Inference with Aggregation Parfactors: Lifted Elimination with First-Order d-Separation

Felipe I. Takiyama
Escola Politécnica
Universidade de São Paulo

Fabio G. Cozman
Escola Politécnica
Universidade de São Paulo

***Abstract*—In this paper we focus on lifted inference for statistical relational models; that is, inference that avoids complete grounding, in models that combine logical and probabilistic assertions. We focus on relational Bayesian networks that can be represented through parfactors and aggregation parfactors. We present a new elimination rule for lifted variable elimination, and show how to use first-order d-separation to extend the reach of existing elimination rules.**

I. Introduction

Bayesian networks and Markov random fields offer excellent tools to express probabilistic assessments over a large number of variables [1]. These graph-based "languages" are propositional in that random variables are not indexed by elements of a domain. For example, we may have variables such as $\mathsf{victoryByBob}$. Now if we have a population, we may be interested in whether Ann, Bob, etc, obtain a victory or not. The natural strategy is to consider *parameterized* random variables such as $\mathsf{win}(x)$, where x is a logical variable running over the population of interest [2], [3], [4]. We might write down a graph such as $\mathsf{bet}(x) \longrightarrow \mathsf{win}(x) \longrightarrow \mathsf{noWinner}$, to encode a possibly large Bayesian network consisting of many branches pointing to $\mathsf{noWinner}$.

In this paper we consider the problem of running inference (that is, computing probabilities) in such parameterized Bayesian networks. We consider Kisynski and Poole's extension of first-order variable elimination [5], [6], currently the state-of-art when it comes to *lifted* inference with aggregation parfactors. Here the meaning of "lifted" inference is that inference is performed without completely grounding all parameterized random variables.

We have two contributions. First, we present a new elimination rule for aggregation parfactors, and show its correctness. Second, we indicate how first-order d-separation tests can be used to enhance the performance of existing elimination rules.

Section II presents the necessary background. We then introduce a new inference rule in Section III, and discuss the effect of d-separation in Section IV.

II. Parfactors and Aggregation Parfactors

Graph-based probabilistic models represent joint distributions over variables $\mathbf{X} = \{X_1, \dots, X_n\}$ [1]. In this paper we assume every random variable to have a finite number of values. A Bayesian network consists of a directed acyclic graph where each node is a variable X_i, coupled with a factorization condition on the joint distribution; namely, $P(\mathbf{X}) = \prod_{i=1}^n P(X_i|\mathrm{pa}(X_i))$, where $P(\cdot)$ denotes probability and $\mathrm{pa}(X)$ denotes the parents of node/variable X. A Markov random field, or Markov network, consists of an undirected acyclic graph where each node is again a variable X_i, coupled with a factorization condition (under the assumption of positivity): $P(\mathbf{X}) = \mathbb{Z}^{-1} \prod_{k=1}^m f_k(\mathbf{X}_k)$, where k runs over the m cliques of the graph, $\mathbf{X}_k$ is the set of variables in the kth clique, f_k is any positive function of $\mathbf{X}_k$, and the normalization constant $\mathbb{Z}$ is called the *partition function*. In very general terms we can say that both Bayesian networks and Markov networks share the property that associated joint probabilities behave as products of *factors*.

The graph-based models in the previous paragraph can be extended to *parameterized* counterparts. In doing so, we wish to obtain a more flexible language, with features of first-order logic. There are several frameworks to do so [7], [8]; here we adopt Poole's language of *parfactors* [4], defined as follows. A *domain* is a set of individuals. We only consider finite domains here. A logical variable x runs over a specific domain, denoted by $\mathcal{D}(x)$. A parameterized random variable, or simply *parvariable*, consists of a symbol $r(t_1, \dots, t_k)$, where r is a predicate and each t_i is either a logical variable or a constant. Here we only consider parvariables with values true and false, but the results can be easily generalized to any finite number of values. A parvariable $r(t_1, \dots, t_k)$, where possible groundings are subject to a set of constraints C, represents a set $\mathsf{RV}(r(t_1, \dots, t_k) : C)$ of random variables; that is, the set obtained by grounding $r(t_1, \dots, t_k)$ in every possible way allowed by C (that is, replacing every logical variable by possible individuals).

A *parfactor* is a parameterized function, as it is a function of parvariables. More formally, a parfactor is a tuple $< L, C, V, \phi >$, where L is a set of logical variables, each tied to a domain, C is a conjunction of inequality constraints on logical variables in L and appropriate constants, V is a set of parvariables, and ϕ is a function of parvariables in V. For instance, C may be $(x \neq y) \wedge (x \neq a) \wedge (y \neq b)$, where x and y are logical variables and a and b are constants. Logical variables in C and ϕ must be in L. A parfactor represents a set of factors, one for each grounding of parvariables in V (where groundings must satisfy constraints in C).

A set of parfactors represents a joint distribution that is produced by multiplying all factors produced by ground-

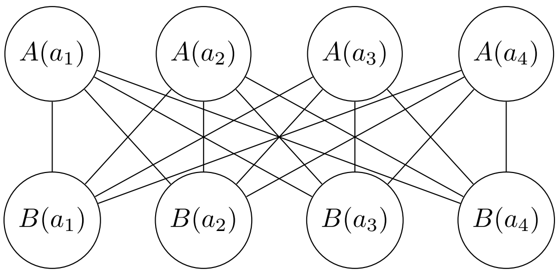

Figure 1. A dense Markov network produced by a single parfactor.

ing. With these conventions, we can use parfactors to encode *relational* Markov networks; that is Markov networks that are specified using relations and logical variables.

The next example builds a Markov network from a single parfactor. Note that for a formula θ, $\mathbb{I}[\theta]$ denotes the indicator function that yields 1 if θ holds and 0 otherwise.

Example 1 Consider the singleton set

$$\{< \{x, y\}, \emptyset, \{A(x), B(y)\}, 1 + 3\mathbb{I}[A(x)] + \mathbb{I}[B(y)] >\}.$$

Note: the function $1 + 3\mathbb{I}[A(x)] + \mathbb{I}[B(y)]$ is a function of $A(x)$ and $B(y)$; for instance, if $A(x)$ is true and $B(y)$ is false, the function yields $1 + 3 + 0 = 4$. Now if x and y have identical domains $\{a_1, a_2, a_3, a_4\}$, we see that this parfactor represents the rather dense Markov network in Figure 1. $\square$

It is also possible to define a parfactor that *aggregates* the values of a set of grounded random variables. In this case it is best to understand the parfactors as specifying a directed model such as a relational Bayesian network. The next example illustrates this case.

Example 2 A set of three parfactors is defined as $\{g_1, g_2, g_3\}$, such that all logical variables run over the same domain $\mathcal{D} = \{a_1, \ldots, a_n\}$, as follows:

$$g_1 =< \{y\}, \emptyset, \{C(y)\}, \phi_1 >,$$

$$g_2 =< \{x, y\}, \emptyset, \{B(x, y), C(y)\}, \phi_2 >,$$

$$g_3 =< \{x\}, \emptyset, \{A(x), B(x, a_1), \ldots, B(x, a_n)\}, \phi_3 >,$$

where each ϕ_i is a parameterized probability distribution:

$$\begin{aligned}
\phi_1 &= P(C(y)) = 0.2 + 0.6\mathbb{I}[C(y)], \\
\phi_2 &= P(B(x, y)|C(y)) = (0.4 + 0.2\mathbb{I}[B(x, y)])\mathbb{I}[C(y)] \\
&\qquad + (0.3 + 0.4\mathbb{I}[\neg B(x, y)])\mathbb{I}[\neg C(y)], \\
\phi_3 &= P(A(x)|B(x, a_1), \ldots, B(x, a_n)) \\
&= \mathbb{I}[A(x) \leftrightarrow \exists y : B(x, y)].
\end{aligned}$$

Note that g_3 aggregates groundings of $B(x, y)$ with respect to an auxiliary logical variable y. $\square$

The parfactors in the previous example, taken as parameterized conditional probability distributions, can be either viewed as specifying a relational Markov network

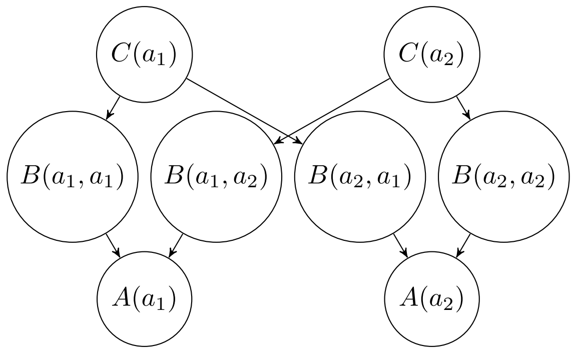

Figure 2. Bayesian network by grounding parfactors in Example 2.

or, more directly, a relational Bayesian network [2] with the following structure:

$$C(y) \longrightarrow B(x, y) \Longrightarrow A(x),$$

where we indicate by the double arrow that the distribution of $A(x)$ given $B(x, y)$ in fact aggregates all $B(x, y)$ with respect to groundings of y. Intuitively, the aggregation is necessary as $B(x, y)$ uses logical variables that are not used by $A(x)$, in this case the logical variable y. By grounding the parfactors we obtain a Bayesian network; in Figure 2 such a grounding is depicted when both logical variables have identical domain $\mathcal{D} = \{a_1, a_2\}$ and empty constraints. From now on we always assume that a given set of parfactors specifies a relational Bayesian network.

We now must define aggregation parfactors: that is, functions with a number of arguments that depends on the number of grounding. Inspired by Kisynski and Poole [5], [6], we define an aggregation parfactor as a tuple $< L, C, r, s, \otimes, \phi_x >$, where L is a set of logical variables and C is a set of constraints (with the same conventions as before), r and s are parvariables such that only one logical variable that is not in s appears in r (this logical variable is denoted by x_{r-s}), and $\otimes$ is a deterministic commutative and associative binary operator; finally ϕ_x is a factor on r. Aggregation parfactors offer a restricted set of *combination* functions in Jaeger's sense [2].

An aggregation parfactor $< L, C, r, s, \otimes, \phi_x >$ encodes the parameterized distribution of s given r. As will be clear later, we also need *generalized aggregation parfactors* that are tuples $< L, C, r, s, V, \otimes, \phi_x >$ [6, Definition 3.3]. All components of such a tuple behave as before, and V is a set of parvariables, subject to some conditions. These parvariables appear in other parfactors that must be combined with the current parfactor in the course of inference; they are called *context* variables (their use is clarified later). Additionally, the factor ϕ_x can now be a function of r and $v_1, \ldots, v_m$ where the latter are parvariables in V. When the factor ϕ_x is unitary (denoted by 1), that is, a function that returns 1 for every value in its domain, we may omit it from the aggregation parfactor. The semantics of such a simplified aggregation parfactors is that it represents a set of factors (one for each appropriate grounding) that yield 1 if $\otimes_{a \in \mathcal{D}(x_{r-s})} r(\ldots, a, \ldots)$ is equal to the value of $s(\ldots)$, and 0 otherwise.

Full-blown aggregation factors in addition contain an explicit set of constraints over x_{r-s} [6]. These additional components are needed to specify the complete AC-FOVE algorithm, but we will not need them here.

As already indicated, even though we can always interpret a set of parfactors as a representation for a relational Markov network, we will assume that they are given as parameterized conditional probability distributions representing a relational Bayesian network. Aggregation parfactors are important in this setting as they allow one of encode the dependence of a random variable on several parent random variables (as in Example 2 for random variables $A(a_i)$). In essence, aggregation parfactors are needed in relational Bayesian networks whenever a parvariable contains less logical variables than its parents.

III. Lifted Inference: the AC-FOVE Algorithm

An *inference* denotes the calculation of the probability of a (possibly grounded) relation perhaps given other grounded relations. For instance, in Example 2 we may be interested in $P(A(a_1)|C(a_2))$. Such a calculation can proceed by grounding all parfactors into a Bayesian network, and then running inference there. Alternatively, inference may be lifted in that it may exploit symmetries in the parfactors, thus producing results without a complete grounding of all parfactors.

Existing algorithms for lifted inference employ *counting formulas*. The motivation is simple: when dealing with parfactors, it is usually the *number* of groundings assigned true or false that matter, and not the specific truth assignment to all groundings. Thus it is often enough to encode the number of true/false assignments as a variable. More precisely, a counting formula is defined as follows [9]. Given a parvariable r and constraints C, a counting formula $\#(x:C)[r]$, where x is a logical variable, is a parvariable (possibly parameterized in logical variables other than x) with values $0,\ldots,n$, where n is the number of groundings of x subject to constraints C. (Note that this definition is somewhat simplified as we assume that all parvariables are Boolean.) If C is empty, it is ommited. The parvariable $\#(x{:}C)[r]$ can obviously be grounded by fixing the logical variables other than x; the meaning of $(\#(x{:}C)[r]) = k$, when the all logical variables in r are grounded, is that exactly k groundings of r, subject to constraints in C, are true. By convention, $\mathsf{RV}(\#(x{:}C)[r])$ is simply $\mathsf{RV}(r : C)$.

The restriction that a single parvariable r appears in a counting formula is not essencial, as it is possible to define counting formulas over a set of parvariables that share the logical variable x [10], [11].

One interesting use of counting formulas is to encode aggregation operations; for instance, the parfactor g_3 in Example 2 can be encoded with a function $\phi_3 = f(A(x), \#y[B(x,y)])$ that yields

$$\begin{cases} 1 & \text{if} \begin{array}{c} (A(x) = \text{true} \wedge \#y[B(x,y)] > 0) \vee \\ (A(x) = \text{false} \wedge \#y[B(x,y)] = 0) \end{array} \\ 0 & \text{otherwise.} \end{cases}$$

(This sort of representation for aggregation parfactors is obtained by Kisynski and Poole's operation of *conversion* [6].) Given that $\otimes$ is commutative and associative, only counts matter; an inference strategy is to convert every aggregation to counting formulas and proceed. However, a more refined strategy is possible, one that exploits the structure of aggregations. We discuss such a strategy later.

We will *always* assume that each pair of parvariables r and s, and associated constraints, satisfies: either $\mathsf{RV}(r : C') = \mathsf{RV}(s : C'')$ or $\mathsf{RV}(r : C') \cap \mathsf{RV}(s : C'') = \emptyset$. It is always possible to guarantee this condition for each operation in inference, possibly by *splitting* parvariables and counting formulas (the splitting operation can be found elsewhere [9]).

We will also assume that each parfactor is kept in *normal form*, possibly by splitting [9]. That is, the set C^* of all constraints in the parfactor and its counting formulas is such that for each inequality $x \neq y$ in C^* we have $\mathcal{E}(x)\backslash\{y\} = \mathcal{E}(y)\backslash\{x\}$, where $\mathcal{E}(x)$ is the set of terms t such that $(x \neq t) \in C^*$ (and likewise for $\mathcal{E}(y)$).

A. C-FOVE

The C-FOVE algorithm [9] is a refinenment on several first-order variable elimination algorithms [4], [12], [13], [14], [15]. The idea is to stay close to the variable elimination algorithm used to compute inferences in Bayesian networks [1]. In variable elimination, at each step a variable is selected and eliminated by multiplication of factors and summation. In C-FOVE, if possible a parvariable is selected and eliminated by multiplication of parfactors and "lifted" summation. To do so, operations of splitting (a parfactor may be split into several), unification (logical substitutions may be needed to allow parfactors to be multiplied together), multiplication, and exponentiation may be needed. These operations are detailed by Milch et al [9]. However, it may happen that in a particular situation no variable can be eliminated; conditions that must be satisfied for a variable to be eliminated are indicated later. In such a case C-FOVE has two options. One option is to ground a selected parvariable, and proceed. Note that grounding a parvariable may create a large number of random variables, thus leading to high computational cost.

The second option for C-FOVE, when elimination is not possible, is to introduce a counting formula, as follows. Suppose $< L, C, \{r_1,\ldots,r_m\}, \phi >$ is a parfactor in normal form, and there is one relation r_i such that a free logical variable x appears only in r_i. We can remove this parfactor from the given set of parfactors, and replace it with a new parfactor, such that inferences on the remaining parvariables are not affected. This new parfactor is

$$< L', C', \{r_1,\ldots,r_{i-1}, \#(x{:}C_x)[r_i], r_{i+1},\ldots,r_m\}, \phi' >,$$

where $L' = L\backslash\{x\}$, C' denotes the set of constraints in C restricted to inequalities involving $L\backslash\{x\}$, C_x denotes the constraints C restricted to inequalities involving only

x, and ϕ' is such that, for $k \in \{0, \dots, n\}$,

$$\begin{aligned}\phi'(r_1, \dots, r_{i-1}, k, r_{i+1}, \dots, r_m) &= \\ &\phi(r_1, \dots, r_{i-1}, \text{true}, r_{i+1}, \dots, r_m)^k \\ &\phi(r_1, \dots, r_{i-1}, \text{false}, r_{i+1}, \dots, r_m)^{n-k}.\end{aligned}$$

Note that counting formulas may be summed out or grounded as needed so as to allow parvariables to be eliminated. These operations are, again, detailed by Milch et al [9].

An enhanced version of C-FOVE is given by Taghipour et al [10]; in this version it is possible to introduce counting formulas on several parvariables simultaneously. That is, we may have $\#(x:C)[r_1, r_2, \dots, r_m]$. Another enhanced version also allows more complex elimination patterns, by grouping random variables [11].

C-FOVE is guaranteed to run inference exactly, and in specific problems it may be able to avoid grounding. However, it may need to ground parvariables even in simple problems.

Example 3 Consider again Example 2, and inference $P(A(a_1))$. C-FOVE cannot eliminate $C(y)$ because it appears in two parfactors whose multiplications yields a parfactor with two logical variables (elimination only applies for a parvariable that contains all logical variables in the parfactor). And C-FOVE cannot eliminate $B(x, y)$: if the aggregation parfactor is turned into a parfactor with counting formula $\#y[B(x, y)]$, then this parfactor must be multiplied with g_2 and $B(x, y)$ cannot be eliminated from the result (elimination only removes a parvariable r from a parfactor when $\text{RV}(r : C) \cap \text{RV}(r' : C) = \emptyset$ for any other r' in the parfactor; $B(x, y)$ and $\#y[B(x, y)]$ violate this). Grounding $B(x, y)$ is the only way to proceed. $\Box$

B. AC-FOVE

The AC-FOVE algorithm follows the same structure of the C-FOVE algorithm, but provides operations that deal directly with aggregation parfactors without converting them into counting formulas [6, Section 3.4.3]. Suppose the aggregation parfactor of interest is $g =< L, C, r, s, V, \otimes, \phi_x >$. The AC-FOVE algorithm offers adapted splitting, multiplication, and elimination operations for such a parfactor, even for cases when the "child" parvariable s contains logical variables that are not in the "parent" parvariable r.

Consider summing out the parvariable r from g, assuming that no other parfactor involves parvariables whose grounding intersects $\text{RV}(r : C)$. In this case, AC-FOVE offers a logarithmic-time algorithm to eliminate r [6, Propositions 3.6 and 3.7], by adapting Pingala's square-and-multiply algorithm. In effect, the conditions assumed by the algorithm impose that all groundings $\text{RV}(r : C)$ must be independent of each other. When such an assumption cannot be made, AC-FOVE examines the possibility that an aggregation parfactor may share other parvariables (those contained in the context set V) with other parfactors. In some cases, the addition of context parvariables works perfectly, as in the next example.

Example 4 Consider a variant on Example 2, where:

$$g_1 =< \{y\}, \emptyset, \{C(y)\}, \phi_1 >,$$

$$g_2 =< \{x, y\}, \emptyset, \{B(x, y), C(x)\}, \phi_2 >,$$

$$g_3 =< \{x\}, \emptyset, \{A(x), B(x, a_1), \dots, B(x, a_n)\}, \phi_3 >,$$

where ϕ_1 and ϕ_3 are as in Example 2, but ϕ_2 is:

$$\begin{aligned}\phi_2 &= P(B(x, y)|C(x)) \\ &= (0.4 + 0.2\mathbb{I}[B(x, y)])\mathbb{I}[C(x)] \\ &\qquad +(0.3 + 0.4\mathbb{I}[\neg B(x, y)])\mathbb{I}[\neg C(x)].\end{aligned}$$

The change is apparently small (from $C(y)$ to $C(x)$ in g_2). Consider inference $P(A(a_1))$. AC-FOVE applies without problems: we fix $C(x)$ at true, and run a logarithmic-time procedure to eliminate $B(x, y)$; then we fix $C(x)$ at false, and run the same procedure. In fact, in this case we can run the algorithm as follows. Fix $C(x)$ at true, then note that $P(A(x)|C(x) = \text{true}) = 1 - P(B(x, y) = \text{false}|C(x) = \text{true})^n$; likewise, $P(A(x)|C(x) = \text{false}) = 1 - P(B(x, y) = \text{false}|C(x) = \text{false})^n$. Hence, $P(A(a_1)) = 0.8(1 - (0.4)^n) + 0.2(1 - (0.7)^n)$. $\Box$

However, as the next example indicates, the elimination rules in the original presentation of AC-FOVE do not successfully handle situations such as Example 2.

Example 5 Consider now Example 2 and inference $P(A(a_1))$. AC-FOVE proceeds as in the previous example [6, Section 3.4.3], by fixing the value of the context parvariable $C(y)$, and eliminating $B(x, y)$. This is obtained by recursively multiplying grounded factors; for instance, $\phi_2(B(x, a_i), C(a_i))$ and $\phi_2(B(x, a_j), C(a_j))$ with $i \neq j$. The result is a factor with variables $C(a_i)$ and $C(a_j)$; note that these are distinct variables and that imposing $C(x)$ to be true does not capture the range of possible values for this pair of variables. The problem is that each iteration of the square-and-multiply method introduces a new grounding of $C(y)$, and in the end the resulting factor will be exponential in size. $\Box$

Despite the fact that the original AC-FOVE algorithm leads to incorrect results in Example 5, we will show that it is possible to exactly compute $P(A(a_1))$ in this example without any grounding. We now develop techniques that allow this to be done correctly, and later show how to compute the desired value $P(A(a_1))$.

C. Fixing AC-FOVE

The last example shows that the elimination of parvariables from aggregation parfactors must be examined more carefully than suggested in the original AC-FOVE. In this section we identify a condition that, if imposed, eliminates the sort of problem we faced in Example 5. The condition is introduced in the following result that mimics Proposition 3.6 from Kisynski's work [6].

Theorem 1 *Suppose that in a set G of parfactors we have the aggregation parfactor $g =< L, C, r, s, V, \otimes, \phi_x >$ such that:*

- *the set of constraints C is in normal form;*
- *the set of logical variables in r and in the set of logical variables in s differ* only *in that the former set contains x_{r-s} and the latter does not;*
- *no other parfactor in G shares random variables with $\mathsf{RV}(r : C)$;*
- *no parvariable in V uses x_{r-s}.*

Then:

$$\sum_{\mathsf{RV}(r:C)} g \quad = \quad < L, C \backslash C_x, V \cup \{s\}, \phi >, \qquad (1)$$

where ϕ is produced by Pingala's algorithm; that is, ϕ is equal to ϕ_m in the following recursive procedure: ϕ_0 equal to ϕ_x for $k = 0$, and for larger k, up to m where m is the number of bits in the binary representation of n without the leftmost bit,

$$\phi_k = \begin{cases} \sum \phi_{k-1}\phi_{k-1} & \textit{if the kth bit is } 0, \\ \sum\sum \phi\phi_{k-1}\phi_{k-1} & \textit{otherwise. } \square \end{cases}$$

The proof of this result is in the Appendix.

The key (new) condition is the last one; that is, no parvariable in V uses x_{r-s}. Note that Example 5 fails exactly because logical variable y appears in $B(x, y)$, does not appear in $A(x)$, and it *does* appear in the context parvariable $C(y)$.

IV. Elimination and d-separation

A visual inspection of Figure 2 should indicate where the original AC-FOVE fails: by fixing the value of a "generic" $C(y)$, we are not fixing the value of all groundings of $C(y)$ that indeed matter for $A(x)$, as the arrows move across individuals in the domain. A visual inspection also clarifies why AC-FOVE easily solves Example 4. Figure 3 shows the Bayesian network that is obtained by grounding the original relational Bayesian network in Example 4 with common domain $\{a_1, a_2\}$, all sets of constraints empty. It is then clear that computing $P(A(a_1))$ is an easy task as every individual induces a simple standalone Bayesian network.

It might seem that only simple relational Bayesian networks such as presented in Example 4 are within the realm of the corrected AC-FOVE (that is, AC-FOVE with the corrected operation discussed in the previous section). The important point is whether the corrected AC-FOVE can indeed avoid grounding. We now show that the corrected AC-FOVE can be quite powerful when coupled with d-separation concepts. Indeed, such a combination leads to immediate inference in Example 5.

The idea of d-separation is to obtain, by graphical means and in polynomial time, all independence relations between a given variable, conditional on a given set of variables, and all other variables in a Bayesian network [1]. With d-separation one can detect in polynomial time all variables that are needed to obtain an inference; all other variables can be discarded. The basic concept of d-separation and algorithms that produce all d-separation relations have been extended to relational Bayesian networks by Taghipour et al [16]. The resulting first-order d-separation method is crucial in lifted inference.

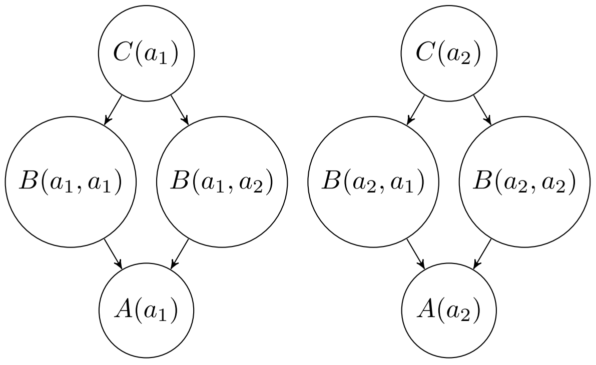

Figure 3. Bayesian network by grounding parfactors in Example 4.

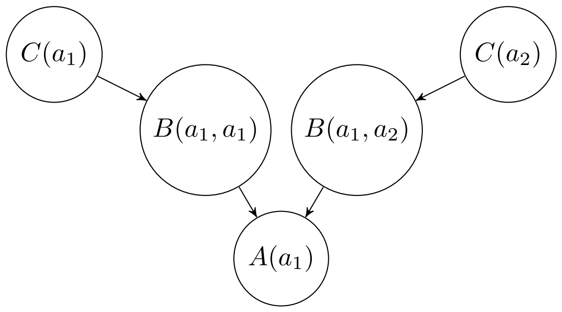

Figure 4. Bayesian network by first-order d-separation and grounding with respect to domain $\{a_1, a_2\}$ in Example 6.

Indeed, first-order d-separation must always be run during lifted inference, to discard as many parvariables as possible, so as to allow lifted operations to apply.

Example 6 Consider Example 5. Suppose first-order d-separation is run, and parvariables that do not affect the inference are discarded. The remaining parvariables, if grounded, produce the Bayesian network in Figure 4. If we have n individuals, the same process produces the grounded Bayesian network in Figure 5. We can now run AC-FOVE, eliminating $C(y)$ and then applying Theorem 1 to obtain

$$P(A(a_1)) = 1-(0.6\times0.8+0.3\times0.2)^n = 1-(0.54)^n. \ \square$$

V. Conclusion

In this paper we have presented novel results about exact lifted inference in first-order probabilistic models based on parfactors. The first contribution is the corrected version of Kisynski's theorem on lifted elimination of aggregation parfactors (Theorem 1). The second contribution is the discussion of d-separation as a technique to enhance lifted inference.

The corrected AC-FOVE algorithm was implemented in a publicly available software package.[1] The current version runs lifted inferences as described in this paper, but first-order d-separation is not integrated yet.

[1]Source code and binary executables are available at https://github.com/ftakiyama/AC-FOVE. We plan to describe this package in more detail elsewhere.

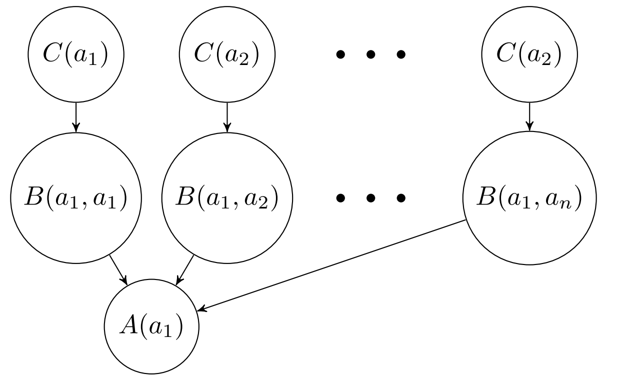

Figure 5. Bayesian network by first-order d-separation and grounding with respect to domain $\{a_1,\dots,a_n\}$ in Example 6.

Future work should explore in more detail the use of d-separation in lifted inference. Additionally, one can also investigate how to increase the expressivity of parfactors in order to derive more flexible and powerful inferences.

Acknowledgments

The first author thanks Jacek Kisynski for kindly clarifying by email several questions concerning aggregation parfactors. The second author is partially supported by CNPq. The work reported here received substantial support by FAPESP grant 2008/03995-5.

Appendix

Proof of Theorem 1: It can be shown [6] that parfactor $< L, C, r, s, V, \otimes, \phi_x >$ is equivalent to the product of two parfactors: $< L, C_x, \{r\} \cup V, \phi_x >$ and $< L \setminus \{x_{r-s}\}, C \setminus C_x, \{\#x_{r-s} : [r], s\}, \phi' >$. Once we eliminate r from these parfactors, we obtain a parfactor $< L, C \setminus C_x, V \cup \{s\}, \phi_m >$.

Let R be the set of parvariables obtained by propositionalizing r on every individual bound to x_{r-s} that satisfies constraints in C. In what follows, the symbol $\sum \phi_x$ means "sum out a parvariable $r' \in R$ from ϕ_x". Since only x_{r-s} is propositionalized and it only appears in r (last condition), every factor produced by $\sum \phi_x$ will be identical (no matter which parvariable $r' \in R$ is eliminated).

We show now that ϕ_m calculated through Pingala's algorithm produces the correct result. Let $(\phi_x)^n$ denote the multiplication of ϕ_x by itself n times. It suffices to show that

$$\phi_m = \underbrace{\sum \cdots \sum}_{n-1} (\phi_x)^n, \forall m \in \mathbb{N}. \qquad (2)$$

We do it by induction. For the base case of $n = 1$, we have $m = 0$ and thus $\phi_0 = \phi_x$. For $n = 2$, we have $m = 1$ and $\phi_1 = \sum (\phi_x)^2$. Suppose Equation (2) is true for $n = j$ and $m = k$. Then, for $n = 2j$ or $n = 2j + 1$ we have to calculate ϕ_{k+1}:

$$\phi_{k+1} = \begin{cases} \sum \phi_k \phi_k \text{ if the (k+1)th bit is 0} \\ \sum \sum \phi_x \phi_k \phi_k \text{ otherwise} \end{cases}$$

$$= \begin{cases} \sum \left(\underbrace{\sum \cdots \sum}_{n-1} (\phi_x)^n \right) \left(\underbrace{\sum \cdots \sum}_{n-1} (\phi_x)^n \right) \\ \sum \sum \phi_x \left(\underbrace{\sum \cdots \sum}_{n-1} (\phi_x)^n \right) \left(\underbrace{\sum \cdots \sum}_{n-1} (\phi_x)^n \right) \end{cases}$$

$$= \begin{cases} \underbrace{\sum \cdots \sum}_{2n-1} (\phi_x)^{2n} \\ \underbrace{\sum \cdots \sum}_{2n} (\phi_x)^{2n+1} \end{cases}$$

Thus, if Equation (2) is valid for n, then it is valid for $2n$ and $2n + 1$. Because every natural number greater than 1 can be expressed by these expressions, Expression ((2)) obtains for every natural number. □

References

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

[2] M. Jaeger, "Relational Bayesian networks," in *Conference on Uncertainty in Artificial Intelligence*, 1997, pp. 266–273.

[3] D. Koller and A. Pfeffer, "Probabilistic frame-based systems," in *National Conference on Artificial Intelligence (AAAI)*, 1998, pp. 580–587.

[4] D. Poole, "First-order probabilistic inference," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 985–991.

[5] J. Kisynski and D. Poole, "Lifted aggregation in directed first-order probabilistic models," in *Int. Joint Conf. on Artificial Intelligence*, 2009, pp. 1922–1929.

[6] J. J. Kisynski, "Aggregation and constraint processing in lifted probabilistic inference," Ph.D. dissertation, Computer Science, University of British Columbia, 2010.

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

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

[9] B. Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling, "Lifted probabilistic inference with counting formulas," in *AAAI*, 2008, pp. 1062–1068.

[10] N. Taghipour and J. Davis, "Generalized counting for lifted variable elimination," in *International Workshop on Statistical Relational AI*, 2012.

[11] N. Taghipour, "Lifted probabilistic inference by variable elimination," Ph.D. dissertation, KU Leuven, 2013.

[12] R. de Salvo Braz, E. Amir, and D. Roth, "Lifted first-order probabilistic inference," in *International Joint Conference in Artificial Intelligence (IJCAI)*, 2006.

[13] ——, "MPE and partial inversion in lifted probabilistic variable elimination," in *AAAI*, 2006.

[14] ——, "Lifted first-order probabilistic inference," in *An Introduction to Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. MIT Press, 2007, pp. 433–451.

[15] ——, "A survey of first-order probabilistic models," in *Innovations in Bayesian Networks*, ser. Studies in Computational Intelligence, Springer, 2008, pp. 289–317.

[16] N. Taghipour, W. Meert, J. Struyf, and H. Blockeel, "First-order Bayes-ball for CP-logic," in *Workshop on Statistical Relational Learning*, Leuven, Belgium, 2009, pp. 1–3.