Bayesian Networks Specified Using Propositional and Relational Constructs: Combined, Data, and Domain Complexity

Fabio Gagliardi Cozman and **Denis Deratani Mauá**
Escola Politécnica, Universidade de São Paulo, Brazil

Abstract

We examine the inferential complexity of Bayesian networks specified through logical constructs. We first consider simple propositional languages, and then move to relational languages. We examine both the combined complexity of inference (as network size and evidence size are not bounded) and the data complexity of inference (where network size is bounded); we also examine the connection to liftability through domain complexity. Combined and data complexity of several inference problems are presented, ranging from polynomial to exponential classes.

Introduction

A Bayesian network over categorical variables is often specified through a set of conditional probability distributions that induce a directed acyclic graph (Pearl 1988). Once distributions are specified, an inference can be computed; that is, the probability of a variable given fixed values for other variables can be calculated. The complexity of such inferences is affected by the topology of the induced graph (Kwisthout, Bodlaender, and Van der Gaag 2010).

We can contemplate many ways to specify conditional probabilities that go beyond flat conditional probability tables. There are indeed many specification languages in the literature. In this paper we wish to study the complexity of inferences not as dependent on graph topology, but rather as dependent on the underlying specification language. We note that the relationship between expressivity and inferential complexity has been studied in depth for logical languages, but existing results do not exhaust the parallel study for Bayesian networks.

In this paper we propose a simple strategy to relate expressivity and inferential complexity in Bayesian networks with binary variables. We consider a syntax where each variable X is either associated with a probability, or logically equivalent to a formula (our syntax is clearly inspired by existing specification schemes (Poole 1993; Sato and Kameya 2001)). Our specification framework allows us to move from sub-Boolean languages to relational ones, and to obtain languages whose inferential complexity ranges from FP to $\#\mathsf{EXP}$-complete problems.

In our study of complexity induced by relational languages, we distinguish three concepts. We have complexity with no bounds on specification, evidence, domain: the *combined* complexity. We then have complexity where specification is bounded: the *data* complexity. Finally, we have complexity with specification and evidence bounded: the *domain* complexity. These concepts are directly related to *dqe-liftability* and *liftability* (Jaeger and Van den Broeck 2012). We show that interesting conclusions can be obtained by looking at data complexity; in particular we present a probabilistic variant of the description logic DL-Lite with polynomial inference when conditioning on binary relations (departing from non-polynomial behavior expected in general with such evidence (Jaeger and Van den Broeck 2012)).

The paper is organized as follows. We first review some concepts; then, as a warm-up discussion to fix notation and basic ideas, we introduce our framework in the propositional case. We then move to relational languages; we first examine combined complexity, then data and domain complexity.

Basic Concepts

A Bayesian network consists of a directed acyclic graph where each node is a variable V_i (and we denote by $\mathrm{pa}(V_i)$ the parents of V_i), and where each random variable V_i is associated with a conditional probability distribution with probability values $\mathbb{P}(V_i = v_i|\mathrm{pa}(V_i) = \pi_i)$, where v_i and π_i denote respectively the values of V_i and of $\mathrm{pa}(V_i)$. We assume throughout that each specified probability value is a nonnegative rational. We only consider finite objects in this paper; there are no measurability concerns, and every function can be taken as a random variable, hence we simply use the term "variable" to refer to a random variable (the latter are clearly different from the logical variables we employ later). The set of variables (nodes) is denoted by $\mathbf{V}$. A Markov condition then implies that the joint distribution for all variables factorizes such that $\mathbb{P}(\mathbf{V} = \mathbf{v}) = \prod_{i=1}^{n} \mathbb{P}(V_i = v_i|\mathrm{pa}(V_i) = \pi_i)$, where v_i and π_i are induced by $\mathbf{v}$. We refer to an event $\{V_i = v_i\}$ as an *assignment*. A *conditional probability table* explicitly contains all probability values for a conditional distribution (that is, each $\mathbb{P}(V_i = v_i|\mathrm{pa}(V_i) = \pi_i)$). If a Bayesian network is specified through conditional probability tables, we say the network is *extensively specified*.

A *language* is simply a set of well-formed formulas.

We use well-known complexity classes P, NP, FP, PSPACE, EXP, NEXP, and #P (Papadimitriou 1994). Recall that #P is the class of functions computed by counting Turing machines in polynomial time; a counting Turing machine is a standard nondeterministic Turing machine that prints in binary notation, on a separate tape, the number of accepting computations induced by the input (Valiant 1979). And we adopt #EXP as the class of functions computed by counting Turing machines in exponential time; note that the number of accepting paths may have exponential size (Papadimitriou 1986). Finally, FPSPACE(P) is the class of integer-valued functions computed by a standard nondeterministic Turing machine in polynomial space, with a polynomially long output (Ladner 1989). If a problem is #P-hard and belongs to $\mathsf{FP}^{\#\mathsf{P}[1]}$ (that is, it can be solved with a call to a #P oracle and then polynomially many computations), we follow de Campos, Stamoulis, and Weyland (2013) and call it #P[1]-equivalent. We introduce the same terminology for exponential problems: if a problem is #EXP-hard and belongs to $\mathsf{FP}^{\#\mathsf{EXP}[1]}$, we call it #EXP[1]-equivalent.

Specification Scheme: Propositional Languages

Consider a set of atomic propositions, $A_1, A_2, \ldots, A_n$, and the set Ω of 2^n truth assignments. We can associate a binary variable V_i with atomic proposition A_i, such that $V_i(\omega) = 0$ when A_i is false, and $V_i(\omega) = 1$ when A_i is true, for $\omega \in \Omega$.

We are interested in specifying Bayesian networks over these variables $V_1, \ldots, V_n$. In fact, from now on we simply conflate atomic propositions and their associated variables; we may write propositional sentences containing variables, or probability values over atomic propositions.

We thus assume that a directed acyclic graph is given, where each node is an atomic proposition. We consider a specification strategy that moves away from tables, directly inspired by probabilistic rules (Poole 1993; Sato and Kameya 2001) and structural models (Pearl 2000). We assume that every variable X is associated with either
• a logical equivalence $X \Leftrightarrow F(Y_1, \ldots, Y_m)$, or
• a probabilistic assessment $\mathbb{P}(X = 1) = \alpha$,
where F is a formula on propositions $Y_1, \ldots, Y_m$ and α is a nonnegative rational.

Note that we avoid direct assessments such as $\mathbb{P}(X|Y) = \alpha$, because one can essentially create negation using such an assessment (by adopting $\mathbb{P}(X = 1|Y = 1) = \mathbb{P}(X = 0|Y = 0) = 0$); in our scheme, the use of negation is a decision about the language.

Denote by $\mathcal{B}(\mathcal{L})$ the class of Bayesian networks specified by taking each $F(\cdot)$ as a formula from a language $\mathcal{L}$. Different languages yield different classes of networks. Note that a logical specification can exploit structures that extensional specifications cannot; for instance, to create a Noisy-Or gate (Pearl 1988), we simply say that $X \Leftrightarrow (Y_1 \wedge Z_1) \vee (Y_2 \wedge Z_2)$, where Z_1 and Z_2 are inhibitor variables. On the other hand, we can specify any Bayesian network as long as $F(Y_1, \ldots, Y_m)$ is allowed to be any propositional sentence. To see that, consider a conditional distribution for X given Y_1 and Y_2; we can specify this distribution through

$$X \Leftrightarrow (\neg Y_1 \wedge \neg Y_2 \wedge Z_{00}) \vee (\neg Y_1 \wedge Y_2 \wedge Z_{01}) \vee (Y_1 \wedge \neg Y_2 \wedge Z_{10}) \vee (Y_1 \wedge Y_2 \wedge Z_{11}),$$

where Z_{ab} are fresh binary variables (that do not appear anywhere else), associated with assessments $\mathbb{P}(Z_{ab}) := \mathbb{P}(X|Y_1 = a, Y_2 = b)$. This can be extended to any set $Y_1, \ldots, Y_m$ of conditioning variables, demanding the same space as a conditional probability table.

Given a Bayesian network $\mathbb{B}$, an assignment Q and a set of assignments $\mathbf{E}$, we are interested in $\mathbb{P}(Q|\mathbf{E})$. The set $\mathbf{E}$ is called the *evidence*. Whenever clear we denote assignments by literals; that is, X and $\neg X$ respectively denote $\{X = 1\}$ and $\{X = 0\}$. Similarly, whenever clear we simply write $\mathbb{P}(X|\neg Y)$ instead of $\mathbb{P}(X = 1|Y = 0)$, and so on.

Denote by $\mathsf{INF}(\mathbb{B}, Q, \mathbf{E})$ the calculation of $\mathbb{P}(Q|\mathbf{E})$ with respect to $\mathbb{B}$. Denote by $\mathsf{INF}(\mathcal{L})$, where $\mathcal{L}$ is a particular language (a set of formulas), the set of $\mathsf{INF}(\mathbb{B}, Q, \mathbf{E})$ for the class of Bayesian networks $\mathcal{B}(\mathcal{L})$.

Note that one might contemplate sophisticated forms of evidence, say disjunction of assignments. We only consider conjunctions of assignments. However, in some sub-Boolean languages one does not have negation, and in those cases it may not make sense to allow negated assignments as evidence. Whenever we want to restrict the language of evidence so that only positive evidence is allowed, we denote the inference problem by $\mathsf{INF}^+(\mathcal{L})$.

In short, our strategy is to use this specification scheme to "parameterize" complexity by the adopted language.

Simple languages, and the power of evidence As any Bayesian network can be specified in our syntax using propositional sentences, $\mathsf{INF}^+(\mathsf{Prop}(\wedge, \neg))$ is #P[1]-equivalent (where $\mathsf{Prop}(\wedge, \neg)$ is the set of well-formed propositional sentences with propositions and the usual negation and conjunction (de Campos, Stamoulis, and Weyland 2013)).[1] Later we will also use languages $\mathsf{Prop}(\wedge)$, $\mathsf{Prop}(\wedge, \vee)$, and $\mathsf{Prop}(\wedge, (\neg))$, where the latter consists of formulas with propositions, conjunction and *atomic negation* (defined as follows: only an atomic proposition directly associated with a probabilistic assessment can be negated).

There is obvious interest in finding simple languages $\mathcal{L}$ with tractable $\mathsf{INF}(\mathcal{L})$, so as to facilitate elicitation, decision-making and learning (Darwiche and Provan 1996; Domingos and Webb 2012; Jaeger 2004; Poon and Domingos 2011; Sanner and MacAllester 2005). However, even simple languages lead to hard counting problems (Roth 1996). For example, take the rather simple language Mon2CNF, the set of propositional sentences in CNF (Conjunctive Normal Form) where each clause has two non-negated atomic propositions: because counting satisfying assignments of sentences in Mon2CNF is a #P-hard problem (Valiant 1979), $\mathsf{INF}^+(\mathsf{Prop}(\wedge, \vee))$ is #P[1]-equivalent.

[1]The #P-hardness result is by Roth (1996); there is a technical obstacle to declaring pertinence to #P in that any problem in the latter class yields integers, while Bayesian network inference yields rationals under our assumptions (so polynomially many operations are needed to process the counting output).

So, is there a tractable sub-Boolean class $\mathcal{B}(\mathcal{L})$? Consider the following result, that shows that matters can change dramatically depending of the type of evidence we allow:

Theorem 1 $\mathsf{INF}^+(\mathsf{Prop}(\wedge, (\neg)))$ *belongs to* FP*, and* $\mathsf{INF}(\mathsf{Prop}(\wedge))$ *is* #P*[1]-equivalent.*

Proof. First consider $\mathsf{INF}^+(\mathsf{Prop}(\wedge, (\neg)))$. To run inference with positive evidence, just run d-separation to collect the set of atomic propositions that must be true given the evidence (note: as soon as a node is set to true, its parents must be true, and so on recursively). Then the probability of $Q \wedge \mathbf{E}$ is just the product of probabilities for these latter atomic propositions to be true, and these probabilities are given in the network specification. Now consider $\mathsf{INF}(\mathsf{Prop}(\wedge))$. Pertinence to #P follows from the fact that the language is a subset of $\mathsf{Prop}(\wedge, \neg)$. To prove hardness, consider a sentence ϕ that is in antimonotone 2-CNF (that is, CNF where each clause has two negated atomic propositions) with m clauses; for the ith clause, $\neg A_{i1} \vee \neg A_{i2}$, introduce the assessment $B_i \Leftrightarrow A_{i1} \wedge A_{i2}$, where B_i is a fresh atomic proposition. Now if $\mathbf{E} = \{\neg B_1 \wedge \neg B_2 \wedge \neg \cdots \wedge \neg B_m\}$, then $2^m \mathbb{P}(\mathbf{E})$ is the number of satisfying assignments of ϕ (the solution to a counting problem in #P). Now $\mathbb{P}(\mathbf{E})$ is not an inference in our scheme, but it can be written (by the Chain Rule) as the product of m probabilities that can be individually produced by inferences; because each one of these inferences is independent of the others, we can create a single network by combining these inferences in parallel, and extract the product of inferences in polynomially many operations.[2] To create this combined network we only need conjunction, so everything can be built with $\mathsf{Prop}(\wedge)$. □

Note that the complexity of inference in extensionally specified Bayesian networks is #P[1]-equivalent with or without evidence (de Campos, Stamoulis, and Weyland 2013; Kwisthout 2011). Theorem 1 shows that once we move to sub-Boolean specifications, evidence has power on its own. Indeed, the effect of evidence has been noted in connection with first-order probabilistic models (Van den Broeck and Darwiche 2013); we return to this later.

One might try to concoct additional languages by using specific syntactic forms in the literature (Darwiche and Marquis 2002). We leave this to future work; instead of pursuing various possible sub-Boolean languages, we wish to move to relational languages, where more structure is available. We do so in the next section, after a quick detour.

A detour into polynomial space As a curiosity, here is a language for which (logical) satisfiability and (probabilistic) inference have similar behavior. Suppose each $F(Y_1, \dots, Y_m)$ is written as $Q_1 Z_1 \dots Q_M Z_M F'(Y_1, \dots, Y_m, Z_1, \dots, Z_M)$, where each Q_i is either $\exists$ or $\forall$, and where F' is a propositional sentence such that bound propositions Z_i do not appear anywhere else. That is, each formula is a Quantified Boolean Formula with free variables Y_i. Denote the resulting language by QBFf; satisfiability in this language is PSPACE-complete (Papadimitriou 1994). Easily, we have:

Proposition 1 $\mathsf{INF}(\mathsf{QBFf})$ *is* $\mathsf{FPSPACE}(\mathsf{P})$*-complete.*

Proof. Counting satisfying assignments of a QBFf is $\#\mathsf{P}^{\mathsf{PSPACE}}$-complete (Bauland et al. 2010); also, we have $\mathsf{FPSPACE}(\mathsf{P}) = \#\mathsf{P}^{\mathsf{PSPACE}}$ (Ladner 1989). □

Relational Languages: Combined Complexity

Recent years have seen enormous interest in combinations of probability and logic, often to specify Bayesian networks (Getoor and Taskar 2007; Raedt 2008). Here we extend our previous specification scheme in a simple manner, again following previous proposals (Poole 1993; Sato and Kameya 2001), and in particular our specification scheme follows the semantics of Jaeger's relational Bayesian networks (Jaeger 1997; 2001), and produces a sub-class of those networks.

Consider a vocabulary $\mathcal{V}$ consisting of names of relations and individuals. For a k-ary relation $r \in \mathcal{V}$, denote by $r(x_1, \dots, x_k)$ an atom where each x_j is either a logical variable or an individual. An atom with no logical variable is a *ground atom*.

We assume that a directed acyclic graph is given, where each node is a relation. We assume that every relation r is associated with either

- a logical equivalence $r(x_1, \dots, x_k) \Leftrightarrow$
  $F(x_1, \dots, x_k, s_1, \dots, s_{m'}, a_1, \dots, a_{m''})$, or
- a probabilistic assessment $\mathbb{P}(r(x_1, \dots, x_k) = 1) = \alpha$,

where F is a formula, and α is a nonnegative rational. We need to impose some conditions on F. First, F contains free logical variables from $\{x_1, \dots, x_k\}$, and may contain additional logical variables that must be bound to quantifiers. Second, F contains relations $s_1, \dots, s_{m'}$ that are parents of r in the graph, and individuals $a_1, \dots, a_{m''}$. Our formulas are always assumed to be well-formed formulas that belong to function-free first-order logic (FFFOL).

We refer to a graph and associated logical equivalences and assessments as a *relational Bayesian network*. Note that we are mimicking our propositional specification scheme, as here again we have a restricted form of probabilistic assessment. This allows one to control the kind of negation that can be used, thus allowing sub-Boolean constructs together with relations and quantification.

The semantics is always based on a set $\mathcal{D}$, the *domain*, and a mapping $\mathbf{I}$, the *interpretation*; the latter takes each individual to an element of the domain, and each k-ary relation to a set of k-tuples of the domain. We assume that the interpretation of individuals is constant across interpretations, a common assumption of *rigidity* from probabilistic logic (Bacchus 1990). We assume that every domain is finite, with size given by input N; we simply assume the domain to be $\{1, 2, \dots, N\}$. Once a domain (that is, N) is fixed, we define a random variable for each grounded relation. The idea is similar to the propositional case: a variable corresponding to a grounded relation is a function over all the possible interpretations, that yields 1 when the grounded relation is true in an interpretation, and 0 otherwise. And again, we conflate grounded relations and their corresponding variables.

A relational Bayesian network is interpreted through its grounding: the semantics consists of grounding every rela-

[2]We thank Cassio Polpo de Campos for alerting us to the fact that the m needed inferences can be combined in a single inference.

$$s(x) \Leftrightarrow \exists y : \forall z : \neg s_1(x) \vee s_2(y) \vee s_3(z)$$
$$t(x) \Leftrightarrow s(x) \wedge \exists y : r(x,y) \wedge s_1(y)$$
$$\mathbb{P}(s_1(x)) = 2/3 \quad \mathbb{P}(s_2(x)) = 1/10$$
$$\mathbb{P}(s_3(x)) = 4/5 \quad \mathbb{P}(r(x,y)) = 1/2$$

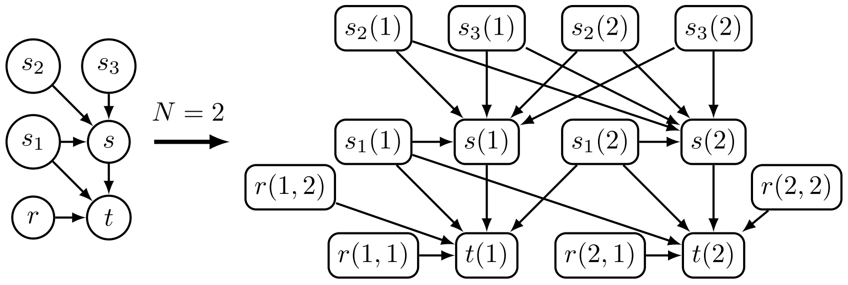

Figure 1: A relational Bayesian network and its grounding with $\mathcal{D} = \{1, 2\}$.

tion, grounding every logical equivalence and every probabilistic assessment, and associating every individual with an element of $\mathcal{D}$. Every relational Bayesian network is so grounded into an extensionally specified Bayesian network (Figure 1). This is identical to grounding in most Probabilistic Relational Models (Getoor and Taskar 2007); we therefore do not discuss this semantics further, as relevant details can be found in previous literature (Jaeger 1997).

Our evidence is now a set of assignments such as $\{r(a,b,c) = 1\}$ and $\{s(b) = 0\}$, denoted respectively by $\{r(a,b,c)\}$ and $\{\neg s(b)\}$. Evidence is positive when no negation appears in the assignments. We do not consider more sophisticated languages for evidence specification.

Domain size in binary notation An important issue is whether the input N is given in binary or unary notation. At first it may seem that binary notation is the natural option. In this case we have:

Lemma 1 *When N is given in binary notation,* $\mathsf{INF}(\mathsf{FFFOL})$ *is in* $\mathsf{FP}^{\#\mathsf{EXP}[1]}$.

Proof. Ground the relational Bayesian network into a Bayesian network of exponential size (each existential quantifier can be written using $\mathcal{O}(N)$ nodes, each one of them a disjunction on two parents; likewise, each universal quantifier can be written as $\mathcal{O}(N)$ nodes, each one of them a conjunction on two parents). Inference on this grounded network can be done with a call to a #EXP oracle, plus some operations to produce a (rational) probability. □

We now consider the inferential complexity of two relatively simple fragments of FFFOL.

Suppose first that F can be a formula in the monadic fragment of FFFOL, denoted by MF (that is, FFFOL restricted to unary relations). For instance, the right hand side of the first sentence in Figure 1 is in MF. Satisfiability in MF is NEXP-complete (Lewis 1980); hence, it is not surprising that

Theorem 2 *When N is given in binary notation,* $\mathsf{INF}(\mathsf{MF})$ *is* #EXP*[1]-equivalent.*

Proof (sketch). As stated by Grove, Halpern, and Koller (1996, Theorem 4.14), counting the number of (suitably defined) distinct model descriptions in monadic first-order logic, without any bound on domain size, is #EXP-hard.[3] The same argument applies here: just turn Lewis' reduction (Lewis 1980, Theorem 4.1) into a parsimonious reduction by reducing the number of counters in his reduction, and reproduce counting by assigning probability $1/2$ to any relation (and by computing a single query). Pertinence comes from Lemma 1. □

[3]Grove, Halpern, and Koller (1996) use #EXP-completeness to mean #EXP[1]-equivalence.

Consider now a language inspired by the popular description logic $\mathcal{ALC}$ (Baader and Nutt 2002). Denote by ALC the language consisting of the set of formulas recursively defined so that any unary relation is a formula, $\neg\phi$ is a formula when ϕ is a formula, $\phi \wedge \varphi$ is a formula when both ϕ and φ are formulas, and $\exists y : r(x,y) \wedge s(y)$ is a formula when r is a binary relation and s is a unary relation. For instance, the right hand side of the second sentence in Figure 1 is in ALC. Even though satisfiability in $\mathcal{ALC}$ belongs to PSPACE, inference is #EXP[1]-equivalent. This fact can be derived from a proof of hardness by Cozman and Polastro (2009), where Turing machines for NEXP-hard problems are explicitly encoded in appropriate variants of $\mathcal{ALC}$. Here we present a short proof, derived from results above:

Theorem 3 *When N is given in binary notation,* $\mathsf{INF}(\mathsf{ALC})$ *is* #EXP*[1]-equivalent.*

Proof. To prove hardness, note that by imposing an assessment $\mathbb{P}(r(x,y)) = 1$, we transform $\exists y : r(x,y) \wedge s(y)$ into $\exists y : s(y)$. Now in Lewis' proof of hardness for monadic logic (used in the proof of Theorem 2), we only need to code a sentence of the form $(\exists x : F_1(x)) \wedge (\neg\exists y_1 : \neg\exists y_2 : F_2(y_1,y_2)) \wedge (\neg\exists z_1 : \exists z_2 : F_3(z_1,z_2))$: such a sentence can be clearly coded within ALC, using $\mathbb{P}(r(x,y)) = 1$ as indicated. Pertinence comes from Lemma 1. □

The message of these results is that even relatively modest fragments of FFFOL take us to #EXP complexity as long as the input N is given in binary notation. In fact, it is not surprising that such encoding moves us away from tractability, for then the *size of the input* is actually $\log N$, and there may be inferences whose result can only be written using exponential space on that input size. For instance, if a language allows existential quantification, one can state $\mathbb{P}(r(x)) = 1/2$ and compute $\mathbb{P}(\exists x : r(x))$; the latter is simply $1 - 1/2^N$, but this number requires N bits (exponentially larger than the input of size $\log N$).

Domain size in unary notation So, suppose the domain size is given in unary notation (that is, N requires input size

N). This is actually the sort of assumption implicit in existing work on liftability (Jaeger and Van den Broeck 2012). Indeed, liftability looks at complexity as a function of N, and it would not be reasonable to worry about polynomial complexity in N if one were to assume an input of size $\log N$.

The difference between binary and unary notation for N is captured by the following analogy. Suppose we are interested in the inhabitants of a city. Their behavior is modeled by a relational Bayesian network. And we have evidence on a few people. By adopting binary notation for N, we just wish to study the consequences of a particular city size; we may say that N is a million, or fifty million, but perhaps this is not even the number of real people in the city. Unary notation for N means that we have a directory, say a telephone catalog, with the names of all inhabitants; even if we do not care about many of them, each one of them exists concretely in our modeled reality.

Now we have:

Lemma 2 *When N is given in unary notation,* $\mathsf{INF}(\mathsf{FFFOL})$ *is in* $\mathsf{FP}^{\#\mathsf{P}[1]}$.

Proof. Ground the relational specification into a Bayesian network of polynomial size in extensional form (add auxiliary variables to bound the in-degree of nodes if necessary, as in the proof of Lemma 1). Inference on this grounded network can be done with a call to a #P oracle, plus some operations to produce a (rational) probability. □

Of course, even very simple sub-Boolean combinations of operators yield #P-hardness for inference, given our previous results. For instance, denote by $\mathsf{QFMF}(\wedge)$ the quantifier-free subset of MF where *only* conjunction can be used. For evidence that can be negated, Theorem 1 already yields #P-hardness (since all grounded relations for an individual are d-separated from the grounded relations for the other individuals, the complexity of the relational language collapses to complexity of the propositional case).

Combined complexity and its discontents As can be noted from previous results, for N in unary notation we obtain #P[1] for the *combined complexity* of inferences in almost any language we can think of. By combined complexity we mean complexity as we place no bound on the size of the relational Bayesian network, no bound on the size of evidence, and no bound on the size of the domain. Now combined complexity is not very informative, for it has little to say when N is given in unary notation, and not much to say either when N is given in binary notation (where we know that exponential behavior is to be expected).

We need finer notions of complexity to capture the behavior of inference for relational specifications. The work on liftability looks in this direction (Jaeger and Van den Broeck 2012). We propose next an analysis that includes liftability.

Data and Domain Complexity, and Liftability

Consider again that we are reasoning about inhabitants of a city. Combined complexity is very demanding: it captures the inferential complexity as we can vary the relational model of inhabitants, the evidence, and the number of inhabitants. We might be interested in fixing a bound on the size of the (supposedly concise) relational model, and studying the complexity of inferences as the evidence accumulates, without any bound on the number of inhabitants. Borrowing from database theory and description logics (Hustadt, Motik, and Sattler 2005; Schaerf 1994), we call the latter notion the *data complexity* of a language. Note that *dqe-liftability* basically means that data complexity is polynomial in N (Van den Broeck 2011; Jaeger and Van den Broeck 2012); clearly, for dqe-liftability to be meaningful there is an implicit assumption that N is given in unary notation, for otherwise polynomial in N means exponential in the input. Data complexity allows one to go beyond the dichotomy given by dqe-liftability: instead of just asking about polynomial time behavior or not, questions may be asked about pertinence and hardness to various complexity classes. Data complexity is based on an assumption that we can force ourselves to develop a relatively small model for the inhabitants of the city, but the number of observations we collect cannot be bounded beforehand.

However, there is yet another dimension in which to analyze complexity. We may be interested in imposing bounds on the size of the relational representation and on the size of evidence, and study the complexity of inferences solely as a function of N. We call the latter notion the *domain complexity* of a language. The concept of *liftability* basically means that domain complexity is polynomial in N (Van den Broeck 2011; Jaeger and Van den Broeck 2012).

In the remainder of this section we focus on data complexity. As far as domain complexity is concerned, we limit ourselves to reproduce later some recent deep results, and to leave more investigation to the future.

Data complexity Denote by $\mathsf{DINF}(\mathcal{L})$ the inference problem where the size of the relational Bayesian network is fixed, while the evidence and N are left free, with N in unary notation. Similarly denote by $\mathsf{DINF}^+(\mathcal{L})$ the same problem when only positive evidence is allowed. These problems characterize data complexity.

Of course, data complexity would be of little value if no useful insight came out of it. We now show this not to be the case.

Consider the following very simple relational language, obtained by restricting the popular description logic DL-Lite (Calvanese et al. 2005). Denote by DLDiet the language consisting of the set of formulas recursively defined so that any unary relation is a formula, $\neg r(x)$ is a formula for any unary relation r, $\phi \wedge \varphi$ is a formula when both ϕ and φ are formulas, and $\exists y : r(x, y)$ is a formula for any binary relation. Directly from Theorem 1 and Lemma 2, we have that

Proposition 2 $\mathsf{INF}(\mathsf{DLDiet})$ *is* #P*[1]-equivalent.*

However,

Theorem 4 $\mathsf{DINF}(\mathsf{DLDiet})$ *belongs to* FP.

Proof (sketch). Without loss of generality, consider a query on some unary relation $q(1)$ grounded on individual 1 and evidence $\mathbf{E}$ on arbitrary individuals. By using d-separation, one can verify that the grounded network containing variables relevant to the query include only unary atoms of the form $s_i(x)$, atoms corresponding to grounded existential formulas $\exists y : r_j(1,y)$ and binary atoms $r_j(1,1),\ldots,r_j(1,N)$. By construction, the existential formulas have only the corresponding binary atoms as parents, and these are all root nodes. Moreover, the in-degree of nodes can be made small (e.g., binary) by inserting new variables corresponding to partial evaluations of the formulas. This way, we can convert the relevant network into an extensional specification form such that the treewidth is smaller than n, the number of relations in the relational Bayesian network. Because this is assumed bounded, inference can be performed, for instance by variable elimination, in polynomial time. $\square$

To further understand the border of tractability, consider a very simple language inspired by the popular description logic $\mathcal{EL}$ (Baader 2003). Denote by EL the language consisting of the set of formulas recursively defined so that any unary relation is a formula, $\phi \wedge \varphi$ is a formula when both ϕ and φ are formulas, and $\exists y : r(x,y) \wedge s(y)$ is a formula when r is a binary relation and s is a unary relation. We have:

Theorem 5 $\mathsf{DINF}(\mathsf{EL})$ *is* $\#\mathsf{P}$*[1]-equivalent.*

Proof. Pertinence follows from Lemma 2. To show hardness, consider an antimonotone 2-CNF formula $\phi(Z_1,\ldots,Z_n) = (\neg Z_{i_1} \vee \neg Z_{i_2}) \wedge \cdots \wedge (\neg Z_{i_{2m-1}} \vee \neg Z_{i_{2m}})$. Construct the relational Bayesian network

$$\mathbb{P}(d(x,y)) = 1\,, \mathbb{P}(t(x)) = \mathbb{P}(\ell(x)) = \mathbb{P}(r(x)) = 1/2\,,$$
$$c(x) \Leftrightarrow (\exists y : \ell(x,y) \wedge t(y)) \wedge (\exists y : r(x,y) \wedge t(y))\,,$$
$$f(x) \Leftrightarrow \exists y : d(x,y) \wedge c(y)\,.$$

The relations $\ell(x,y)$ and $r(x,y)$ represent the left and right literals, respectively, of each clause; $\ell(x,y)$ is true iff variable Z_y is the first literal of the x-th clause, $r(x,y)$ is true iff Z_y is the second literal of the x-th clause. The relation $t(x)$ denotes the evaluation of each literal: $t(x)$ is true iff $\neg Z_x$ evaluates to true. Relation $c(x)$ represents *un*satisfiability of each clause: $c(x)$ is true iff the x-th clause evaluates to false. The relation $d(x,y)$ induce dummy variables. Finally, $f(x)$ denotes the satisfiability of the formula ϕ. Let $N = \max(n,m)$. We solve the counting problem by computing $\mathbb{P}(f(1)|\mathbf{E})$, where $\mathbf{E}$ is the conjunction of

$$\begin{array}{rl} \ell(x,y) & \forall x,y \text{ s.t. } Z_y \text{ is the left literal of clause } x\,, \\ \neg\ell(x,y) & \forall x,y \text{ s.t. } Z_y \text{ is \textit{not} the left literal of clause } x\,, \\ r(x,y) & \forall x,y \text{ s.t. } Z_y \text{ is the right literal of clause } x\,, \\ \neg r(x,y) & \forall x,y \text{ s.t. } Z_y \text{ is \textit{not} the right literal of clause } x\,. \end{array}$$

As $\mathbb{P}(\neg f(1)|\mathbf{E}) = 2^{-n}\sum_{z_1,\ldots,z_n}\phi(z_1,\ldots,z_n)$, the number of models of ϕ is $2^n(1-\mathbb{P}(f(1)|\mathbf{E}))$. $\square$

In the proof above we make use of negated evidence on binary relations; data complexity in EL with negated evidence only on unary relations, or with positive evidence, are open questions with potential impact on probabilistic description logics (Lukasiewicz and Straccia 2008).

Discussion We have visited many languages so far. The following table captures some interesting facts about combined and data complexity (always assuming N in unary notation). Here $\#\mathsf{P}[1]$ denotes $\#\mathsf{P}[1]$-equivalence. Here, $\mathsf{QFMF}(\wedge,\neg)$ is similar to $\mathsf{QFMF}(\wedge)$, but with negation.

$\mathcal{L}$	$\mathsf{INF}^+/\mathsf{INF}(\mathcal{L})$	$\mathsf{DINF}^+/\mathsf{DINF}(\mathcal{L})$
$\mathsf{QFMF}(\wedge,(\neg))$	FP/#P[1]	FP/FP
$\mathsf{QFMF}(\wedge,\neg)$	#P[1]/#P[1]	FP/FP
DLDiet	#P[1]/#P[1]	FP/FP
EL	#P[1]/#P[1]	?/#P[1]
ALC	#P[1]/#P[1]	?/#P[1]
MF	#P[1]/#P[1]	2-variable frag.: FP/FP

Data complexity of $\mathsf{QFMF}(\wedge)$ and $\mathsf{QFMF}(\wedge,\neg)$ follows from the fact that the relational representation for one individual can be compiled beforehand (Darwiche and Marquis 2002); all other relations are d-separated and can be discarded. The data complexity for the two-variable fragment of MF follows from results on conditioning by Van den Broeck and Darwiche (2013); the general case seems to be open. The data complexity of EL and of ALC for positive evidence is open.

Domain complexity for many languages lies in FP, from results on liftability by Van den Broeck, Wannes, and Darwiche (2014). We should note that existing results also show fundamental limits on liftability for various languages (Jaeger and Van den Broeck 2012).

Conclusion

We have presented a framework for specification and analysis of Bayesian networks. Our conventions are inspired by existing proposals, with a few changes that allow us to go from sub-Boolean languages to various fragments of function-free first-order logic. In particular, the ability to examine languages without negation allows us to bypass inherent limits imposed by treewidth (Kwisthout, Bodlaender, and Van der Gaag 2010). In our analysis, we have displayed specification languages with (combined) complexity ranging from FP to $\#\mathsf{P}[1]$ to $\#\mathsf{EXP}[1]$.

We have also introduced the three dimensional framework of combined/data/domain complexity. We have investigated data complexity, and shown that it can capture subtle distinctions between specification languages. Concerning domain complexity, we only note that it subsumes liftability, and leave further investigation to the future.

As far as future work is concerned, the goal must be to obtain an understanding of expressivity/complexity in Bayesian networks as rich as the understanding that we now have about logical languages. For instance, we might consider Bayesian networks specified by operators from description and modal logics, or look at languages that allow variables to have more than two values. In a different direction, we might look at parameterized counting classes (Flum and Grohe 2004), so as to refine the analysis even further.

Acknowledgements

The first author is partially supported by CNPq (grant 305395/2010-6) and the second author is supported by FAPESP (grant 2013/23197-4). We thank Cassio Polpo de Campos for valuable insights concerning complexity proofs.

References

Baader, F., and Nutt, W. 2002. Basic description logics. In *Description Logic Handbook*. Cambridge University Press. 47–100.

Baader, F. 2003. Terminological cycles in a description logic with existential restrictions. In *International Joint Conf. on Artificial Intelligence*, 364–369.

Bacchus, F. 1990. *Representing and Reasoning with Probabilistic Knowledge: A Logical Approach.* MIT Press.

Bauland, M.; Bohler, E.; Creignou, N.; Reith, S.; Schnoor, H.; Vollmer, H. 2010. The complexity of problems for quantified constraints. *Theory Computing Systems* 47:454–490.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2005. *DL-Lite*: Tractable description logics for ontologies. *AAAI*, 602–607.

Cozman, F. G.; and Polastro, R. B. 2009. Complexity analysis and variational inference for interpretation-based probabilistic description logics. In *Conf. on Uncertainty in Artificial Intelligence*, 117–125. Corvallis, Oregon.

Darwiche, A.; and Provan, G. 1996. Query DAGs: A practical paradigm for implementing belief-network inference. In *Conf. on Uncertainty in Artificial Intelligence*, 203–210.

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

de Campos, C. P.; Stamoulis, G.; and Weyland, D. 2013. A structured view on weighted counting with relations to quantum computation and applications. Technical Report 133, Electronic Colloquium on Computational Complexity.

Domingos, P.; and Webb, W. A. 2012. A tractable first-order probabilistic logic. In *AAAI*, 1902–1909.

Flum, J.; Grohe, M. 2004. The parameterized complexity of counting problems. *SIAM J. of Computing* 33(4):892–922.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Grove, A.; Halpern, J.; and Koller, D. 1996. Asymptotic conditional probabilities: the unary case. *SIAM Journal on Computing* 25(1):1–51.

Hustadt, U.; Motik, B.; and Ulrike Sattler. 2005. Data complexity of reasoning in very expressive description logics. In *International Joint Conf. on Artificial Intelligence*, 466–471.

Jaeger, M.; and Van den Broeck, G. 2012. Liftability of probabilistic inference: Upper and lower bounds. In *Statistical Relational AI Workshop*.

Jaeger, M. 1997. Relational Bayesian networks. In *Conf. on Uncertainty in Artificial Intelligence*, 266–273.

Jaeger, M. 2001. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Mathematics and Artificial Intelligence* 32:179–220.

Jaeger, M. 2004. Probabilistic role models and the guarded fragment. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 235–242.

Kwisthout, J. H. P.; Bodlaender, H. L.; and Van der Gaag, L. 2010. The necessity of bounded treewidth for efficient inference in Bayesian networks. *European Conf. on Artificial Intelligence*, 237–242.

Kwisthout, J. H. P. 2011. The computational complexity of probabilistic inference. Technical Report ICIS–R11003, Radboud University Nijmegen, The Netherlands.

Ladner, R. E. 1989. Polynomial space counting problems. *SIAM Journal of Computing* 18(6):1087–1097.

Lewis, H. R. 1980. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences* 21:317–353.

Lukasiewicz, T.; and Straccia, U. 2008. Managing uncertainty and vagueness in description logics for the semantic web. *Journal of Web Semantics* 6:291–308.

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

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

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge, United Kingdom: Cambridge University Press.

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

Poole, D. 2003. First-order probabilistic inference. In *International Joint Conf. on Artificial Intelligence*, 985–991.

Poon, H.; and Domingos, P. 2011. Sum-product networks: a new deep architecture. In *Conf. on Uncertainty in Artificial Intelligence*, 337–346.

Raedt, L. D. 2008. *Logical and Relational Learning*. Springer.

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

Sanner, S.; and MacAllester, D. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *International Joint Conf. on Artificial Intelligence*, 1384–1390.

Schaerf, A. Query answering in concept-based knowledge representation systems: Algorithms, complexity, and semantic issues. Thesis, Università di Roma "La Sapienza".

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

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

Van den Broeck, G. 2011. On the completeness of first-order compilation for lifted probabilistic inference. In *Conf. on Neural Information Processing Systems*.

Van den Broeck, G.; and Darwiche, A. 2013. On the complexity and approximation of binary evidence in lifted inference. In *Conf. on Neural Information Processing Systems*.

Van den Broeck, Guy; Wannes, M.; Darwiche, A. 2014. Skolemization for weighted first-order model counting. In *International Conf. on Principles of Knowledge Representation and Reasoning*.