $j = 1,\ldots,m$. *Then*

$$G_{\mathcal{B}}(S) = \left\{ P(b_1,\ldots,b_m) : \mathcal{B}^t \xrightarrow[\{\vec{b}_1/b_1,\ldots,\vec{b}_m/b_m,\vec{a}_1/a_1,\ldots,\vec{a}_n/a_n\}]{} \mathcal{B} \right\} \ ,$$

where $\vec{a}_k$ *denotes the (*t*-tuple) product constants* $(a_k,\ldots,a_k)$ *for* $k = 1,\ldots,n$.

Theorem 2 above provides the following method, called the *product homomorphism method* [4], for obtaining a hypothesis finding algorithm for $\mathcal{F}_{m,n}$:

1. Find a combinatorial characterization for the existence of multiply rooted homomorphisms from products of unary partial function graphs to unary partial function graphs.

2. Give an algorithm such that for every $\mathcal{C}_{\mathcal{B},m} \in \mathcal{F}_{m,n}$ and for every set S of ground P-atoms it translates the combinatorial characterization in time polynomial in m, n, and $|S|$ into a basic clause C such that
   - $C_{\mathcal{B}} = G_{\mathcal{B}}(S)$ and
   - C can be evaluated with respect to $\mathcal{B}$ in time polynomial in m and n.

5 Application of the product homomorphism method

Using the product homomorphism method, in this section we derive an efficient PAC algorithm for learning simple logic programs with partial function graph background knowledge.

5.1 Homomorphisms between unary partial function graphs

In order to apply the product homomorphism method to unary partial function graphs, we first need to find necessary and sufficient conditions for the existence of multiply rooted homomorphisms from products of unary partial function graphs into unary partial function graphs. Since unary partial function graphs are closed under product by (i) of Proposition 1, in the next theorems we study rooted homomorphisms between unary partial function graphs. Furthermore, as a homomorphism always maps one connected component into one connected component, it is sufficient to consider unary partial function graphs consisting of a single connected component.

Theorem 3. *Let G_1 and G_2 be unary partial function graphs consisting of a single connected component, let $b_1, \ldots, b_k \in V(G_1)$ be distinct vertices for some $k \geq 1$, and $c_1, \ldots, c_k \in V(G_2)$. Then $G_1 \xrightarrow[\{b_1/c_1,\ldots,b_k/c_k\}]{} G_2$ iff*

(i) $G_1 \xrightarrow[\{b_i/c_i\}]{} G_2$ *for every* $i = 1, \ldots, k$,

(ii) $f^{(d_1)}(c_u) = f^{(d_2)}(c_v)$ *for every* $1 \leq u < v \leq k$, *where* $(d_1, d_2) = d_{pf}(b_u, b_v)$.

Condition (i) of the above theorem indicates that one has to study *singly* rooted homomorphisms between unary partial function graphs. The following theorem gives a necessary and sufficient condition for the existence of a singly rooted homomorphism between unary partial function graphs. We denote by $n_1 \mid n_2$ that n_1 divides n_2.

Theorem 4. *Let G_1 and G_2 be unary partial function graphs consisting of a single connected component and let $b \in V(G_1)$, $c \in V(G_2)$. Then $G_1 \xrightarrow[\{b/c\}]{} G_2$ iff*

(i) G_2 *is cyclic satisfying* $L_{cycle}(G_2) \mid L_{cycle}(G_1)$ *whenever* G_1 *is cyclic,*

(ii) $h(f^{(k)}(c)) \geq h(f^{(k)}(b))$ *for every* $k \geq 0$.

5.2 Products of unary partial function graphs

The product homomorphism method indicates that the learning algorithm must consider the product of t copies of the graph representing the background knowledge, where t is the number of positive examples. However, that product is exponentially large; it contains n^t nodes, where n is the number of constants mentioned in the background knowledge. Therefore we cannot work with this graph explicitly. Instead, we must show that the relevant parameters implied by Theorems 3 and 4, i.e., cycle lengths, heights, and distances between vertices can be computed directly from those of the original graph corresponding to the background knowledge.

For computing cycle lengths, we can directly apply (iii) of Proposition 1. We start by giving a lemma that can be used for determining the height of a product vertex.

Lemma 1. *Let G_i be unary partial function graphs for $i = 1, \ldots, t$, $G = \prod_{i=1}^{t} G_i$, and $\vec{b} = (b_1, \ldots, b_t) \in V(G)$. Then $h(f^{(k)}(\vec{b})) = \min_{i=1,\ldots,t} h(f^{(k)}(b_i))$ for every $k \geq 0$.*

To state Lemma 4 below for computing the distance between two product vertices, in the following lemma we first characterize the distance of a product vertex from its root. Then, in Lemma 3, we give a necessary and sufficient condition for two vertices of the product graph to be in the same connected component.

Lemma 2. *Let G_i be unary partial function graphs, $b_i \in V(G_i)$ for $i = 1, \ldots, t$, and consider the product graph $G = \prod_{i=1}^{t} G_i$ and product vertex $\vec{b} = (b_1, \ldots, b_t) \in V(G)$. Let I denote the set of indices $\{i : 1 \leq i \leq t,\ r(b_i)$ is noncyclic$\}$. Then the distance of $\vec{b}$ from its root $r(\vec{b})$ is given by*

$$\delta(\vec{b}) = \begin{cases} \max_{i=1,\ldots,t} \delta(b_i) & \text{if } I = \emptyset \\ \min_{i \in I} \delta(b_i) & \text{otherwise .} \end{cases}$$

Lemma 3. *Let G_i be unary partial function graphs for $i = 1, \ldots, t$, $G = \prod_{i=1}^{t} G_i$, and $\vec{b} = (b_1, \ldots, b_t)$ $\vec{c} = (c_1, \ldots, c_t) \in V(G)$. Then $\vec{b}$ and $\vec{c}$ are in the same connected component of G iff for every $i = 1, \ldots, t$ it holds that $f^{(\delta(\vec{b})+d)}(b_i) = f^{(\delta(\vec{c}))}(c_i)$, where $d = 0$ if some of the b_i's belongs to a noncyclic connected component; otherwise, d is a nonnegative integer satisfying*

$$d \equiv \sigma(f^{(\delta(\vec{b}))}(b_k), f^{(\delta(\vec{c}))}(c_k)) \pmod{L_{cycle}(b_k)}$$

for every $k = 1, \ldots, t$.

The following lemma follows directly from Lemmas 2 and 3.

Lemma 4. *Let G_i, G, b_i, and c_i, $1 \leq i \leq t$, be defined as in the previous lemma and assume that the product vertices $\vec{b} = (b_1, \ldots, b_t)$ and $\vec{c} = (c_1, \ldots, c_t)$ belong to the same connected component of G. Then*

$$d_{pf}(\vec{b}, \vec{c}) = \begin{cases} (\delta(\vec{b}) - d_1, \delta(\vec{c}) - d_1) & \text{if } r(\vec{b}) = r(\vec{c}) \\ (\delta(\vec{b}) + d_2, \delta(\vec{c})) & \text{otherwise ,} \end{cases}$$

where $d_1 = \max\{d \geq 0 : f^{(\delta(\vec{b})-d)}(b_i) = f^{(\delta(\vec{c})-d)}(c_i)$ *for* $i = 1, \ldots, t\}$ *and* d_2 *is the smallest nonnegative integer satisfying*

$$d_2 \equiv \sigma(f^{(\delta(\vec{b}))}(b_i), f^{(\delta(\vec{c}))}(c_i)) \pmod{l_i}$$

for every $i = 1, \ldots, t$.

Since the congruence system in Lemmas 3 and 4 can be solved efficiently (see, e.g., [6]), one can decide efficiently whether $\vec{b}$ and $\vec{c}$ are in the same connected component, and if so, then their distance $d_{pf}(\vec{b}, \vec{c})$ can be computed in polynomial time.

5.3 A combinatorial characterization of $G_{\mathcal{B}}$

Combining the results of Sections 5.1 and 5.2 with Theorem 2, in this section we give a combinatorial characterization of the concept generated by a set of ground P-atoms wrt. unary partial function graph background knowledge. Let S be the set of ground atoms $\{P(\mathbf{b}_1), \ldots, P(\mathbf{b}_t)\}$, where $\mathbf{b}_i = (b_{i,1}, \ldots, b_{i,m})$ for $i = 1, \ldots, t$ ($t > 1$). Let $\vec{b}_j$ denote the product vertex $(b_{1,j}, \ldots, b_{t,j})$ for $j = 1, \ldots, m$. Let

$$\begin{aligned}
I_{\text{const}}(S) &= \{(j,q) : 1 \leq j \leq m, 1 \leq q \leq n, \text{ and } \vec{b}_j = \vec{a}_q\} \ , \\
I_{\text{var}}(S) &= \{j : 1 \leq j \leq m, \vec{b}_j \neq \vec{a}_q \text{ for some } q, 1 \leq q \leq n\} \ , \\
I_{\text{cyclic}}(S) &= \{j \in I_{\text{var}}(S) : r(\vec{b}_j) \text{ is cyclic}\} \ , \\
H_{\text{pairs}}(S) &= \{(u,v) : u, v \in I_{\text{var}}(S), u < v, \text{ and } \vec{b}_u, \vec{b}_v \text{ are in the same connected component}\} \ , \\
H_{\text{const}}(S) &= \{(j,q) : j \in I_{\text{var}}(S), 1 \leq q \leq n, \text{ and } \vec{b}_j, \vec{a}_q \text{ are in the same connected component}\} \ .
\end{aligned}$$

Theorem 5. *If* $\mathcal{B}$ *is a unary partial function graph then*

$$\begin{aligned}
G_{\mathcal{B}}(S) = \{&P(b_1, \ldots, b_m) : \\
&b_j = a_q \text{ for all } (j,q) \in I_{const}(S), && (1)\\
&f^{(L_{cycle}(\vec{b}_j))}(r(b_j)) = r(b_j) \text{ for all } j \in I_{cyclic}(S), && (2)\\
&h(f^{(k)}(b_j)) \geq h(f^{(k)}(\vec{b}_j)) \text{ for all } k = 0, \ldots, n \text{ and } j \in I_{var}(S), && (3)\\
&f^{(d_1)}(b_u) = f^{(d_2)}(b_v) \text{ for all } (u,v) \in H_{pairs}(S), \text{ where } (d_1, d_2) = d_{pf}(\vec{b}_u, \vec{b}_v), && (4)\\
&f^{(d_1)}(b_j) = f^{(d_2)}(a_q) \text{ for all } (j,q) \in H_{const}(S), \text{ where } (d_1, d_2) = d_{pf}(\vec{b}_j, \vec{a}_q)\} && (5)
\end{aligned}$$

Proof sketch. By Theorem 2, $P(b_1, \ldots, b_m) \in G_{\mathcal{B}}(S)$ iff

$$(\mathcal{B})^t \xrightarrow[\{\vec{b}_1/b_1, \ldots, \vec{b}_m/b_m, \vec{a}_1/a_1, \ldots, \vec{a}_n/a_n\}]{} \mathcal{B} \ . \qquad (6)$$

Thus, it is sufficient to show that (6) holds iff conditions (1) - (5) hold. For the "only if" part, the proof of (1) is automatic, (2), (3) hold by Theorem 4, and (4), (5) by (ii) of Theorem 3. To prove the "if" part, the connected components of $(\mathcal{B})^t$ can be considered separately.

(i) For connected components not containing any non-constant product vertex from $\vec{b}_1, \ldots, \vec{b}_m$, there are projections providing a homomorphism into $\mathcal{B}$ and mapping each occurring product constant $\vec{a}$ to a.

(ii) By Theorems 3 and 4, (2)–(4) provide the required rooted homomorphisms for connected components containing at least one non-constant vertex from $\vec{b}_1, \ldots, \vec{b}_m$.

(iii) Finally, for connected components containing at least one non-constant vertex from $\vec{b}_1, \ldots, \vec{b}_m$ and at least one constant product vertex, (2)–(3) and (5) provide the required rooted homomorphisms by Theorems 3 and 4. □

5.4 Concept representation and polynomial learnability

The last step of the product homomorphism method is to give an algorithm translating the combinatorial characterization given by Theorem 5 into an efficiently evaluable basic clause. From our previous results [4] on the length of product cycles it follows that there are cases when the size of any consistent basic clause is exponential in n, i.e., when $L_{\text{cycle}}(\vec{b}_j)$ in (2) is exponential in n. Thus, the standard representation using only the predicates P and R is not suitable for polynomial learnability if the size of the target concept is not considered as a learning parameter. Therefore, we introduce new predicates of the form $\text{PATH}_d(x, y)$, which hold if there is a path of length d from x to y, for every d. Note that $\text{PATH}_d(x, x)$ holds iff $d = 0$ or $d > 0$ and $L_{\text{cycle}}(x) \mid d$. Using the extended representation language, we are ready to give Algorithm 1 computing a clause that represents $G_{\mathcal{B}}(S)$ for a set S of ground P-atoms and unary partial function graph background knowledge $\mathcal{B}$.

Algorithm 1 UNARYPARTIALFUNCTIONGRAPH

Require: ground set $S = \{P(\mathbf{b}_1), \ldots, P(\mathbf{b}_t)\}$ and a unary partial function graph $\mathcal{B}$
Ensure: clause C such that $C_{\mathcal{B}} = G_{\mathcal{B}}(S)$

1: **let** $C = \{P(t_1, \ldots, t_m)\}$, where $t_j = a_q$ if $(j, q) \in I_{\text{const}}(S)$ for some q, otherwise t_j is the variable $x_{\vec{b}_j}$ for $j = 1, \ldots, m$
2: **for all** $j \in I_{\text{cyclic}}(S)$ **do**
  if $\delta(\vec{b}_j) = 0$ **then** $C = C \cup \{\neg\text{PATH}_{L_{\text{cycle}}(\vec{b}_j)}(t_j, t_j)\}$
  else $C = C \cup \{\neg\text{PATH}_{\delta(\vec{b}_j)}(t_j, y_j), \neg\text{PATH}_{L_{\text{cycle}}(\vec{b}_j)}(y_j, y_j)\}$
3: **for all** $j \in I_{\text{var}}(S)$ and $k = 0, \ldots, \delta(\vec{b}_j)$ **do**
  if $k < h(f^{(k)}(\delta(\vec{b}_j))) < \infty$ **then** $C = C \cup \{\neg\text{PATH}_k(t_j, y'_j), \neg\text{PATH}_{h(f^{(k)}(\vec{b}_j))}(y''_j, y'_j)\}$
4: **for all** $(u, v) \in H_{\text{pairs}}(S)$ **do**
  $C = C \cup \{\neg\text{PATH}_{d_1}(t_u, z_{u,v}), \neg\text{PATH}_{d_2}(t_v, z_{u,v})\}$ where $(d_1, d_2) = d_{pf}(\vec{b}_u, \vec{b}_v)$
5: **for all** $(j, q) \in H_{\text{const}}(S)$ **do**
  $C = C \cup \{\neg\text{PATH}_{d_1}(t_j, z'_{j,q}), \neg\text{PATH}_{d_2}(a_q, z'_{j,q})\}$ where $(d_1, d_2) = d_{pf}(\vec{b}_j, \vec{a}_q)$
6: **return** C

Steps (1)–(5) of Algorithm 1 translate Conditions (1)–(5) of Theorem 5, respectively. We have the following theorem.

Theorem 6. *Algorithm 1 is correct, i.e., $C_{\mathcal{B}} = G_{\mathcal{B}}(S)$, and it is polynomial in m, n, and $|S|$. The size of C (in the extended representation language) is $O\left(m^2 + mn\right)$. C can be evaluated wrt. $\mathcal{B}$ in time polynomial in m and n.*

Example 1. To illustrate Algorithm 1, consider the unary partial function graph G given in Fig. 1. Using the binary background predicate R, let $\mathcal{B}$ be the corresponding background knowledge, i.e., $R(u, v) \in \mathcal{B}$ iff $(u, v) \in E(G)$ for every $u, v \in V(G)$.[2]

Let P be a ternary target predicate and $S = \{P(a_6, a_3, a_{12}), P(a_{11}, a_9, a_{18})\}$. Thus, $\vec{b}_1 = (a_6, a_{11})$, $\vec{b}_2 = (a_3, a_9)$, and $\vec{b}_3 = (a_{12}, a_{18})$. By definition, $I_{\text{const}}(S) = \emptyset$, $I_{\text{var}}(S) = \{1, 2, 3\}$, and by (ii) of Proposition 1, $I_{\text{cyclic}}(S) = \{3\}$. To compute $H_{\text{pairs}}(S)$, we apply Lemma 3 for every (u, v), $u, v \in I_{\text{var}}(S)$. For $(1, 2)$ corresponding to $(\vec{b}_1, \vec{b}_2) = ((a_6, a_{11}), (a_3, a_9))$, we have to check whether

$$f^{(\delta(\vec{b}_1)+d)}(a_6) = f^{(\delta(\vec{b}_2))}(a_3) \text{ and } f^{(\delta(\vec{b}_1)+d)}(a_{11}) = f^{(\delta(\vec{b}_2))}(a_9) \tag{7}$$

hold for $d = 0$. By Lemma 2, we have

$$\begin{aligned}\delta(\vec{b}_1) = \delta((a_6, a_{11})) &= \min_{i \in \{1\}} \delta(b_i) = \delta(a_6) \;\; = 1\\ \delta(\vec{b}_2) = \delta((a_3, a_9)) \; &= \min_{i \in \{1\}} \delta(b_i) = \delta(a_3) = 2 \;\; .\end{aligned}$$

In both equations, $I = \{1\}$, as $r(a_{11})$ and $r(a_9)$ are cyclic. The equations in (7) thus hold by

$$f^{(1)}(a_6) = a_5 = f^{(2)}(a_3) \text{ and } f^{(1)}(a_{11}) = a_{12} = f^{(2)}(a_9)$$

respectively. Hence, $(1, 2) \in H_{\text{pairs}}(S)$. It can be shown in the same way that $H_{\text{pairs}}(S) = \{(1, 2)\}$ and $H_{\text{const}}(S) = \emptyset$.

Now we are ready to illustrate Algorithm 1 step by step on inputs S and $\mathcal{B}$.

Step 1: $C = \{P(x_{\vec{b}_1}, x_{\vec{b}_2}, x_{\vec{b}_3})\}$ because $I_{\text{const}} = \emptyset$.

Step 2: Since $I_{\text{cyclic}}(S) = \{3\}$, we have to compute $\delta(\vec{b}_3)$ and $L_{\text{cycle}}(\vec{b}_3)$. By Lemma 2, $\delta(\vec{b}_3) = 0$, and from (iii) of Proposition 1 we have $L_{\text{cycle}}(\vec{b}_3) = \text{lcm}\{3, 4\} = 12$. Thus, in this step we add the literal $\neg\text{PATH}_{12}(x_{\vec{b}_3}, x_{\vec{b}_3})$ to C.

Step 3: For $j = 1$ and $k = 0$ we first have to compute $h(f^{(0)}(\vec{b}_1))$. By Lemma 1, we have $h(f^{(0)}(\vec{b}_1)) = \min\{h(f^{(0)}(a_6)), h(f^{(0)}(a_{11}))\} = \min\{0, 1\} = 0$. For $j = 1$ and $k = \delta(\vec{b}_1) = 1$, $h(f^{(1)}(\vec{b}_1)) = 1$ by Lemma 1. Since $h(f^{(1)}(\vec{b}_1)) \not> 1$, we add no literals to C. For similar reasons, we add no new literals to C for $j = 2, 3$.

Step 4: As $H_{\text{pairs}} = \{(1, 2)\}$, we have $d_{pf}(\vec{b}_1, \vec{b}_2) = (1, 2)$ by Lemma 4, and add therefore $\neg\text{PATH}_1(x_{\vec{b}_1}, z_{1,2})$, $\neg\text{PATH}_2(x_{\vec{b}_2}, z_{1,2})$ to C.

Step 5: $H_{\text{const}}(S) = \emptyset$ and thus, no literals will be added to C in this step.

Step 6: The algorithm finally returns the clause

$$P(x_{\vec{b}_1}, x_{\vec{b}_2}, x_{\vec{b}_3}) \leftarrow \text{PATH}_{12}(x_{\vec{b}_3}, x_{\vec{b}_3}), \text{PATH}_1(x_{\vec{b}_1}, z_{1,2}), \text{PATH}_2(x_{\vec{b}_2}, z_{1,2}) \;\; .$$

[2] We note that B is obtained from the background knowledge in the running example of [3] by removing edge (a_5, a_4).

Since the VC-dimension of a concept class $\mathcal{C}$ is at most $\log(\mathcal{C})$, from the bound on the size of the target concept in Theorem 6 it follows that the VC-dimension of $\mathcal{C}_{\mathcal{B},m}$ is polynomial in m and n if $\mathcal{B}$ is a unary partial function graph. Thus, by Theorems 1 and 6 we have the following main result of this paper.

Theorem 7. *Using the extended representation language, simple logic programs with $\mathcal{B}$ being a unary partial function graph are efficiently PAC-learnable.*

6 Remarks and open problems

Using the product homomorphism method, we have shown that simple logic programs with unary partial function graph background knowledge are polynomially PAC-learnable in an extended representation language. The importance of this result is that we have not assumed any bound on the size of the target clause, the target clause is not necessarily determinate, and its size may be exponential in the standard representation language.

Finally, we list some interesting open problems for further research. In practical applications, usually there is no consistent hypothesis consisting of a single clause. Since unary partial function graphs generalize forests, from our previous results in [4] it follows that the problem of deciding whether there exists a consistent hypothesis consisting of k clauses is NP-complete for any fixed $k \geq 3$ for unary partial function graph background knowledge. It would be interesting to see whether the optimal solution can be approximated in polynomial time. A further research topic would be to investigate whether the positive PAC result of this paper holds for colored unary partial function graphs, i.e., when the vocabulary is extended by a set of unary background predicates. Finally, it would be interesting to apply the product homomorphism method to other classes of directed graphs, in particular to classes generalizing unary partial function graphs.

References

1. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.
2. R. Diestel. *Graph theory*. Springer-Verlag, New York, 2nd edition, 2000.
3. T. Horváth, R. Sloan, and G. Turán. Learning logic programs by using the product homomorphism method. In *Proc. of the 10th Annual Conference on Computational Learning Theory (COLT-97)*, pages 10–20, New York, 1997. ACM Press.
4. T. Horváth and G. Turán. Learning logic programs with structured background knowledge. *Artificial Intelligence*, 128(1-2):31–97, May 2001.
5. J.-U. Kietz and S. Džeroski. Inductive logic programming and learnability. *SIGART Bulletin*, 5(1):22–32, 1994.
6. D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Second Edition, Addison-Wesley, Reading, 1981.
7. J. W. Lloyd. *Foundations of Logic Programming, Second Edition*. Springer-Verlag, 1987.
8. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
9. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1985.