Lifted Probabilistic Inference with Counting Formulas

Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, Leslie Pack Kaelbling
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139, USA
{milch,lsz,kersting,mhaimes,lpk}@csail.mit.edu

Abstract

Lifted inference algorithms exploit repeated structure in probabilistic models to answer queries efficiently. Previous work such as de Salvo Braz *et al.*'s first-order variable elimination (FOVE) has focused on the sharing of potentials across interchangeable random variables. In this paper, we also exploit interchangeability within individual potentials by introducing *counting formulas*, which indicate how many of the random variables in a set have each possible value. We present a new lifted inference algorithm, C-FOVE, that not only handles counting formulas in its input, but also creates counting formulas for use in intermediate potentials. C-FOVE can be described succinctly in terms of six operators, along with heuristics for when to apply them. Because counting formulas capture dependencies among large numbers of variables compactly, C-FOVE achieves asymptotic speed improvements compared to FOVE.

Introduction

Models expressed in relational probabilistic languages (Getoor & Taskar 2007) compactly define joint distributions for large numbers of random variables. For example, suppose we have invited 50 people to a workshop, and want to reason about how many will attend. We might believe that the attendance variables all depend on a variable indicating whether our workshop topic is "hot" this year. A relational language allows us to specify parameters that apply to $\mathsf{attends}(\mathsf{X})$ and $\mathsf{topicHot}$ for all invitees X, rather than specifying the same parameters 50 times. The model represents the fact that the $\mathsf{attends}(\mathsf{X})$ variables are interchangeable.

Algorithms that exploit such interchangeability to answer queries more efficiently are called *lifted* inference algorithms (Pfeffer *et al.* 1999). The state of the art in exact lifted inference is the first-order variable elimination (FOVE) algorithm of de Salvo Braz *et al.* (2007), which extends earlier work by Poole (2003). FOVE performs variable elimination on a set of parameterized factors, or *parfactors*. Each parfactor specifies a set of factors with the same numeric values. For instance $\forall \mathsf{X}.\phi(\mathsf{attends}(\mathsf{X}), \mathsf{topicHot})$ represents a factor for each invitee X. FOVE performs *lifted* elimination to sum out entire families of random variables such as $\mathsf{attends}(\mathsf{X})$ with a single computation.

In this paper, we extend the parfactor representation to capture not just repeated factors, but also interchangeability of random variables within factors. This symmetry is captured by *counting formulas*: for instance, the parfactor $\phi(\#_{\mathsf{X}}[\mathsf{attends}(\mathsf{X})], \mathsf{topicHot})$ includes a formula that counts the number of invitees X who attend, and encodes a dependency between this count and the popularity of the workshop. Here it is the $\mathsf{attends}$ variables that are interchangeable: the weight specified by the factor depends only on how many people attend, not which particular ones attend. To represent the same dependency without a counting formula would require a factor on all of the $\mathsf{attends}$ variables, which would be exponentially large in the number of invitees.

We present an extension of the FOVE algorithm, called C-FOVE, that performs variable elimination on parfactors with counting formulas. C-FOVE is built upon six basic operators that have simple correctness conditions. The most interesting operator is one that creates new counting formulas to compactly represent new dependencies introduced during variable elimination. C-FOVE chooses automatically when to apply these operators, using a heuristic method that seeks to minimize the size of intermediate parfactors. As we will see in the evaluation, these advances enable C-FOVE to sometimes perform inference in time linear in the number of random variables where FOVE requires exponential time.

We proceed as follows. We begin by introducing parfactors and counting formulas. Then, after providing a brief overview of C-FOVE, we describe each of its six operations in detail and present our experimental evaluation. Before concluding, we discuss related work.

Representation

We consider models where each random variable corresponds to a ground atom of the form $p(c_1, \ldots, c_n)$, where p is a predicate and $c_1, \ldots, c_n$ are constant symbols. Each predicate p returns a value in some finite range $range(p)$. More generally, an *atomic formula* has the form $p(t_1, \ldots, t_n)$ where each t_i is a *term*, that is, a constant symbol or a logical variable. Each logical variable X takes values of a particular type; we will write $dom(X)$ for the set of constant symbols of this type, and $|X|$ for $|dom(X)|$. We use $\mathbf{v}$ to denote an assignment of values to random variables, and $\alpha(\mathbf{v})$ to denote the value of a ground atom α in $\mathbf{v}$.

We define a probability distribution on the set of random

variables using a set of *factors*. A factor is a pair $f = (A, \phi)$ where A is a list of ground formulas $(\alpha_1, \ldots, \alpha_m)$ and ϕ is a *potential* on A: that is, a function from $range\,(A) = \times_{i=1}^{m}(range\,(\alpha_i))$ to non-negative real numbers. A factor f defines a weighting function on instantiations: $w_f(\mathbf{v}) = \phi(\alpha_1(\mathbf{v}), \ldots, \alpha_m(\mathbf{v}))$. A set of factors F defines a probability distribution proportional to $w_F(\mathbf{v}) = \prod_{f \in F} w_f(\mathbf{v})$.

A *substitution* θ for a set of logical variables L maps each variable $X \in L$ to a term. The result of applying a substitution θ to a formula α is denoted $\alpha\theta$. We will use $gr\,(L)$ to denote the set of *ground substitutions* that map all the variables in L to constants. We will also consider restricted sets of groundings $gr\,(L : C)$, where C is a *constraint* consisting of inequalities that include logical variables in L and constant symbols.

Parfactors

A *parfactor* (Poole 2003) compactly describes a set of factors. Each parfactor has the form (L, C, A, ϕ) where L is a set of logical variables, C is a constraint on L, A is a list of formulas, and ϕ is a potential on A. We will often write parfactors in the syntax $\forall X_1, \ldots, X_n : C\,.\,\phi(\alpha_1, \ldots, \alpha_m)$, such as $\forall \mathsf{X}.\phi_1(\mathsf{attends}(\mathsf{X}), \mathsf{topicHot})$.

Applying a substitution θ to a parfactor $g = (L, C, A, \phi)$ yields a new parfactor $g\theta = (L', C\theta, A\theta, \phi)$, where L' is obtained by renaming the variables in L according to θ and dropping those that are mapped to constants. If θ is a ground substitution, then $g\theta$ is a factor. The set of *groundings* of a parfactor $g = (L, C, A, \phi)$ is the set of factors $gr\,(g) = \{g\theta : \theta \in gr\,(L : C)\}$. The weighting function for a set of parfactors G is a product over the groundings of all the parfactors in G:

$$w_G(\mathbf{v}) = \prod_{g \in G} \prod_{f \in gr(g)} w_f(\mathbf{v}).$$

Finally, for an atom α and a constraint C, we define the set of random variables $\mathrm{RV}\,(\alpha : C) = \{\alpha\theta : \theta \in gr\,(L : C)\}$. For a parfactor $g = (L, C, A, \phi)$, $\mathrm{RV}\,(g) = \bigcup_{(\alpha \in A)} \mathrm{RV}\,(\alpha : C)$. When we are referring to an occurrence of an atom α in a parfactor (L, C, A, ϕ), we will write $\mathrm{RV}\,(\alpha)$ as an abbreviation for $\mathrm{RV}\,(\alpha : C)$.

Counting Formulas

We extend previous work by allowing parfactors to contain *counting formulas*, which are useful for cases where the value of the parfactor depends only on the number of random variables having each possible value. For instance, in our meeting example, suppose we are interested in a random variable $\mathsf{overflow}$ that is true when the number of people attending our workshop is greater than 35. We would like to avoid representing this dependency with an exponentially large factor on $\mathsf{overflow}, \mathsf{attends}(\mathsf{p}_1), \ldots, \mathsf{attends}(\mathsf{p}_n)$. Because the $\mathsf{attends}(\mathsf{X})$ variables are not independent given $\mathsf{overflow}$, we cannot use the parfactor $\forall \mathsf{X}.\phi(\mathsf{overflow}, \mathsf{attends}(\mathsf{X}))$.

The counting formula $\#_{\mathsf{X}}\,[\mathsf{attends}(\mathsf{X})]$, however, can be used to represent this dependency. The possible values of this counting formula are *histograms*, counting how many ground substitutions for X yield each possible value for $\mathsf{attends}(\mathsf{X})$. If $range\,(\mathsf{attends}) = \{\mathsf{true}, \mathsf{false}\}$, then each possible histogram has two buckets, one for true and one for false. The sum of the values in these two buckets is the domain size $|\mathsf{X}|$. So if $|\mathsf{X}| = 50$, then there are 51 possible histograms: $(0, 50), (1, 49), \ldots, (50, 0)$. We can represent our example using a factor $\phi'(\mathsf{overflow}, \#_{\mathsf{X}}\,[\mathsf{attends}(\mathsf{X})])$, where ϕ' is a potential of size 2×51 whose value depends on whether there are more than 35 entries in the true bucket.

The general form of a counting formula is $\#_{X:C}\,[\alpha]$, where X is a logical variable that is *bound* by the $\#$ sign, C is a constraint on X, and α is an atom containing X. The constraint and atom can also contain *free* logical variables, such as the variable Y in $\#_{\mathsf{X}:\mathsf{X} \neq \mathsf{Y}}\,[\mathsf{knows}(\mathsf{X}, \mathsf{Y})]$.

Definition 1 *Let* $\mathbf{v}$ *be an instantiation of the random variables, and let* γ *be a counting formula* $\#_{X:C}\,[\alpha]$ *with no free variables. Then* value *of* γ *on* $\mathbf{v}$ *is the histogram* $h : range\,(\alpha) \to \mathbb{N}$ *such that for each value* $u \in range\,(\alpha)$*,*

$$h(u) = |\{\theta \in gr\,(X : C) : \alpha\theta(\mathbf{v}) = u\}|$$

The range of a counting formula $\gamma = \#_{(X:C)}\,[\alpha]$, denoted $range\,(\gamma)$, is the set of histograms having a bucket for each element of $range\,(\alpha)$ with entries adding up to $size\,(X : C)$.[1] The set of random variables that γ depends on is $\mathrm{RV}\,(\gamma) = \bigcup_{(\theta \in gr(X:C))} \mathrm{RV}\,(\alpha\theta)$.

Normal-Form Constraints

The range of a counting formula $\#_{X:C}\,[\alpha]$ consists of histograms whose entries add up to $size\,(X : C)$. However, when C contains free variables, the number of allowed groundings of X may depend on the values these free variables take on. For instance, in the parfactor $\forall \mathsf{Y}.\phi(\mathsf{famous}(\mathsf{Y}), \#_{\mathsf{X}:(\mathsf{X} \neq \mathsf{a}, \mathsf{X} \neq \mathsf{Y})}\,[\mathsf{knows}(\mathsf{X}, \mathsf{Y})])$, the number of allowed values for X is $|\mathsf{X}| - 2$ for most values of Y, but it is $|\mathsf{X}| - 1$ when $\mathsf{Y} = \mathsf{a}$.

To avoid such difficulties, we require the constraints in each parfactor to be in a certain normal form. Given a constraint C, let E_X^C be the *excluded set* for X, that is, the set of terms t such that $(X \neq t) \in C$.

Definition 2 *Let* g *be a parfactor with constraint* C*, and let* C^* *be the set of inequalities formed by taking the union of* C *with the constraints in all the counting formulas in* g*. Then* g *is in* normal form *if, for each logical variable inequality* $X \neq Y$ *in* C^**, there is a common excluded set* E *such that* $E_X^{C^*} = E \cup \{Y\}$ *and* $E_Y^{C^*} = E \cup \{X\}$*.*

It can be shown that if a parfactor is in normal form, then $size\,(X : C)$ has a fixed value, namely $|X| - |E_X^C|$, regardless of the binding to the free variables in C. We can convert any parfactor to normal form by splitting it: for instance, the parfactor above can be broken into the parfactor $\forall \mathsf{Y} : \mathsf{Y} \neq \mathsf{a}.\ \phi(\mathsf{famous}(\mathsf{Y}), \#_{\mathsf{X}:(\mathsf{X} \neq \mathsf{a}, \mathsf{X} \neq \mathsf{Y})}\,[\mathsf{knows}(\mathsf{X}, \mathsf{Y})])$ and the factor $\phi(\mathsf{famous}(\mathsf{a}), \#_{\mathsf{X}:\mathsf{X} \neq \mathsf{a}}\,[\mathsf{knows}(\mathsf{X}, \mathsf{a})])$. This normal form eliminates the need to use a separate constraint solver, as in de Salvo Braz *et al.* (2007).

[1] If $|\,range\,(\alpha)\,| = r$ and $size\,(X : C) = n$, then the number of such histograms is given by a function called "r multichoose n", which is equal to $\binom{n+r-1}{r-1}$. When r is fixed and the domain size n grows, this number is exponentially smaller than the number of assignments to the random variables, which is r^n.

Inference Overview

Given a set of parfactors G, we want to compute the marginal of the weighting function w_G on a set of query random variables.

Definition 3 *Let G be a set of parfactors, and $\mathcal{V} \subset \mathrm{RV}(G)$ be a set of RVs. The* marginal *of w_G on $\mathcal{V}$, denoted $w_{G|\mathcal{V}}$, is defined for each assignment $\mathbf{v} \in range(\mathcal{V})$ as:*

$$w_{G|\mathcal{V}}(\mathbf{v}) = \sum_{\mathbf{u} \in range(\mathrm{RV}(G) \setminus \mathcal{V})} w_G(\mathbf{v}; \mathbf{u}).$$

If the model G is simply a set of factors on ground atoms, we can compute the desired marginal using the variable elimination (VE) algorithm (Zhang & Poole 1994). VE performs a sequence of sum-out operations, each of which eliminates a single random variable. Our C-FOVE algorithm deals with parfactors and counting formulas using six operators: a *lifted elimination* operator that can sum out a whole family of random variables, and five other operators that modify the parfactor set G—without changing the weighting function—so that the elimination operator can be applied. The algorithm continues applying operators to G until only the query variables remain in $\mathrm{RV}(G)$.

Inference Operators

This section describes the operators used by the C-FOVE algorithm and gives conditions under which they are correct. The next section gives a method for choosing the sequence of operations to perform for a given inference problem.

Lifted Elimination

We begin by defining an operation $\textsc{Sum-Out}(g_i, \alpha_{i,j})$ that takes a single parfactor $g_i = (L_i, C_i, A_i, \phi_i)$ and sums out all the random variables covered by its j-th formula $\alpha_{i,j}$. Before defining the conditions where this operation is correct, we consider exactly what it does. If $\alpha_{i,j}$ is a ground atom, we create a new potential ϕ_i' where the j-th dimension has been summed out and remove $\alpha_{i,j}$ from A_i'. This is the same operation as summing out random variables in VE. If $\alpha_{i,j}$ is an atom that contains logical variables, we apply exactly the same operations as in the ground case. We will discuss why this is correct shortly, but for now it is enough to observe that we are avoiding the repeated sums that would be required if we first grounded and then eliminated. For counting formulas, we would like to do basically the same operation (sum dimension j out of ϕ_i). However, remember that each histogram $h \in range(\alpha_{i,j})$ is used as an index into ϕ_i for a whole set of instantiations of the underlying random variables. As we sum the table entries, we must also multiply each entry by the number of instantiations that map to it. We can compute this number as follows:

Proposition 1 *Let (L, C, A, ϕ) be a normal-form parfactor and α be a formula in A. Then for any ground substitution θ and value $u \in range(\alpha)$, the number of assignments $\mathbf{v}$ to $\mathrm{RV}(\alpha\theta)$ such that $(\alpha\theta)(\mathbf{v}) = u$ is a fixed function $\textsc{Num-Assign}(\alpha, u)$, independent of θ. If α is an atom, then $\textsc{Num-Assign}(\alpha, u) = 1$. If α is a counting formula $\#_{X:D}[\beta]$, then for any histogram $h \in range(\alpha)$,*

$$\textsc{Num-Assign}(\alpha, h) = \frac{size(X : D)!}{\prod_{(u' \in range(\beta))} h(u')!}$$

where $h(u')$ is the count for the value u' in histogram h.

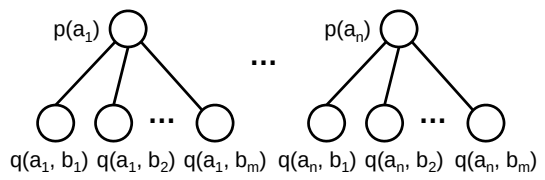

Figure 1: Markov net defined by $\forall \mathsf{X}, \mathsf{Y}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}, \mathsf{Y}))$ with $dom(\mathsf{X}) = \{\mathsf{a}_1, \ldots, \mathsf{a}_n\}$ and $dom(\mathsf{Y}) = \{\mathsf{b}_1, \ldots, \mathsf{b}_m\}$.

Now we can formally define the SUM-OUT operator and the conditions under which it yields the correct marginal:

Proposition 2 *Let G be a set of parfactors $\{g_1, \ldots, g_n\}$, where $g_i = (L_i, C_i, A_i, \phi_i)$ and $A_i = (\alpha_{i,1}, \ldots, \alpha_{i,m_i})$. Suppose that for $\alpha_{i,j}$:*

(S1) *All logical variables in L_i occur as arguments in $\alpha_{i,j}$; and*

(S2) *For all other formulas $\alpha_{k,l}$ with $(k,l) \neq (i,j)$, $\mathrm{RV}(\alpha_{i,j}) \cap \mathrm{RV}(\alpha_{k,l}) = \emptyset$.*

Let $\textsc{Sum-Out}(G, \alpha_{i,j})$ be the same as G except that g_i is replaced with $g_i' = (L, C, A', \phi')$, where A' is A with $\alpha_{i,j}$ removed, and for each $\mathbf{v} \in range(A')$,

$$\phi'(\mathbf{v}) = \sum_{u \in range(\alpha_{i,j})} \textsc{Num-Assign}(\alpha_{i,j}, u) \times \phi(v_1, \ldots, v_{j-1}, u, v_j, \ldots, v_{m_i - 1}).$$

Then $w_{\textsc{Sum-Out}(G, \alpha_{i,j})} = w_{G|(\mathrm{RV}(G) \setminus \mathrm{RV}(\alpha_{i,j}))}$.

Condition (S2) guarantees that the random variables $\mathrm{RV}(\alpha_{i,j})$ that we are summing out are completely eliminated from the set of parfactors. If (S2) does not hold, we may be able to apply a multiplication operator (defined below) to create a larger parfactor where it does.

Condition (S1) guarantees the operation is correct when we sum out formulas with free variables. As an example, consider the parfactor $\forall \mathsf{X}, \mathsf{Y}.\ \phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}, \mathsf{Y}))$. Fig. 1 shows the Markov network defined by this parfactor. To sum out the random variables $\mathrm{RV}(\mathsf{q}(\mathsf{X}, \mathsf{Y}))$, we could create this ground network and sum out each variable $\mathsf{q}(\mathsf{a}_i, \mathsf{b}_j)$ from each factor $\phi(\mathsf{p}(\mathsf{a}_i), \mathsf{q}(\mathsf{a}_i, \mathsf{b}_j))$ in turn. Because these factors all come from the same parfactor, we would get the same resulting factor $\phi'(\mathsf{p}(\mathsf{a}_i))$ each time. We can obtain the same result in a single step by summing $\mathsf{q}(\mathsf{X}, \mathsf{Y})$ out of the original parfactor to create a new parfactor $\forall \mathsf{X}, \mathsf{Y}.\ \phi'(\mathsf{p}(\mathsf{X}))$.

However, if we tried to sum out $\mathsf{p}(\mathsf{X})$—which does not contain all the logical variables in the parfactor—the computation would be incorrect. Summing out each $\mathsf{p}(\mathsf{a}_i)$ variable creates a joint potential on its m neighbors, the variables $\mathsf{q}(\mathsf{a}_i, \mathsf{b}_j)$, that cannot be represented as a parfactor of the form $\forall \mathsf{X}, \mathsf{Y}.\ \phi'(\mathsf{q}(\mathsf{X}, \mathsf{Y}))$. As we will see shortly, the potential that is created in this case depends only on the number of Y such that $\mathsf{q}(\mathsf{X},\mathsf{Y})$, and can be represented with a counting formula. Once we introduce operations that make counting formulas, we will be able to sum out $\mathsf{p}(\mathsf{X})$.

Elimination-Enabling Operators

In general, there is no guarantee that the conditions for lifted elimination will be satisfied for any of the formulas in the parfactors. The operators described in this section change the current set of parfactors G to a new set G' that may have more opportunities for lifted elimination, while preserving the weighting function $w_G = w_{G'}$.

Counting The counting operator introduces counting formulas into parfactors. Although these formulas increase the size of the new parfactors, they are useful because they create opportunities for lifted elimination. For example, recall our discussion of the parfactor $\forall \mathsf{X}, \mathsf{Y}.\ \phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}, \mathsf{Y}))$ in Fig. 1. We could not sum out $\mathsf{p}(\mathsf{X})$ because doing so would introduce dependencies between the q random variables. The counting operator allows us to first introduce a counting formula on Y, to create a new parfactor $\forall \mathsf{X}.\phi'(\mathsf{p}(\mathsf{X}), \#_{\mathsf{Y}}[\mathsf{q}(\mathsf{X}, \mathsf{Y})])$. We can now sum out $\mathsf{p}(\mathsf{X})$ because (S1) is satisfied: we have, in essence, induced exactly the new dependencies that would have been formed by summing out $\mathsf{p}(\mathsf{X})$ from the original parfactor. In general, we introduce counting formulas as follows:

Proposition 3 *Let $g \in G$ be a normal-form parfactor (L, C, A, ϕ) with $A = (\alpha_1, \ldots, \alpha_m)$ and $X \in L$ such that there is exactly one formula $\alpha_i \in A$ where X occurs free, and α_i is an atom. Let g' be a new parfactor $(L \setminus \{X\}, C^{\downarrow}_{L \setminus \{X\}}, A', \phi')$ where A' is the same as A except that $\alpha'_i = \#_{X : C^{\downarrow}_{\{X\}}}[\alpha_i]$. For $\mathbf{v} \in$ range (A'), let:*

$$\phi'(\mathbf{v}) = \prod_{u \in range(\alpha_i)} \phi(v_1, \ldots, v_{i-1}, u, v_{i+1}, \ldots, v_m)^{v_i(u)}.$$

Then $w_G = w_{G \setminus \{g\} \cup \{g'\}}$.

Here $C^{\downarrow}_{L'}$ denotes the constraint C restricted to inequalities involving variables in L'. The equation for $\phi'(\mathbf{v})$ is obtained by thinking of w_g as a product over all groundings of X, and partitioning those groundings according to the value u that they yield for α_i. The weight for groundings where $\alpha_i = u$ is raised to the power $v_i(u)$, which is the number of such groundings according to the histogram v_i.

Exponentiation The exponentiation operator removes a logical variable X from a parfactor (L, C, A, ϕ) when X does not occur free in any formula in A. For instance, the logical variable Y can be removed from $\forall \mathsf{X}, \mathsf{Y}.\ \phi(\mathsf{p}(\mathsf{X}))$. Such parfactors are created by the SUM-OUT operator, as we saw in the previous section.

When we remove such a logical variable, we get a parfactor with fewer groundings: in our example, removing Y decreases the number of groundings from $|\mathsf{X}| \times |\mathsf{Y}|$ to $|\mathsf{X}|$. Thus, to preserve the same weighting function, we need to raise the entries in ϕ to the power $|\mathsf{Y}|$.

Proposition 4 *Let $g \in G$ be a normal-form parfactor (L, C, A, ϕ). Let X be a logical variable in L that does not occur free in any formula in A. Let $L' = L \setminus \{X\}$. Let $g' = (L', C^{\downarrow}_{L'}, A, \phi')$ where for $\mathbf{v} \in$ range (A),*

$$\phi'(\mathbf{v}) = \phi(\mathbf{v})^{size(X : C)}.$$

Then $w_G = w_{G \setminus \{g\} \cup \{g'\}}$.

Multiplying parfactors One reason we might not be able to sum out a formula α is that RV (α) overlaps with the random variables of another formula in a different parfactor. In this case, we need to multiply the parfactors:

Proposition 5 *Let $g_1, \ldots, g_n \in G$ be parfactors with the same logical variables L and constraint C, where $g_i = (L, C, A_i, \phi_i)$ and $A_i = (\alpha_{i,1}, \ldots, \alpha_{i,m_i})$. Then define a new parfactor* MULTIPLY $(g_1, \ldots, g_n) = (L, C, A', \phi')$ *as follows. Let $A' = \bigcup_{i=1}^{n} A_i$. For each $i \in \{1, \ldots, n\}$ and $\ell \in \{1, \ldots, m_i\}$, let $j(i, \ell)$ be the index in A' of $\alpha_{i,\ell}$. For each $\mathbf{v} \in$ range (A'), let*

$$\phi'(\mathbf{v}) = \prod_{i=1}^{n} \phi_i(v_{j(i,1)}, \ldots, v_{j(i,m_i)}).$$

Then $w_G = w_{G \setminus \{g_1, \ldots, g_n\} \cup \{\text{MULTIPLY}(g_1, \ldots, g_n)\}}$.

In the next section, where we describe operator ordering, we will see how multiplication is used in practice.

Splitting As we will see later in the paper, there are cases where we need to split a parfactor g into two parfactors g'_1 and g'_2 such that $gr(g'_1) \cup gr(g'_2) = gr(g)$. Specifically, we will split g on a variable binding $X = t$, where X is a logical variable in g and t is either a constant or another logical variable. For example, we could split $\forall \mathsf{X}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}))$ on $\mathsf{X} = \mathsf{a}$ to yield $\phi(\mathsf{p}(\mathsf{a}), \mathsf{q}(\mathsf{a}))$ and $\forall \mathsf{X} : \mathsf{X} \neq \mathsf{a}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}))$. We could also split $\forall \mathsf{X}, \mathsf{Y}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{Y}))$ on $\mathsf{X} = \mathsf{Y}$, yielding $\forall \mathsf{Y}.\phi(\mathsf{p}(\mathsf{Y}), \mathsf{q}(\mathsf{Y}))$ and $\forall \mathsf{X}, \mathsf{Y} : \mathsf{X} \neq \mathsf{Y}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{Y}))$.

Proposition 6 *Let $g \in G$ be a parfactor (L, C, A, ϕ), X be a logical variable in L, and t be a term such that $t \notin E^C_X$. Define new parfactors $g'_{=}$ and $g'_{\neq}$ such that $g'_{=} = g[X/t]$ and $g'_{\neq} = (L, C \cup \{X \neq t\}, A, \phi)$. Then $w_G = w_{G \setminus \{g\} \cup \{g'_{=}, g'_{\neq}\}}$.*

In general, we want to split as little as possible. However, there are situations where no other operators are applicable. If necessary, we can eliminate a logical variable X from a parfactor by splitting on $X = a$ for every $a \in gr(X : C)$.

Expanding counting formulas One final operator is useful when a formula counts a set of random variables that partially overlap with another formula. To be able to sum out these variables, we must create a new parfactor where the counting formula is *expanded* into a counting formula with an additional constraint, and an atom. For example, given factors $\phi_1(\#_{\mathsf{X}}[\mathsf{p}(\mathsf{X})])$ and $\phi_2(\mathsf{p}(\mathsf{c}))$, we can convert the first factor to an expanded version $\phi'_1(\mathsf{p}(\mathsf{c}), \#_{\mathsf{X} : \mathsf{X} \neq \mathsf{c}}[\mathsf{p}(\mathsf{X})])$ so that we can multiply and sum out $\mathsf{p}(\mathsf{c})$.

Proposition 7 *Let $g \in G$ be a parfactor (L, C, A, ϕ) with $A = (\alpha_1, \ldots, \alpha_m)$, where $\alpha_i = \#_{X : D}[\beta]$. Let t be a term that is not in E^D_X and is in E^C_Y for each logical variable $Y \in E^D_X$. Now let $g' = (L, C, A', \phi')$ where A' includes all the formulas in A except α_i, plus two additional formulas $\alpha'_{=} = \beta[X/t]$ and $\alpha'_{\neq} = \#_{X : (D \cup (X \neq t))}[\beta]$. For each $\mathbf{v} \in$ range (A'), let*

$$\phi'(\mathbf{v}) = \phi(v_1, \ldots, v_{i-1}, h(\mathbf{v}), v_{i+1}, \ldots, v_m)$$

where $h(\mathbf{v})$ is the histogram obtained by taking the histogram $\alpha'_{\neq}(\mathbf{v})$ and adding 1 to the count for the value $\alpha'_{=}(\mathbf{v})$. Then $w_G = w_{G \setminus \{g\} \cup \{g'\}}$.

As with splitting, we can fully expand a counting formula by expanding it on $X = a$ for every $a \in gr(X : D)$.

The C-FOVE Algorithm

Now that we have the basic operations, we can combine them to form a complete lifted variable elimination algorithm. We assume that we are given a set of parfactors G representing the model; observed values for certain ground atoms; and some query ground atoms. First, for each evidence atom α_i with observed value v_i, we add a factor $\phi(\alpha_i)$ to G, where ϕ assigns weight 1 to v_i and weight 0 to all other values. The weighting function $w_G(\mathbf{v})$ defined by the resulting set of parfactors is proportional to the conditional probability of $\mathbf{v}$ given the evidence. Our task, then, is to sum out all the random variables in the model except those corresponding to the query atoms.

We do this by iteratively modifying G using the operations described above. The problem of choosing an efficient sequence of operations is a generalization of the elimination ordering problem from standard VE, which is NP-hard. Our system attempts to find a good ordering using greedy search. At each step of the algorithm, we generate a set of operations that can be applied to the current set of parfactors; assign each one a cost; and execute the lowest-cost operation. We continue until G is just a set of factors on the query atoms.

Summing Out Random Variables Globally

To help our greedy search algorithm find better orderings, we let the set of available operations consist not of the basic operations defined above, but of *macro-operations* that bundle together a sequence of basic operations. The most important macro-operator is GLOBAL-SUM-OUT $(\alpha : C)$, where α is a formula and C is a constraint. When applied to a set of parfactors G, GLOBAL-SUM-OUT $(\alpha : C)$ combines multiplication, summing out from a single parfactor, and exponentiation so that the resulting set of parfactors represents the marginal of w_G on the random variables not in RV $(\alpha : C)$.

First, GLOBAL-SUM-OUT $(\alpha : C)$ identifies the parfactors $\{g_1, \ldots, g_n\}$ in G whose random variables overlap with RV $(\alpha : C)$. In order for the operation to be applicable, each of these parfactors g_i must contain only one formula occurrence α_i that overlaps with RV $(\alpha : C)$, and all the logical variables in g_i must occur as arguments in α_i. Then for each $i \in \{1, \ldots, n\}$, the operation attempts to construct a variable renaming θ_i — that is, a one-to-one substitution from variables to variables — that makes α_i equal to α and makes g_i's constraint equal to C. If such a substitution exists, then it is unique: each logical variable X in g_i occurs at some position in α_i's argument list, so $\theta_i(X)$ must be the variable at the same position in α's argument list.

If the desired variable renamings $\theta_1, \ldots, \theta_n$ exist, then GLOBAL-SUM-OUT $(\alpha : C)$ is applicable. It first replaces each g_i with $g_i\theta_i$, which does not change the weighting function because the renamings are one-to-one. The resulting parfactors $g_1\theta_1, \ldots, g_n\theta_n$ all have the same logical variables, so the multiplication operation of Prop. 5 can be applied to yield a product parfactor g. The applicability constraints for GLOBAL-SUM-OUT $(\alpha : C)$ ensure that the conditions of Prop. 2 are satisfied. So the operation can sum out α from g to get the desired marginal. If any of the logical variables in α no longer occur in the resulting parfactor, they are removed using the exponentiation operation (Prop. 4).

Shattering

Suppose our model contains a parfactor $\forall \mathsf{X}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}))$, and we have a deterministic evidence factor on the ground atom $\mathsf{p}(\mathsf{c})$. We cannot apply GLOBAL-SUM-OUT $(\mathsf{p}(\mathsf{X}))$ or GLOBAL-SUM-OUT $(\mathsf{p}(\mathsf{c}))$, because $\mathsf{p}(\mathsf{X})$ and $\mathsf{p}(\mathsf{c})$ depend on overlapping random variables. In order to sum these variables out, we need to split the first parfactor on $\mathsf{X} = \mathsf{c}$, yielding $\phi(\mathsf{p}(\mathsf{c}), \mathsf{q}(\mathsf{c}))$ and $\forall \mathsf{X} : \mathsf{X} \neq \mathsf{c}.\phi(\mathsf{p}(\mathsf{X}), \mathsf{q}(\mathsf{X}))$. We can then apply GLOBAL-SUM-OUT $(\mathsf{p}(\mathsf{c}))$ and GLOBAL-SUM-OUT $(\mathsf{p}(\mathsf{X}))$. Similarly, if we have a parfactor $\phi(\#_{\mathsf{X}}[\mathsf{p}(\mathsf{X})])$ and evidence on $\mathsf{p}(\mathsf{c})$, we need to expand the counting formula to get $\phi'(\mathsf{p}(\mathsf{c}), \#_{(\mathsf{X}:\mathsf{X}\neq\mathsf{c})}[\mathsf{p}(\mathsf{X})])$.

Thus, C-FOVE needs to perform splits and expansions in order to handle observations (and queries) that break symmetries among random variables. Following Poole (2003), we define a *shattering* operation. Shattering performs all the splits (and, in our case, expansions) that are necessary to ensure that for any two formula occurrences α_1 and α_2 in the parfactor set G and the query set, either RV (α_1) = RV (α_2) or RV $(\alpha_1) \cap$ RV $(\alpha_2) = \emptyset$. We also do additional splits and expansions in some cases to ensure that the constraints are in normal form. As in de Salvo Braz *et al.* (2007), our algorithm performs shattering at the beginning of elimination.

Our algorithm for shattering is essentially the same as that of Poole (2003). Two changes are necessary to deal with counting formulas. First, Poole determines whether RV $(\alpha_1 : C_1)$ and RV $(\alpha_2 : C_2)$ overlap by seeing if α_1 and α_2 are unifiable (and if applying the most general unifier to C_1 and C_2 yields no contradictions). This is not correct for comparing an atom and a counting formula, which may have overlapping random variables even though there is no substitution that makes them equal. Thus, when we check for overlaps with a counting formula $\gamma = \#_{X:D}[\beta]$, we apply unification to the atom β rather than γ itself. When checking constraints, we combine the counting constraint D with the constraint from the parfactor where γ occurs. Second, Poole's algorithm eliminates partial overlaps by repeatedly splitting parfactors on variable bindings of the form $X = a$ or $X = Y$. In our case, the algorithm may yield variable bindings that apply to the bound variable in a counting formula. When this occurs, we expand the counting formula.

Choosing an Operation

After the initial shattering, our greedy-search implementation of C-FOVE uses four macro-operators. The first is GLOBAL-SUM-OUT (α, C), which we discussed above. At each step, we consider all the valid GLOBAL-SUM-OUT operations for formulas that occur in the current parfactor set. The second operator is COUNTING-CONVERT(g, X), where g is a parfactor and X is a logical variable that occurs only in a single atom in g. This operator performs the conversion described in Prop. 3. The next operator is PROPOSITIONALIZE(g, X), which takes a parfactor g

(with constraint C) and a logical variable X, and splits g on $X = a$ for every constant $a \in gr(X : C)$. The operator then invokes the shattering procedure to eliminate any partial overlaps with other parfactors. Finally, there is FULL-EXPAND(g, γ), which takes a parfactor g and a counting formula $\gamma = \#_{X:D}[\beta]$, and expands γ on $X = a$ for every $a \in gr(X : D)$. Like PROPOSITIONALIZE, FULL-EXPAND performs shattering.

We define a cost for each operation by computing the total size of all the potentials that the operation creates. GLOBAL-SUM-OUT reduces the size of a potential in its summing-out phase, but its multiplication phase can sometimes create a large potential. As we will see in our experiments, a COUNTING-CONVERT operation is sometimes cheaper, even though it increases the size of a potential. PROPOSITIONALIZE and FULL-EXPAND are operations of last resort.

Completeness and Complexity

The C-FOVE operators are always sufficient for summing out the non-query variables: in the worst case, we can propositionalize all the parfactors and fully expand all the counting formulas, so that G becomes a set of factors on ground atoms. Then the standard VE algorithm can be executed by applying GLOBAL-SUM-OUT to ground atoms. C-FOVE can also replicate any elimination ordering used by FOVE: we can replicate FOVE's elimination operation by applying PROPOSITIONALIZE or COUNTING-CONVERT until a target atom α can be summed out, and then summing out α and all the other affected formulas. The use of counting formulas allows us to separate COUNTING-CONVERT from the summing-out operation, which gives us greater flexibility in choosing elimination orderings.

The theoretical complexity of inference on a graphical model is usually measured by its *treewidth* — a graph-theoretic concept that determines the size of the largest potential created while executing the best possible sequence of sum-out operations (Dechter 1999). An appropriate measure in the lifted setting is the size of the largest potential created by the optimal sequence of C-FOVE operations (of which sum-out is just one). In models with sufficient symmetry, this maximum potential size may be exponentially smaller than that predicted by the treewidth of the underlying graphical model, since C-FOVE can represent large factors compactly using counting formulas.

Evaluation

We compare the performance of C-FOVE, FOVE[2], and a ground VE algorithm on two inference problems: *competing workshops* and *workshop attributes*. In the competing workshops problem, we are organizing a workshop and have invited a number of people. For each person P, there is a binary random variable $\mathsf{attends}(\mathsf{P})$ indicating whether that person attends our workshop. In addition, there are competing workshops W that each have a binary random variable $\mathsf{hot}(\mathsf{W})$ indicating whether they are focusing on popular research areas. There is also a random variable series, indicating whether our workshop is successful enough to start a series of related meetings. The distribution is defined by two parfactors. First, there is a parfactor $\forall \mathsf{P}, \mathsf{W}.\ \phi_1(\mathsf{attends}(\mathsf{P}), \mathsf{hot}(\mathsf{W}))$ that has small potential values and models the fact that people are not likely to attend our workshop if there are popular competing workshops. There is also a parfactor $\forall \mathsf{P}.\ \phi_2(\mathsf{attends}(\mathsf{P}), \mathsf{series})$ with larger values, indicating that we are more likely to start a series when more people attend our workshop.

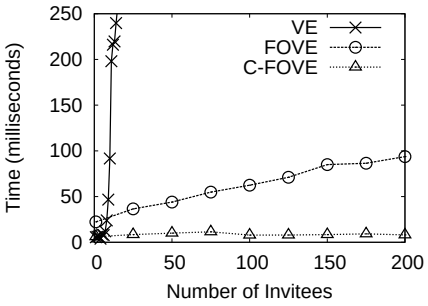

Figure 2: Performance on the competing workshops problem.

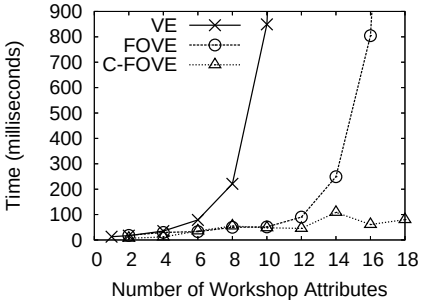

Figure 3: Performance on the workshop attributes problem.

Figure 2 shows a graph of running times for answering a query on series in the competing workshop problem. The marginals were computed for 10 workshops with an increasing number of people n. The cost of VE scales linearly in n (after n exceeds the number of workshops), but with a very large constant, because it must sum out each $\mathsf{attends}(\mathsf{P})$ variable separately and create a large potential on the $\mathsf{hot}(\mathsf{W})$ variables each time. FOVE applies counting elimination to both $\mathsf{attends}(\mathsf{P})$ and $\mathsf{hot}(\mathsf{W})$; its cost is much lower, but still increases with n. C-FOVE chooses to apply counting conversion on W to create a parfactor $\forall \mathsf{P}.\ \phi_1'(\mathsf{attends}(\mathsf{P}), \#_{\mathsf{W}}[\mathsf{hot}(\mathsf{W})])$; it then multiplies together the parfactors and sums out $\mathsf{attends}(\mathsf{P})$ at a lifted level. Finally, it sums out $\#_{\mathsf{W}}[\mathsf{hot}(\mathsf{W})]$. This cost of this sequence of operations is constant with respect to the number of people.

The second evaluation is in the workshop attributes domain. This scenario is similar to the previous one, but instead of competing workshops, we now include a set of m binary random variables $\mathsf{attr}_1 \ldots \mathsf{attr}_m$ that represent different attributes of our workshop (location, date,

[2]We use the FOVE code available at `http://l2r.cs.uiuc.edu/~cogcomp`

fame of the organizers, etc.). The distribution is defined by $m + 1$ distinct parfactors: $\forall \mathsf{P}.\ \phi_0(\mathsf{attends}(\mathsf{P}), \mathsf{series})$, $\forall \mathsf{P}.\ \phi_1(\mathsf{attends}(\mathsf{P}), \mathsf{attr}_1), \ldots, \forall \mathsf{P}.\ \phi_n(\mathsf{attends}(\mathsf{P}), \mathsf{attr}_m)$.

Figure 3 shows performance for answering a query on series with 50 people and an increasing number of attributes m. VE and FOVE both eliminate the attends(P) variables first. Although FOVE does this at a lifted level, both algorithms create potentials over all m workshop attributes, with a cost that is exponential in m. C-FOVE, on the other hand, chooses to apply counting conversion to attends(P) in each parfactor $\forall \mathsf{P}.\phi_i(\mathsf{attends}(\mathsf{P}), \mathsf{attr}_i)$, and then eliminate the associated attr_i variable. As a last step, it multiplies the resulting factors and sums out the $\#_{\mathsf{P}}[\mathsf{attends}(\mathsf{P})]$ formula. The time required is only linear in both the number of attributes and the number of invitees.

Related Work

The idea that lifted dependencies are often determined by the number of objects with specific properties is not new. The FOVE algorithm (de Salvo Braz, Amir, & Roth 2005; 2006) has a counting elimination operator that is equivalent to introducing a set of counting formulas and then summing them out immediately; it cannot keep counting formulas around while it eliminates other random variables, as C-FOVE does in the workshop attributes scenario. Gupta *et al.* (2007) present a message passing algorithm for MAP inference that allows a limited form of counting dependencies in the input language. The introduction and elimination of counting formulas during inference distinguishes C-FOVE from both of these approaches.

The parfactor representation was introduced by Poole (2003) and used in the FOVE algorithm (de Salvo Braz, Amir, & Roth 2007). A parfactor is similar to a clique template in a relational Markov network (Taskar, Abbeel, & Koller 2002); a clause in a Markov logic network (Richardson & Domingos 2006); or a relational potential in a logical conditional random field (Gutmann & Kersting 2006): these representations all define potentials that are repeated in an undirected graphical model.

Discussion and Future Work

In this paper, we presented the C-FOVE algorithm, which uses counting formulas to compactly represent interchangeability within large potentials during variable elimination. Our experiments illustrate cases where flexible elimination orderings and the introduction of counting formulas lead to asymptotic improvements in running time.

Counting formulas are a specific instance of a class of related functions. Many other aggregations (MAX, AVERAGE, SUM, etc.) can be computed as functions of count histograms. They can be represented compactly with counting formulas; we could represent them even more compactly by adding, say, SUM formulas, which have the same value for many different histograms. Exploring these possibilities is an important area for future work.

Additionally, we would like to incorporate counting formulas into approximate approaches that perform lifted belief propagation (Jaimovich, Meshi, & Friedman 2007) or incorporate lifting into sampling algorithms (Milch & Russell 2006; Zettlemoyer, Pasula, & Kaelbling 2007). Another way to scale up these algorithms is to exploit *approximate interchangeability*, where we have slightly different beliefs about the objects in some set, but we are willing to treat them as interchangeable to speed up inference.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted first-order probabilistic inference. In *Proc. 19th IJCAI*, 1319–1325.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2006. MPE and partial inversion in lifted probabilistic variable elimination. In *Proc. 21st AAAI*.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2007. Lifted first-order probabilistic inference. In Getoor, L., and Taskar, B., eds., *Introduction to Statistical Relational Learning*. MIT Press. 433–451.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1-2):41–85.

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

Gupta, R.; Diwan, A.; and Sarawagi, S. 2007. Efficient inference with cardinality-based clique potentials. In *Proc. 24th ICML*.

Gutmann, B., and Kersting, K. 2006. TildeCRF: Conditional random fields for logical sequences. In *Proc. 17th ECML*.

Jaimovich, A.; Meshi, O.; and Friedman, N. 2007. Template-based inference in symmetric relational Markov random fields. In *Proc. 23rd UAI*.

Milch, B., and Russell, S. 2006. General-purpose MCMC inference over relational structures. In *Proc. 22nd UAI*, 349–358.

Pfeffer, A.; Koller, D.; Milch, B.; and Takusagawa, K. 1999. SPOOK: A system for probabilistic object-oriented knowledge. In *Proc. 15th UAI*, 541–550.

Poole, D. 2003. First-order probabilistic inference. In *Proc. 18th IJCAI*, 985–991.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Taskar, B.; Abbeel, P.; and Koller, D. 2002. Discriminative probabilistic models for relational data. In *Proc. 18th UAI*.

Zettlemoyer, L. S.; Pasula, H. M.; and Kaelbling, L. P. 2007. Logical particle filtering. In *Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*.

Zhang, N. L., and Poole, D. 1994. A simple approach to Bayesian network computations. In *Proc. 10th Canadian Conf. on AI*.