JMLR: Workshop and Conference Proceedings vol 52, 110-122, 2016 PGM 2016

Probabilistic Graphical Models Specified by Probabilistic Logic Programs: Semantics and Complexity

Fabio Gagliardi Cozman FGCOZMAN@USP.BR
Escola Politécnica, Universidade de São Paulo, São Paulo, Brazil

Denis Deratani Mauá DDM@IME.USP.BR
Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, Brazil

Abstract

We look at probabilistic logic programs as a specification language for probabilistic models, and study their interpretation and complexity. Acyclic programs specify Bayesian networks, and, depending on constraints on logical atoms, their inferential complexity reaches complexity classes #P, #NP, and even #EXP. We also investigate (cyclic) stratified probabilistic logic programs, showing that they have the same complexity as acyclic probabilistic logic programs, and that they can be depicted using chain graphs.

Keywords: Probabilistic logic programming, complexity theory.

1. Introduction

Bayesian networks and Markov random fields can be specified using a variety of languages, ranging from logical ones such as PRISM (Sato and Kameya, 2001) and Markov logic (Richardson and Domingos, 2006), to template languages such as BUGS (Gilks et al., 1993). Among *probabilistic logic programming* languages, the approach started by Poole (1993) and Sato (1995) has been very popular: here a *normal logical program* is enlarged with independent *probabilistic facts*. Any propositional Bayesian network over binary variables can be specified with an *acyclic* probabilistic logic program; conversely, any propositional acyclic program can be interpreted as a Bayesian network (Poole, 1993, 2008). Thus *propositional* acyclic programs behave as Bayesian networks. However, it is suprising that, as we show in this paper, the complexity of acyclic programs goes up the counting hierarchy *even* when we restrict predicates to have bounded arity. And complexity goes up to exponential levels when we do not impose this restriction. This is the first set of contributions in this paper (Section 2).

Of course, probabilistic logic programs need not be acyclic. In fact, Sato's original work on *distribution semantics* did not impose acyclicity, only asked for it to obtain efficiency (Sato and Kameya, 2001). Cyclic programs are attractive as they offer a possible specification strategy for cyclic probabilistic graphical models, a challenging topic with applications for instance in structural equations (Pearl, 2009) and feedback systems (Nodelman et al., 2002). It so happens that there are simple syntactic conditions on probabilistic logic programs, lighter than acyclicity, that guarantee that a program specifies a *unique* probability distribution over atoms. In particular, such uniqueness obtains when the program is *stratified* (Fierens et al., 2014). We show how stratified programs can be depicted using chain graphs, by resorting to *loop formulas* (Lin and Zhao, 2002), and we then show that the complexity of stratified probabilistic logic programs is *not* harder than the complexity of their acyclic counterparts. This is the second set of contributions in this paper (Section 3).

2. Acyclic probabilistic logic programs and Bayesian networks

Section 2.1 collects concepts from logic programming, and Section 2.2 presents general facts about probabilistic logic programs, quickly focusing on acyclic ones. We present in Section 2.3 a brief review of complexity theory; the knowledgeable reader can (and perhaps should!) skim through that material, as the only non-standard concept is "equivalence" for counting classes. We present our results on the complexity of acyclic probabilistic logic programs in Section 2.4

2.1 Normal logic programs: syntax, interpretations, and dependency graphs

Take a fixed vocabulary consisting of logical variables $X, X_1, \ldots$, predicates $\mathsf{r}, \mathsf{r}_r, \ldots$, and constants $a, b, \ldots$. A *term* is a constant or a logical variable; an *atom* is written as $\mathsf{r}(t_1, \ldots, t_n)$, where r is a predicate of arity n and each t_i is a term (a 0-arity atom is written as r). An atom is *ground* if it does not contain logical variables. A *normal logic program* consists of a set of rules written as $A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n$., where the A_i are atoms. The *head* of this rule is A_0; the right-hand side is the *body*. A rule without a body, written simply as A_0., is a *fact*. A *literal* is either A (positive) or $\mathbf{not}A$ (negative), where A is an atom. A program without negation is *definite*, and a program without variables is *propositional*. The *Herbrand base* is the set of all ground atoms built from constants and predicates in a program. We do not consider functions in this paper, to stay with finite Herbrand bases. A *substitution* is a (partial) function that maps logical variables into terms. A *grounding* is a substitution mapping into constants; a rule can be grounded by substitution of all its logical variables. The grounding of a program is the propositional program obtained by applying every possible grounding of a rule using only the constants in the program (i.e., using only ground atoms in the Herbrand base). An *interpretation* is a consistent set of literals such that every atom in the Herbrand base is mentioned. The *dependency graph* of a program is a directed graph where each predicate is a node, and where there is an edge from a node B to a node A if there is a rule where A appears in the head and B appears in the body (if B appears right after $\mathbf{not}$, the edge is *negative*; otherwise, it is *positive*). The *grounded* dependency graph is the dependency graph of the propositional program obtained by grounding. An acyclic program is one with an acyclic grounded dependence graph.

2.2 Probabilistic logic programs, acyclicity, and Bayesian networks

A *probabilistic logic program* consists of a pair $\langle \mathbf{P}, \mathbf{PF} \rangle$, where $\mathbf{P}$ is a normal logic program and $\mathbf{PF}$ is a set of *probabilistic facts*. A probabilistic fact is a fact that does not unify with any rule head, and that is associated with a (rational) probability value. If a probabilistic fact contains logical variables, it is interpreted as a set of probabilistic facts (one per grounding). All ground probabilistic facts are assumed stochastically independent. A truth assignment for all probabilistic facts is a *total choice*; a total choice is represented as a list containing only those atoms that are set to true (the remaining probabilistic facts are set to false by the total choice). We often write A to denote $\{A = \mathsf{true}\}$ and $\neg A$ to denote $\{A = \mathsf{false}\}$. We abbreviate "probabilistic logic program" by PLP, and we use, whenever possible, the syntactic conventions of the ProbLog system[1] (Fierens et al., 2014). The only unusual aspect of this syntax is the use of $\alpha :: A$ to specify that the probability of atom A is α.

1. At https://dtai.cs.kuleuven.be/problog/index.html.

Example 1 *Here is a ProbLog probabilistic logic program:*

$0.7 :: \mathsf{burglary}.$ $0.2 :: \mathsf{earthquake}.$ $0.9 :: \mathsf{a}.$ $0.8 :: \mathsf{b}.$ $0.1 :: \mathsf{c}.$
$\mathsf{neighbor}(1).$ $\mathsf{neighbor}(2).$
$\mathsf{alarm} :- \mathsf{burglary}, \mathsf{earthquake}, \mathsf{a}.$ $\mathsf{alarm} :- \mathsf{burglary}, \mathbf{not}\ \mathsf{earthquake}, \mathsf{b}.$
$\mathsf{alarm} :- \mathbf{not}\ \mathsf{burglary}, \mathsf{earthquake}, \mathsf{c}.$ $\mathsf{calls}(X) :- \mathsf{alarm}, \mathsf{neighbor}(X).$

A truth assignment for some probabilistic facts is $\{\mathsf{burglary} = \mathsf{true}, \mathsf{earthquake} = \mathsf{false}\}$. □

The semantics of a PLP is given by the semantics of its grounding. Consider then a propositional PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$. Due to the assumption of independence, we can obtain the probability of each total choice $\mathbf{C}$ (note that $\mathbf{C}$ can be given simply as a list of atoms) as the product of their individual probabilities: $\mathbb{P}(\mathbf{C}) = \prod_{A \in \mathbf{C}} \mathbb{P}(A) \prod_{A \in \mathbf{PF} \setminus \mathbf{C}} [1 - \mathbb{P}(A)]$, where $\mathbb{P}(A)$ denotes the probability annotating the probabilistic fact $\mathbb{P}(A) :: A$ in the PLP. Note also that for each total choice $\mathbf{C}$, we obtain a logical program $\mathbf{P} \cup \mathbf{C}$. The idea is that, if the truth assignment for all atoms is unique for program $\mathbf{P} \cup \mathbf{C}$, for each $\mathbf{C}$, then the product measure over probabilistic facts induces a unique probability distribution over all atoms. This is precisely what happens when $\mathbf{P}$ is acyclic.

Indeed, an acyclic logic program uniquely specifies the truth assignment over all atoms by its universally adopted semantics (Apt and Bezem, 1991). Namely, the semantics is the *unique* interpretation that satifies the *Clark completion* of $\mathbf{P}$ (Clark, 1978). This completion is a transformation that takes $\mathbf{P}$ and produces a first-order theory; roughly speaking, the transformantion replaces each $:-$ by a logical equivalence $\Leftrightarrow$, and replaces each $\mathbf{not}$ by a classical negation $\neg$; it introduces existential quantification over logical variables in the body and not in the head; it produces a single formula out of the rules that share the head (this is done by introducing a disjuction of the bodies of those rules), and it universally quantifies all remaining logical variables.

So, given an acyclic PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$, for any total choice the program $\mathbf{P} \cup \mathbf{C}$ is acyclic, and the probabilities over probabilistic facts induce a *unique* probability distribution over all atoms. This distribution is the program's semantics. It so happens that this distribution is given by a Bayesian network whose structure is the program's grounded dependency graph, and whose parameters are given by the program's Clark completion (Poole, 1993, 2008):

Example 2 *Example 1 describes an acyclic* PLP*, whose dependency graph is the graph depicted in Figure 1 (predicate* neighbor *is omitted as it is always* true*). The figure shows probabilistic assessments and sentences in the Clark completion. For each total choice, there is a unique interpretation for all non-root nodes. Note also that the graph and the completion clearly specify a Bayesian network over the atoms, as each node can be viewed as a random variable yielding* 1 *when the grounding is* true *in an interpretation, and* 0 *otherwise.* □

Conversely, a Bayesian network can be specified by an acyclic propositional PLP (Poole, 1993, 2008). The argument is simple, and we show it by turning Example 2 upside down:

Example 3 *Consider the converse of Example 2: suppose we have only the graph in Figure 1, now without the nodes* a, b, c*, and, instead of the assessments and sentences in the figure, we are given the following probabilities:* $\mathbb{P}(\mathsf{burglary}) = 0.7$*,* $\mathbb{P}(\mathsf{earthquake}) = 0.2$*,* $\mathbb{P}(\mathsf{alarm}|\pi_{\mathsf{true},\mathsf{true}}) = 0.9$*,* $\mathbb{P}(\mathsf{alarm}|\pi_{\mathsf{true},\mathsf{false}}) = 0.8$*,* $\mathbb{P}(\mathsf{alarm}|\pi_{\mathsf{false},\mathsf{true}}) = 0.1$*,* $\mathbb{P}(\mathsf{alarm}|\pi_{\mathsf{false},\mathsf{false}}) = 0.0$*, where* $\pi_{ij} = \{\mathsf{burglary} = i, \mathsf{earthquake} = j\}$*, and finally suppose that for* $X \in \{1, 2\}$*, we have probabilities* $\mathbb{P}(\mathsf{calls}(X)|\mathsf{alarm} = \mathsf{false}) = 0$*,* $\mathbb{P}(\mathsf{calls}(X)|\mathsf{alarm} = \mathsf{true}) = 1$*. This Bayesian network can be*

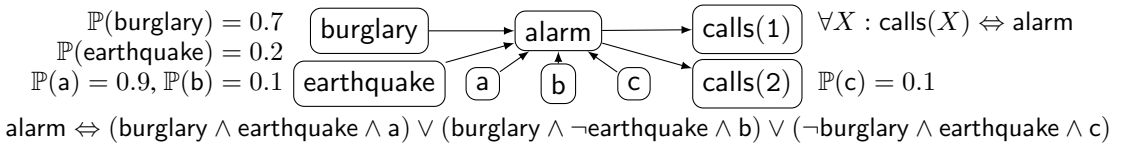

Figure 1: Bayesian network extracted from Example 1.

specified by the PLP *in Example 1, with the understanding that nodes* a, b *and* c *are auxiliary (note that the predicate* neighbor *appears in Example 1, but it does not affect the resulting network as its value is fixed (to* true*) in that example).* □

Given this two-way translation, immediately we see that inference in acyclic propositional PLPs must have the complexity of Bayesian network inference. To formalize this and similar results, we must define precisely our computational problem. We do it right away.

We are interested in the computation of probabilities such as $\mathbb{P}(\mathbf{Q}|\mathbf{E})$, where $\mathbf{Q}$ and $\mathbf{E}$ are truth assignments to selected sets of grounded atoms. For instance, $\mathbf{Q} = \{\text{alarm} = \text{true}, \text{calls}(2) = \text{false}\}$. Of course, $\mathbb{P}(\mathbf{Q}|\mathbf{E}) = \mathbb{P}(\mathbf{Q}, \mathbf{E}) / \mathbb{P}(\mathbf{E})$, hence it is easy enough to obtain the conditional probability from the marginal ones. Note also that complexity classes related to counting can capture the computation of marginal probabilities, but are not believed to be closed under division (Hemaspaandra and Ogihara, 2002). Hence, we focus on the complexity of computing $\mathbb{P}(\mathbf{Q})$; a conditional probability can be easily produced, if so desired, by an extra division.

More precisely, in this paper we are interested in the *inferential complexity* of PLPs, where, as **input**, we have a PLP whose probabilities are rational numbers, and a truth assignment $\mathbf{Q}$ to some ground atoms in its Herbrand base; and as **output**, we have the probability $\mathbb{P}(\mathbf{Q})$. We are also interested in the complexity of inferences when the probabilistic logic program is fixed, and the input is the query. This is the *query complexity* of the program: a PLP where probabilities are rational numbers is fixed, and we have, as **input**, a truth assignment $\mathbf{Q}$ as before, and, as **output**, we again want the probability $\mathbb{P}(\mathbf{Q})$. These concepts are based on analogous concepts found in database theory (Cozman and Mauá, 2015b).[2] In practice one may face situations where the program is large and inferential complexity is the key concept; in other circumstances the program may be small compared to the query, and query complexity is the key concept.

Before we present results on inferential and query complexity of probabilistic models specified by acyclic programs, we must review some needed complexity theory.

2.3 Complexity classes: a few needed complexity classes and reductions

A *language* is a set of bit-strings; a *complexity class* is a set of languages, and we use well-known complexity classes P, NP, EXP (Dantsin et al., 2001; Papadimitriou, 1994). An element of a class of languages/functions is a *problem*. We also use oracle Turing machines, where the oracle itself may be a language or a function. If A and B are classes of languages/functions, $\mathsf{A}^{\mathsf{B}} = \cup_{\mathcal{L} \in \mathsf{B}} \mathsf{A}^{\mathcal{L}}$ ($\mathsf{A}^{\mathcal{L}}$ is class A with oracle $\mathcal{L}$). The *polynomial hierarchy* PH is the class of languages $\bigcup_i \Delta_i^P = \bigcup_i \Pi_i^P = \bigcup_i \Sigma_i^P$, where $\Delta_i^P = \mathsf{P}^{\Sigma_{i-1}^P}$, $\Pi_i^P = \mathsf{co}\Sigma_i^P$, $\Sigma_i^P = \mathsf{NP}^{\Sigma_{i-1}^P}$ and $\Sigma_0^P = \mathsf{P}$.

We use the class #P as defined by Valiant (1979). That is, #P is the class of *integer-valued* functions computed in polynomial time by *counting* Turing machines (a standard nondeterministic

2. We have used *combined* and *data* complexity in previous work, but *inferential* and *query* seem more appropriate.

Turing machine that prints in binary notation, on a separate tape and with unit cost, the number of accepting computations induced by the input). Valiant also defines, for every (decision) complexity class A, the class $\#\mathsf{A}$ to be $\cup_{\mathcal{L}\in\mathsf{A}}(\#\mathsf{P})^{\mathcal{L}}$, where $(\#\mathsf{P})^{\mathcal{L}}$ is the class of functions counting the accepting paths of nondeterministic polynomial time Turing machines with $\mathcal{L}$ as oracle. Hence, $\#\mathsf{NP}$ is the class of functions computed by a counting Turing machine with a NP oracle. And $\#\mathsf{EXP}$, in Valiant's sense, contains functions computed by a counting Turing machine in polynomial time with EXP as oracle. Valiant's class $\#\mathsf{EXP}$ is not always appropriate for our purposes, as it can only produce a counter of polynomial size, and it is often the case that we must compute a probability value with exponentially many bits in the output. For this reason we use $\#\mathsf{EXP}$ to denote the class of functions that can be computed by counting Turing machines taking exponential time, as proposed by Papadimitriou (1986). Note that Papadimitriou's $\#\mathsf{EXP}$ allows for an exponentially large output.

One might be tempted to use $\#\mathsf{P}$-completeness theory to analyze the complexity of probabilistic inference. However, probabilities are not integers produced by counting; some sort of normalization is needed, but we cannot rely on division "inside" counting complexity classes (Hemaspaandra and Ogihara, 2002). In fact, in his seminal work on the complexity of Bayesian networks, Roth (1996) notes that "strictly speaking the problem of computing the degree of belief is not in $\#\mathsf{P}$, but easily seem equivalent to a problem in this class". The challenge is to formalize such an equivalence. We adopt the strategy proposed by Bulatov et al. (2012) in their study of weighted constraint satisfaction problems: they define *weighted reductions* that are suited for our purposes (a similar strategy is adopted by Kwisthout (2011), who discusses $\#\mathsf{P}$ membership *modulo normalization*).

For functions F_1 and F_2 from an input language to the positive rational numbers, a *weighted reduction* is a pair of polynomial time functions G_1 and G_2 such that $F_1(\ell) = G_1(\ell)F_2(G_2(\ell))$ for all ℓ, where G_1 is a function from the language of interest to the positive rational numbers, and G_2 is a transducer. In other words, a weighted reduction is a parsimonious reduction G_2 scaled by a polynomial time computable rational $G_1(\ell)$. In this paper, for complexity class C in the polynomial hierarchy, a function F is $\#\mathsf{C}$-hard if any function in $\#\mathsf{C}$ can be reduced to F via a weighted reduction. And a function F is $\#\mathsf{C}$-equivalent if it is $\#\mathsf{C}$-hard and $G \cdot F$ is in $\#\mathsf{C}$ for some polynomial time function G that maps strings to positive rationals. Finally, F is $\#\mathsf{EXP}$-equivalent if it is $\#\mathsf{EXP}$-hard via weighted reductions where G_1 is allowed to require exponential time effort, and if there is membership of $G \cdot F$ in $\#\mathsf{EXP}$ for some exponential time function G.

The canonical complete problem for class $\#\Pi_k^P$ via parsimonious reductions is $\#\Pi_k\mathsf{SAT}$ (Durand et al., 2005): as **input**, a formula $\varphi(Y) = \forall\, X_1 \exists\, X_2 \ldots Q_k\, X_k \psi(X_1, \ldots, X_k, Y)$, where each of $X_1, \ldots, X_k, Y$ is a set of propositional variables and ψ is in 3-Conjunctive Normal Form (3CNF); as **output**, the number of assignments to the variables Y that make the formula φ evaluate to true.

2.4 The complexity of Bayesian networks specified by acyclic probabilistic logic programs

By combining arguments around Examples 2 and 3, we see that inference in acyclic propositional PLPs is $\#\mathsf{P}$-equivalent (Roth, 1996). One might suspect that a bound on predicate arity would yield the same $\#\mathsf{P}$-equivalence, because the grounding of a PLP would then produce only polynomially-many ground atoms. Surprisingly, this is *not* the case here, as shown by the next theorem, our main result in this section.

Theorem 1 *The inferential complexity of inference in acyclic* PLP*s with bounded predicate arity is* $\#\mathsf{NP}$*-equivalent.*

Proof Membership follows as logical reasoning with this language is Δ_2^P-complete (Eiter et al., 2007, Table 5); for each fixed total choice of the probabilistic facts, logical reasoning determines whether the probability of that total choice is to be taken into account; hence by counting accepting computations (within #NP), one reaches a number that must be later normalized (hence the need for weighted reductions). Hardness is shown by building a PLP that solves a $\#\Pi_1$SAT problem where the formula $\varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is in 3CNF as the conjunction of clauses $c_1, \ldots, c_k$, and the problem is to determine the number of assignments of $x_1, \ldots, x_n$ for which $\forall y_1, \ldots, y_m \varphi(\cdot, y_1, \ldots, y_m)$ is true. For instance,

$$\varphi(x_1, x_2, x_3, y_1) \equiv (\neg x_1 \vee x_2 \vee y_1) \wedge (x_1 \vee \neg x_2 \vee y_1) \wedge \neg y_1. \quad (1)$$

For each propositional variable y_j, we use a corresponding logical variable Y_j. Denote by $\mathbf{Y}_i$ the ordered set of logical variables in $Y_1, \ldots, Y_m$ corresponding to propositional variables in c_i. In Expression (1), $\mathbf{Y}_1 = \mathbf{Y}_2 = \mathbf{Y}_3 = [Y_1]$. Introduce 0-arity predicates $\mathsf{x}_1, \ldots, \mathsf{x}_n$, and predicates $\mathsf{nc}_1, \ldots, \mathsf{nc}_k$. The arity of nc_i is equal to the number of logical variables in $\mathbf{Y}_i$.

For each nc_i, go over the possible assignments of $\mathbf{Y}_i$. If $\mathbf{Y}_i$ makes the clause c_i true, move to the next assignment. If instead $\mathbf{Y}_i$ leaves the clause dependent on some propositional variables in $x_1, \ldots, x_n$, say propositional variables x_{i_1}, x_{i_2} and x_{i_3} (where we allow a propositional variable to appear more than once), then introduce the rule: $\mathsf{nc}_i(\mathbf{y}_j) :- [\mathbf{not}]\mathsf{x}_{i_1}, [\mathbf{not}]\mathsf{x}_{i_2}, [\mathbf{not}]\mathsf{x}_{i_3}.$, where each **not** is included if and only if the literal containing x_{i_1} is positive. Now introduce, for each nc_i, a rule $\mathsf{exists} :- \mathsf{nc}_i(\mathbf{Y}_i).$. Finally, add probabilistic fact $0.5 :: \mathsf{x}_i.$, for each x_i, and compute $\mathbb{P}(\mathsf{exists})$. The Clark completion of the PLP just constructed encodes the $\#\Pi_1$SAT problem of interest, and the desired counting is $2^n\mathbb{P}(\neg\mathsf{exists})$, thus proving #NP-hardness. For instance, given the formula in Expression (1), generate the following PLP:
$\mathsf{nc}_1(\mathsf{false}) :- \mathsf{x}_1, \mathbf{not}\ \mathsf{x}_2.$ $\mathsf{nc}_2(\mathsf{false}) :- \mathbf{not}\ \mathsf{x}_1, \mathsf{x}_2.$ $\mathsf{nc}_3(\mathsf{true}).$
$\mathsf{exists} :- \mathsf{nc}_1(Y_1).$ $\mathsf{exists} :- \mathsf{nc}_2(Y_1).$ $\mathsf{exists} :- \mathsf{nc}_3(Y_1).$ $0.5 :: \mathsf{x}_1.$ $0.5 :: \mathsf{x}_2.$
The number of truth assignments for x_1 and x_2 such that every truth assignment to y_1 is satisfying is equal to $2^2\mathbb{P}(\neg\mathsf{exists})$. ■

Intuitively, this results shows that, to produce an inference for a PLP with bounded predicate arity, one must go through the truth assignments for polynomially large groundings, guessing one at a time (thus a counting nondeterministic Turing machine), and, for *each* assignment, it is then necessary to use an NP-oracle to construct the probability values. Theorem 1 suggests that acyclic PLPs capture a larger set of probabilistic languages than many probabilistic relational models that stay within #P (Cozman and Mauá, 2015a).

The next step is to remove the bound on arity. We obtain:

Theorem 2 *The inferential complexity of inference in acyclic* PLP*s is* #EXP*-equivalent.*

Proof Membership follows from grounding and then running inference with an exponentially large counting Turing machine (up to a multiplying rational). Hardness follows from the fact that plate models (Koller and Friedman, 2009) can be encoded by acyclic PLPs, using the same encoding illustrated by Example 3, and the inferential complexity of inference in enhanced plates with unbounded nesting is #EXP-equivalent (Cozman and Mauá, 2015a). ■

Consider query complexity. The following result is handy (and in fact we use it later):

Theorem 3 *Query complexity is #P-hard for any class of programs that include the programs formed by* $\left\{ \begin{array}{c} 0.5 :: \mathsf{t}(X). \quad \mathsf{c}(Y) :- \mathsf{pos}(X,Y), \mathsf{t}(X). \quad \mathsf{c}(Y) :- \mathsf{neg}(X,Y), \mathsf{f}(X). \\ \mathsf{a}(X) :- \mathsf{t}(X), \mathsf{f}(X). \quad \mathsf{b}(X) :- \mathsf{t}(X). \quad \mathsf{b}(X) :- \mathsf{f}(X). \end{array} \right.$.

Proof Consider a CNF formula $\varphi(x_1, \ldots, x_n)$ with clauses $c_1, \ldots, c_m$. Let P_j (resp., N_j) denote the indices of the positive (negative) literals x_i ($\neg x_i$) in clause j. We will encode the truth-value of a clause c_j as $\mathsf{c}(j)$, the truth-value of a positive (negative) literal ℓ_i as $\mathsf{t}(i)$ ($\mathsf{f}(i)$), and the occurrence of a positive (negative) literal $x_i \in P_j$ ($x_i \in N_j$) as $\mathsf{pos}(i,j)$ ($\neg x_i$). The atoms $\mathsf{a}(i)$ and $\mathsf{b}(i)$ will ensure a single truth-value assignment to each variable x_i by setting $\mathsf{a}(i) = \mathsf{false}$ and $\mathsf{b}(i) = \mathsf{true}$. So assemble a query $\mathbf{Q}$ containing assignments to $\mathsf{c}(j) = \mathsf{true}$ for $j = 1, \ldots, m$, $\mathsf{pos}(i,j) = \mathsf{true}$ for $i \in P_j, j = 1, \ldots, m$, $\mathsf{neg}(i,j) = \mathsf{true}$ for $i \in N_j, j = 1 \ldots, m$, and $\mathsf{a}(i) = \mathsf{false}$ and $\mathsf{b}(i) = \mathsf{true}$ for $i = 1, \ldots, n$. The Clark completion defines $\mathsf{c}(j) \Leftrightarrow \bigvee_{i \in P_j} \mathsf{t}(i) \vee \bigvee_{i \in N_j} \mathsf{f}(i)$ for every c_j, and that $\mathsf{t}(i) \Leftrightarrow \neg \mathsf{f}(i)$ for every x_i. Thus $2^n \mathbb{P}(\mathbf{Q})$ is the model count of the formula. ■

We then obtain:

Theorem 4 *The query complexity of inference for acyclic* PLP*s is #P-equivalent.*

Proof Hardness follows from Theorem 3, and membership follows from the fact that data complexity of logical reasoning in acyclic logical programs is polynomial (Dantsin et al., 2001, Theorem 5.1), and any total choice of probabilistic facts is a certificate. ■

There are subclasses of acyclic PLPs that characterize well-known tractable Bayesian networks. An obvious one is the class of such programs with bounded treewidth, as Bayesian networks subject to such bound are tractable (Koller and Friedman, 2009). As a more interesting example, the acyclic definite PLPs with a bound on predicate arity, with at most one atom per body, correspond to *noisy-or Bayesian networks* (Heckerman, 1990): for these programs, the *Quick-Score* algorithm runs in polynomial time when $\mathbf{Q}$ only contains negated atoms. Alas, this tractability result is quite fragile, as "positive" evidence breaks polynomial behavior as long as $\mathsf{P} \neq \mathsf{NP}$ (Shimony and Domshlak, 2003). Yet another tractable class consists of acyclic definite propositional PLPs such that each atom is the head of at most one rule: inference in this class is polynomial when $\mathbf{Q}$ contains only true (by adapting results from Cozman and Mauá (2015b)). This is also a fragile result:

Proposition 1 *Inference for the class of acyclic propositional* PLP*s such that each atom is the head of at most one rule is #P-equivalent even if (a)* $\mathbf{Q}$ *contains only* true *(and the program may not be definite); (b) the program is definite (and* $\mathbf{Q}$ *may contain* false*).*

Proof Case (a) follows from the fact that such programs can directly encode MAJSAT problems; Case (b) follows by adapting Theorem 1 in Ref. (Cozman and Mauá, 2015b). ■

3. Stratified probabilistic logic programs: interpretation and complexity

A stratified normal logic program is one where the grounded dependency graph has no cycles containing a negative edge (Apt et al., 1988).[3] A stratified program has the useful property that its universally adopted semantics produces a single interpretation.

3. Often such a normal logic program is referred to as a *locally stratified normal logic programs*.

Example 4 *Here is a (cyclic!) stratified program on the relation between smoking, stress, and social influence (Fierens et al., 2014): facts* $\mathsf{influences}(a,b)$**.**, $\mathsf{influences}(b,a)$**.**, $\mathsf{stress}(b)$**.**, *and rules* $\mathsf{smokes}(X) :- \mathsf{stress}(X)$**.** *and* $\mathsf{smokes}(X) :- \mathsf{influences}(Y,X), \mathsf{smokes}(Y)$**.**. *The grounded dependency graph is depicted in Figure 2 (left).* □

Example 5 *Here is a stratified* PLP *with cycles and negation:* $\mathsf{b} :- \mathsf{a}, \mathsf{f}.$, $\mathsf{f} :- \mathsf{b}, \mathsf{g}.$, $\mathsf{g} :- \mathsf{b}.$, $\mathsf{c} :- \mathbf{not}\ \mathsf{b}.$, $\mathsf{h} :- \mathsf{c}.$, $\mathsf{i} :- \mathsf{h}.$, $\mathsf{j} :- \mathsf{i}.$, $\mathsf{c} :- \mathsf{j}.$, $\mathsf{d} :- \mathbf{not}\ \mathsf{c}.$, $\mathsf{k} :- \mathsf{d}.$, $\mathsf{d} :- \mathsf{k}.$, $\mathsf{l} :- \mathsf{k}, \mathsf{e}.$. □

Alas, the semantics of stratified programs is not simple; we will present it using the *stable model semantics* (Gelfond and Lifschitz, 1988). First, a *model* M for a normal logic program $\mathbf{P}$ is an interpretation such that $A_0 \in M$ if there is $A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n.$ where $A_1, \ldots, A_m \in M$ and $A_{m+1}, \ldots, A_n \notin M$. A model M is *minimal* if there is no model M' such that $M' \subset M$. For a program $\mathbf{P}$ and an interpretation $\mathcal{I}$, the *reduct* $\mathbf{P}^{\mathcal{I}}$ is the program obtained by (i) grounding $\mathbf{P}$, (ii) removing all rules that contain in their body a negative literal $\mathbf{not}\ A$ with $A \in \mathcal{I}$, and finally (iii) removing the remaining negative literals. The reduct is clearly a definite program. Then $\mathcal{I}$ is a *stable model* for $\mathbf{P}$ if and only if $\mathcal{I}$ is a minimal model for P^I. A stratified program always has a single stable model that is the semantics of the program.

Now, let us return to our study of probabilistic logic programs as specification languages for probabilistic models, and see whether we can find some graph-based tool to help us represent their semantics. Note that for acyclic PLPs the grounded dependency graph is a correct representation for the constraints in the program, as the parents of a node suffice to determine the truth assignment for the node. Given a stratified program, what exactly are the constraints it encodes? Some constraints are again given by the Clark completion of the program. However, to obtain a complete set of constraints, we must use *loop formulas* (Lin and Zhao, 2002). Loop formulas are added to the Clark completion so that the resulting logical theory has only the "right models" (in the stratified case, a single stable model for each total choice). For a given cycle in the program, a corresponding loop formula is of the form $G \Rightarrow H$, where H is a conjunction involving atoms in a cycle, and G is a formula containing some atoms that are not in the cycle but that appear as parents of nodes in the cycle. The precise definition of loop formulas is somewhat involved; details are omitted but the reader can find guidance from Lin and Zhao (2002) and Lifschitz and Razborov (2006).

Example 6 *Consider a probabilistic version of Example 4, by taking the two rules in that example and the probabilistic facts* $0.3 :: \mathsf{influences}(a,b)$**.**, $0.3 :: \mathsf{influences}(b,a)$**.**, *and* $0.8 :: \mathsf{stress}(b)$**.**. *The grounded dependency graph for this* PLP *is in Figure 2. It is tempting to interpret this graph as a Bayesian network, but of course this is not quite right as the graph is cyclic. The Clark completion imposes the following constraints (note the obvious abbreviations that we use from now on to save space):* $\neg\mathsf{in}(a,a)$, $\neg\mathsf{in}(b,b)$ *and* $\neg\mathsf{st}(a)$, *and additionally* $\mathsf{sm}(a) \Leftrightarrow \mathsf{in}(b,a) \wedge \mathsf{sm}(b)$ *and* $\mathsf{sm}(b) \Leftrightarrow \mathsf{st}(b) \vee (\mathsf{in}(a,b) \wedge \mathsf{sm}(a))$. *However, these latter two constraints are not sufficient to encode the semantics: for total choice* $\mathbf{C}' = \{\mathsf{in}(a,b) = \mathsf{true}, \mathsf{in}(b,a) = \mathsf{true}, \mathsf{st}(b) = \mathsf{false}\}$, *the Clark completion produces only* $\mathsf{sm}(a) \Leftrightarrow \mathsf{sm}(b)$. *Thus we get two possible interpretations, even though the program resulting from this total choice has a single stable model. Now, a loop formula to be added to our running example is* $\neg(\mathsf{st}(a) \vee \mathsf{st}(b)) \Rightarrow \neg\mathsf{sm}(a) \wedge \neg\mathsf{sm}(b)$, *a constraint that eliminates the unwanted model of the Clark completion, leaving both* $\mathsf{sm}(a)$ *and* $\mathsf{sm}(b)$ *as* false *for total choice* $\mathbf{C}'$. □

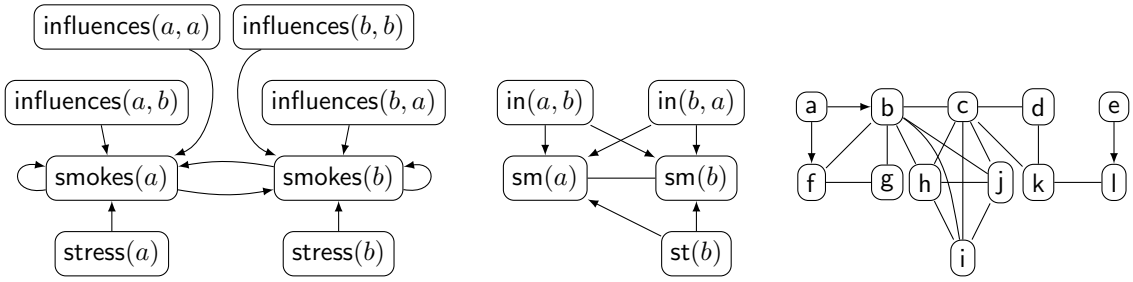

Figure 2: Left: grounded dependency graph for program in Example 4. Middle: chain graph for Example 4. Right: chain graph for Example 7.

Hence, to graphically represent the constraints in our PLP, we must at least encode the loop formulas in the grounded dependency graph. Thus consider the following procedure. First, take the grounded dependency graph, remove directions of all edges, and remove self-loops. Second, remove the root nodes that do not correspond to a given fact (probabilistic or otherwise), as these nodes have fixed truth value by the semantics. Third, create the set of all constraints (Clark completion and loop formulas). For each constraint, connect all atoms that appear in the constraint with an undirected edge. Finally, direct each edge that goes from a probabilistic (ground) fact to another atom (from the probabilistic fact to the atom). In doing so, we obtain a *chain graph* (Koller and Friedman, 2009) that represents all constraints and probability assessments in the PLP (such a chain graph is interpreted using the LWF semantics (Lauritzen and Richardson, 2002); note that logical programs that specify chain graphs have been studied before (Hommersom et al., 2009)). For Example 6, we obtain the graph in Figure 2 (middle), a representation for the meaning of the stratified program.

Example 7 *Take the rules in Example 5 and the probabilistic facts* $0.5 :: \mathsf{a}.$ *and* $0.5 :: \mathsf{e}.$*. The loop formulas are are* $\neg\mathsf{b} \wedge \neg\mathsf{f} \wedge \neg\mathsf{g}$*,* $\neg\mathsf{b} \wedge \neg\mathsf{f}, \mathsf{b} \Rightarrow \neg\mathsf{c} \wedge \neg\mathsf{h} \wedge \neg\mathsf{i} \wedge \neg\mathsf{j}$*, and* $\mathsf{c} \Rightarrow \neg\mathsf{d} \wedge \neg\mathsf{k}$*. The chain graph in Figure 2 (right) conveys these constraints and the Clark completion.* □

It seems appropriate to take these chain graphs as the graph-based interpretation of stratified PLPs. Having looked at the interpretation of stratified programs, we now look at their inferential complexity. As the number of loop formulas for a cyclic program may be large (Lifschitz and Razborov, 2006), one might fear that in moving from acyclic to stratified programs we must pay a large penalty. This is *not* the case: the complexity classes remain the same as in Section 2. Here is our main result in this section:

Theorem 5 *For locally stratified* PLP*s, inferential complexity is (a)* $\#\mathsf{P}$*-equivalent for propositional programs, (b)* $\#\mathsf{NP}$*-equivalent for programs with bounded predicate arity, and (c)* $\#\mathsf{EXP}$*-equivalent for general programs. For locally stratified* PLP*s, query complexity is* $\#\mathsf{P}$*-equivalent for general programs.*

Proof For propositional stratified PLPs, membership follows as logical reasoning with this language is polynomial (Eiter et al., 2007, Table 2), and any total choice of probabilistic facts is a certificate; hardness comes from grounding the program in Theorem 3.

For stratified programs with bounded predicate arity, membership follows as logical reasoning with this language is Δ_2^P-complete (Eiter et al., 2007, Table 5); for each total choice, logical reasoning determines whether the probability of that total choice is to be taken into account; hence by

counting accepting computations (within $\#\mathsf{NP}$), one reaches a number that must be later normalized. Hardness follows from Theorem 1.

For general stratified PLPs, membership follows from the fact that we can ground the PLP into an exponentially large propositional PLP, and then encode inference with a Turing machine there. Hardness is argued as in the proof of Theorem 2 (that is, by using plate models that can be expressed using acyclic programs with negation).

Finally, membership for query complexity follows from the fact that the data complexity of the corresponding logical inference is polynomial (Dantsin et al., 2001); hardness follows from Theorem 3. ■

We noted, at the end of Section 2, that some sub-classes of acyclic programs display polynomial behavior. We now show an analogue result for a sub-class of definite (and therefore stratified, but possibly cyclic) programs with unary and binary predicates:

Proposition 2 *Inferential complexity is polynomial for queries containing only positive literals, for* PLP*s where each atom is the head of at most one rule, and rules take one of the following forms:*

$$\begin{cases} \alpha :: \mathsf{a}(X). \quad \beta :: \mathsf{a}(a). \quad \gamma :: \mathsf{r}(X,Y). \quad \mathsf{a}(a). \quad \mathsf{r}(a,b). \\ \mathsf{a}(X) :- \mathsf{a}_1(X), \ldots, \mathsf{a}_k(X). \qquad \mathsf{a}(X) :- \mathsf{r}(X,Y). \qquad \mathsf{a}(X) :- \mathsf{r}(Y,X). \end{cases}$$

Proof [Sketch] We show that the inference can be reduced to a tractable weighted model counting problem. First, ground the program in polynomial time (because each rule has at most two variables). Since the resulting program is definite, only atoms that are ancestors of the queries in the (grounded) dependency graph are relevant for determining the truth-value of the query in any logic program induced by a total choice (this follows as resolution is complete for propositional definite programs). Thus, discard all atoms that are not ancestors of a query atom. For the query to be true, the remaining atoms that are not probabilistic facts are forced to be true by the semantics. So collect all rules of the sort $\mathsf{a}(a) :- \mathsf{r}(a,b)$., $\mathsf{a}(a) :- \mathsf{r}(b,a)$., plus all facts and all probabilistic facts. This is an acyclic program, so that its Clark completion gives the stable model semantics. This completion is a formula containing a conjunction of subformulas $\mathsf{a}(a) \Leftrightarrow \bigvee_b \mathsf{r}(a,b)$, $\mathsf{a}(a) \Leftrightarrow \bigvee_a \mathsf{r}(a,b)$, and unit (weighted) clauses corresponding to (probabilistic) facts. The query is satisfied only on models where the lefthand side of the definitions are true, which is equivalent to reducing the subformulas to their righthand side. The resulting weighted model counting problem has been shown to be polynomial-time solvable (Mauá and Cozman, 2015). ■

4. Conclusion

We can summarize our main contributions as follows. We have clarified the complexity of acyclic PLPs, showing that their ability to encode Bayesian networks leads to quite expressive complexity classes. We then studied the complexity of (locally) stratified PLPs and their representation by chain graphs — in particular we have shown that acyclic and stratified programs lead to the same complexity classes: $\#\mathsf{P}$-equivalence for propositional programs, $\#\mathsf{NP}$-equivalence for programs with bounded predicate arity, $\#\mathsf{EXP}$-equivalence for programs (without bound on predicate arity), and finally $\#\mathsf{P}$-equivalence for query complexity.

Much more is yet to be explored concerning the encoding of probabilistic models as PLPs. For instance, we have not discussed non-stratified PLPs: What models do they capture? What graphical representations can represent them? Also, other classes of PLPs deserve attention: tight, strict, order-consistent programs may lead to interesting results. There are also several extensions of normal programs that should be investigated as specification languages; for instance, disjunctive programs (Dantsin et al., 2001). The inclusion of functions (with appropriate restrictions to ensure decidability) is another challenge.

Acknowledgements

The first author is partially supported by CNPq, grant 308433/2014-9. The second author received financial support from the São Paulo Research Foundation (FAPESP), grant 2016/01055-1.

References

K.R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. *Foundations of Deductive Databases and Logic Programming*, pages 89–148, 1988.

K R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.

A. Bulatov, M Dyer, L. A. Goldberg, M. Jalsenius, M. Jerrum, and D. Richerby. The complexity of weighted and unweighted #CSP. *J. of Computer and System Sciences*, 78:681–688, 2012.

K. L. Clark. Negation as failure. *Logic and Data Bases*, pages 293–322. Springer, 1978.

F. G. Cozman and D. D. Mauá. The complexity of plate probabilistic models. *Scalable Uncertainty Management*, pages 36–49. Springer, 2015a.

F. G. Cozman and D. D. Mauá. Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity. *AAAI Conf. Artificial Intelligence*, 2015b.

E. Dantsin, T. Eiter, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005.

T. Eiter, W. Faber, M. Fink, and S. Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, 5: 123–165, 2007.

D. Fierens, G. van den Broeck, J. Renkens, D. Shrerionov, B. Gutmann, G. Janssens, and L. de Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2014.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Int. Logic Programming Conf. and Symp.*, pages 1070–1080, 1988.

W. Gilks, A. Thomas, and D. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43:169–178, 1993.

D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. *Conf. on Uncertainty in Artificial Intelligence*, pages 163–172, 1990.

L. A. Hemaspaandra and M. Ogihara. The witness reduction technique. *The Complexity Theory Companion*, Texts in Theoretical Computer Science, pages 91–108. 2002.

A. Hommersom, N. Ferreira, P. J. F. Lucas. Integrating logical reasoning and probabilistic chain graphs. *ECML PKDD*, pages 548–563, 2009.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

J. Kwisthout. The computational complexity of probabilistic inference. Tech.Report ICIS-R11003, 2011.

S. L. Lauritzen and T. S. Richardson. Chain graph models and their causal interpretations. *Journal Royal Statistical Society B*, 64-3:321–361, 2002.

V. Lifschitz and A. Razborov. Why are there so many loop formulas? *ACM Transactions on Computational Logic*, 7(2):261–268, 2006.

F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *AAAI*, pages 112–117, 2002.

D. D. Mauá and F. G. Cozman. DL-Lite Bayesian networks: A tractable probabilistic graphical model. *Scalable Uncertainty Management*, pages 5064. Springer, 2015.

U. Nodelman, C. R. Shelton, and D. Koller. Continuous time Bayesian networks. *Conf. on Uncertainty in Artificial Intelligence*, pages 378–387, 2002.

C. H. Papadimitriou. A note on succinct representations of graphs. *Information and Control*, 71: 181–185, 1986.

C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

J. Pearl. *Causality: Models, Reasoning, and Inference (2nd edition)*. Cambridge University Press, 2009.

D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

D. Poole. The Independent Choice Logic and beyond. *Probabilistic Inductive Logic Programming*, pages 222–243. Springer, 2008.

M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82 (1-2): 273–302, 1996.

T. Sato. A statistical learning method for logic programs with distribution semantics. *Int. Conf. on Logic Programming*, pages 715–729, 1995.

T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. of Artificial Intelligence Research*, 15:391–454, 2001.

S. E. Shimony and C. Domshlak. Complexity of probabilistic reasoning in directed-path singly-connected Bayes networks. *Artificial Intelligence*, 151(1/2):213–225, 2003.

L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. of Computing*, 8 (3):410–421, 1979.