Generalized Counting for Lifted Variable Elimination

Nima Taghipour **Jesse Davis**
Department of Computer Science, KU Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
{nima.taghipour,jesse.davis}@cs.kuleuven.be

Abstract

Lifted probabilistic inference methods exploit symmetries in the structure of probabilistic models to perform inference more efficiently. In lifted variable elimination, the symmetry among a group of interchangeable random variables is captured by *counting formulas*, and exploited by operations that handle such formulas. In this paper we generalize the structure of counting formulas and present a set of inference operators that introduce and eliminate these formulas from the model. This generalization expands the range of problems that can be solved in a lifted way. Our work is closely related to the recently introduced method of joint conversion. Due to its more fine grained formulation, however, our approach can provide more efficient solutions than joint conversion.

1 Introduction

Probabilistic logical languages combine elements of first-order logic with graphical models to succinctly model complex, uncertain, structured domains [3]. These domains often involve a large number of objects, making efficient inference a major challenge. To address this problem, lifted probabilistic inference was introduced [5]. Lifted inference methods exploit the symmetries in the structure of the model to perform inference more efficiently.

Lifted inference uses two main techniques or tools for *lifting*: (1) *divide* the problem into isomorphic and independent subproblems, *solve one* instance, and *aggregate* the result, and (2) *count* the number of isomorphic configurations for a group of *interchangeable* objects instead of enumerating all possible configurations. Our focus in this paper is on the second tool, counting, in the context of lifted variable elimination (LVE) [5, 2, 4].

LVE uses *counting formulas* to capture the the symmetry among a group of interchangeable objects [4]. A counting formula aggregates the joint state of a group of random variables into *histograms* that show only the *number* of variables with each state, without distinguishing between the individuals. For instance, the formula $\#_X[Attends(X)]$, captures the number of people who attend a workshop, without distinguishing between their identity. As the number of possible aggregate states (histograms) is much smaller than the number of joint states of the group, lifted operations achieve large efficiency gains by directly manipulating these formulas, instead of the individual variables.

Counting formulas, as used so far in LVE [4], have specific syntactic restrictions. Some of these restrictions are not fundamental, and can be removed to arrive at more general counting formulas, which yields more opportunities for lifting. One such restriction is that a counting formula contains only a single atom, i.e., it aggregates the state of a *group of individual* random variables. In this paper, first we generalize the definition of counting formulas in a straightforward way, to allow a counting formula to aggregate the state of a *group of tuples* of random variables. For instance, we allow a counting formula such as $\#_X[Attends(X), Presents(X)]$ that counts the number of people that (do not) attend *and* (do not) present a paper at the workshop. Second, we present a set of inference operators that introduce and manipulate these generalized formulas, and show that these expand the opportunities for lifting, and hence for more efficient inference, compared to the original formulation of counting operations.

Our work is closely related to Apsel and Brafman's [1] on *joint conversion* and *just-different* counting conversion. However, our method uses a more fine grained formulation, and can offer more efficient solutions than joint conversion.

2 Representation

Probabilistic logical models combine graphical models with elements of first-order logic. Many representation languages exist for such models [3]. Like earlier work on LVE [5, 2, 4], we represent the model with *parametric factors*. This formalism can compactly represent undirected probabilistic models on large numbers of objects. We now introduce the necessary terminology.

The term "variable" can be used in both the logical and probabilistic context. To avoid confusion, we use the term *logvar* to refer to logical variables, and *randvar* to refer to random variables. We write variable names in uppercase, and their values in lowercase.

We consider factorized probabilistic models. A *factor* $f = \phi_f(\mathcal{A}_f)$, where $\mathcal{A}_f = (A_1, \ldots, A_n)$ are randvars and ϕ_f is a *potential* function, maps configurations of $\mathcal{A}_f$ to a real-number. An *undirected model* is a set of factors F over randvars $\mathcal{A} = \bigcup_{f \in F} \mathcal{A}_f$ and represents the following probability distribution: $\mathcal{P}_F(\mathcal{A}) = \frac{1}{Z} \prod_{f \in F} \phi_f(\mathcal{A}_f)$, with Z a normalization constant.

Our representation compactly defines a set of factors, using concepts from first-order logic. A *constant* represents an object in our universe. A *term* is either a constant or a logvar. A *predicate* P has an arity n and a finite range ($range(P)$); it maps n-tuples of objects (constants) to the range. An *atom* is of the form $P(t_1, t_2, \ldots, t_n)$, where the t_i are terms. A ground atom is an atom $P(c_1, \ldots, c_n)$ where the c_i are constants. The range of such a ground atom is $range(P)$. Each ground atom represents a randvar (e.g., $BloodType(joe)$). Note that the range of predicates, and hence randvars, is not limited to $\{true, false\}$ as in logic (e.g., $range(BloodType(joe)) = \{a, b, ab, o\}$).

Each Logvar X has a finite domain $D(X)$, which is a set of constants. Unless mentioned otherwise, without loss of generality, we assume that $D(X) = \{x_1, \ldots, x_n\}$ for the logvars. A *substitution*, $\theta = \{X_1 \to t_1, \ldots, X_n \to t_n\}$, is a mapping of logvars to terms. A *grounding substitution* maps all logvars to constants. Applying θ to a, denoted $a\theta$, replaces each occurrence of X_i in a with t_i.

A *constraint* $C_{\mathbf{X}}$ on a set of logvars $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a conjunction of inequalities of the form $X_i \neq t$ where t is a constant in $D(X_i)$ or a logvar in $\mathbf{X}$ (the conditions $X_i \in D(X_i)$ are left implicit). We write C instead of $C_{\mathbf{X}}$ when $\mathbf{X}$ is apparent from the context. By $gr(\mathbf{X}|C_{\mathbf{X}})$ we denote the set of ground substitutions to $\mathbf{X}$ that are consistent with $C_{\mathbf{X}}$.

A *parametrized randvar (PRV)* is a constrained atom of the form $P(\mathbf{X})|C$, where $P(\mathbf{X})$ is an atom and C is a constraint on $\mathbf{X}$. A PRV $P(\mathbf{X})|C$ represents a set of ground atoms (and hence, a set of randvars) $\{P(\mathbf{X})\theta | \theta \in gr(\mathbf{X}|C)\}$. Given a PRV $\mathcal{V}$, we use $RV(\mathcal{V})$ to denote the set of randvars it represents; we also say these randvars are *covered* by $\mathcal{V}$.

Example. The PRV $\mathcal{V} = Smokes(X)|X \neq x_1$, with $D(X) = \{x_1, \ldots, x_n\}$, represents $n-1$ randvars $\{Smokes(x_2), \ldots Smokes(x_n)\}$. □

A *valuation* of a randvar (set of randvars) is an assignment of a value to the randvar (resp. an assignment of values to all randvars in the set).

A *parametric factor* or *parfactor* is of the form $\forall \mathbf{L} : C.\phi(\mathcal{A})$, with $\mathbf{L}$ a set of logvars, C a constraint on $\mathbf{L}$, $\mathcal{A} = (A_i)_{i=1}^n$ a sequence of atoms parametrized with $\mathbf{L}$, and ϕ a potential function on $\mathcal{A}$. The set of logvars occurring in $\mathcal{A}$ is denoted $logvar(\mathcal{A})$, and we have $logvar(\mathcal{A}) \subseteq \mathbf{L}$. When $logvar(\mathcal{A}) = \mathbf{L}$, we write the parfactor as $\phi(\mathcal{A})|C$. A factor $\phi(\mathcal{A}')$ is called a *grounding* of a parfactor $\phi(\mathcal{A})|C$ if $\mathcal{A}'$ can be obtained by instantiating $\mathbf{L}$ according to a grounding substitution $\theta \in gr(\mathbf{L}|C)$. The set of all groundings of a parfactor g is denoted $gr(g)$.

Example. Parfactor $g = \phi_1(Smokes(X))$ represents the set of n ground factors $gr(g) = \{\phi_1(Smokes(x_1)), \ldots, \phi_1(Smokes(x_n))\}$.

When talking about a *model* below, we mean a set of parfactors. In essence, a set of parfactors G is a compact way of defining a set of factors $F = \{f | f \in gr(g) \wedge g \in G\}$. The corresponding probability distribution is $\mathcal{P}_G(\mathcal{A}) = \frac{1}{Z} \prod_{f \in F} \phi_f(\mathcal{A}_f)$.

3 Lifted Variable Elimination

The state of art in lifted variable elimination (LVE) is the result of various complementary efforts [5, 4, 1, 2]. In this section we briefly review the C-FOVE [4] algorithm, which forms the basis of our work. Another extension to C-FOVE, namely *joint formulas* [1], is discussed and compared with in Section 7.

Variable elimination calculates the marginal distribution of some variable by *eliminating* randvars in a specific order from the model until reaching the desired marginal [6]. To eliminate a single randvar V, it first *multiplies* all the factors containing V into a single factor and then *sums out* V from that single factor. LVE does this on a lifted level by eliminating parametrized randvars (i.e., whole groups of randvars) from parfactors (i.e., group of factors). LVE performs inference using a set of *lifted* operators that we summarize below. Table 1 shows the outer loop of LVE.

Lifted sum-out sums-out a PRV, and hence all the

Inputs: G: a model; Q: the query randvar.
while G contains other randvars than Q:
  if a PRV $\mathcal{V}$ can be eliminated by lifted sum-out
    $G \leftarrow$ eliminate $\mathcal{V}$ in G by lifted sum-out
  else apply an enabling operator on parfactors in G
end while
return G

Table 1: Outer loop of LVE.

randvars represented by that PRV, from the model. It is applicable when each randvar represented by the PRV appears in exactly one grounding of exactly one parfactor in the model. This is possible when the atom representing that PRV contains *all* the logvars in that parfactor. The goal of all other operators is to manipulate the parfactors into a form that satisfies this precondition. In this sense, all operators except lifted sum-out can be seen as *enabling operators*.

Lifted multiplication performs the equivalent of many factor multiplications in a single lifted operation. It prepares the model for sum-out by replacing all the parfactors that share a PRV by a single equivalent product parfactor in the model.

Shattering and *splitting* rewrite the model such that the first group of operations can be applied on it.

Additionally, LVE has another important enabling operator, *counting conversion*, which introduces counting formulas into the model. We introduce and generalize counting formulas, along with operations that handle them, in the following sections.

4 Generalized Counting Formulas

Counting is one of the fundamental techniques used for lifting. It is applied to lift the computations performed on a group of interchangeable randvars in the *same factor*. This is achieved by manipulation of *counting formulas*. Such a formula aggregates the state of a group of interchangeable randvars into histograms that show the number of randvars in the group with each value. These structures take the computations to the level of the aggregate state of the group, without considering the state of individual randvars. This allows for more efficient computations as the number of possible aggregate states (histograms) is polynomial in the domain size, whereas the number of joint states is exponential in the domain size.

In this section, we generalize counting formulas, such that they aggregate the state of a group of *tuples* of atoms, instead of a group of atoms. This permits lifting in cases where not all individual randvars are exchangeable in a group, but specific tuples of randvars are. Our definition of a counting formula is a straightforward generalization of their existing formulation [4].

Counting Formulas. We define a counting formula to be of the form $\gamma = \#_{X:C}[P_1(\mathbf{X}_1), \ldots, P_k(\mathbf{X}_k)]$, with C a constraint on the *counted logvar* X, and $X \in \mathbf{X}_i (i = 1, \ldots, k)$. The counted logvar X is bound by the counting formula and excluded from $logvar(\gamma)$. A grounded counting formula is a counting formula in which all logvars except the counted logvar are replaced by constants. Such a formula represents a *counting randvar* whose range is the set of possible histograms that distribute n elements into $r = \prod_{i=1}^{k} |range(P_i)|$ buckets. More precisely its state is the histogram function $h = \{(r_i, n_i)\}_{i=1}^{r}$, that shows for each $r_i \in \times_{i=1}^{k} range(P_i)$ the number n_i of tuples $(P_1(\ldots, x, \ldots), \ldots, P_k(\ldots, x, \ldots))$ whose state is r_i. The state of the counting randvar thus depends deterministically on the state of the randvars $\cup_{i=1}^{k} RV(P_k(\ldots, X, \ldots)|C)$, or more precisely on the state of *tuples* of randvars $RV(P_1(\ldots, X, \ldots), \ldots, P_k(\ldots, X, \ldots)|C)$. For simplicity, we denote a histogram $h = \{(r_i, n_i)\}_{i=1}^{r}$ by the list of counts $(n_1, \ldots, n_r)$, when the elements r_i are apparent from the context.

Example. Having $D(X) = \{ann, bob, carl, dave\}$, the counting randvar $\gamma = \#_X[Smokes(X), Asthma(X)]$ aggregates the state of tuples of randvars $RV(\gamma) = \{(Smokes(x_i), Asthma(x_i))|x_i \in D(X)\}$. Suppose that $RV(\gamma)$ are assigned the following set of values:

X	$Smokes(X)$	$Asthma(X)$
ann	f	t
bob	f	f
$carl$	f	f
$dave$	t	t

Then the value of the counting randvar is the histogram $h = \{(tt, 1), (tf, 0), (ft, 1), (ff, 2)\}$. □

Our definition of a counting formula, which allows a tuple of atoms in a counting formula, is a simple generalization of Milch et al.'s definition, which only allows a single atom (a 1-tuple). We show how these formulas provide us with data structures that create more opportunities for lifting through a suite of model conversion operations, in Section 5, and present a lifted sum-out operation for these formulas in Section 6. But first, we recall a *normal form* for the model, which guarantees correct semantics, and facilitates handling of counting formulas [4].

Normal parfactors. As mentioned, the range of a counting formula depends on the number of randvars that it counts, which in turn depends on its associated constraint. For instance consider the formula $\#_{Y:\{Y \neq X, Y \neq x_1\}}[F(X, Y)]$. For $X = x_1$, it represents

a CRV that counts $n-1$ randvars (Y can take on all n values except x_1), but for $X = x' \neq x_1$ it represents a CRV that counts $n-2$ randvars (Y can take on all n values except x_1 and x'). To ensure that all CRVs in the groundings of a parfactor have the same range, following Milch et al., we require the constraints and parfactors to be in a *normal form*. A constraint C is in normal form if for each pair of logvars X_1 and X_2 with an inequality constraint $X_1 \neq X_2$:

$$E_{X_1} \setminus \{X_2\} = E_{X_2} \setminus \{X_1\}$$

where the *excluded* set $E_X = \{t \mid (X \neq t) \in C\}$ is the set of terms in an inequality constraint with X in constraint C. This guarantees that a logvar X_i has the same number of values in C, given any value to the rest of the logvars. This number is denoted $|X : C|$ and equals $|D(X)| - |E_X|$. A parfactor is in normal form if the conjunct of its constraint with the constraints of its counting formulas is in normal form. Any model can rewritten into an equivalent one in normal form using the auxiliary operations [4].

5 Conversion Operations

In this section we present conversion operations that rewrite the model in terms of counting formulas. The first operation is a generalization of C-FOVE's counting conversion [4], while the rest are new operations. Throughout this paper, we illustrate the application of the new operators on examples for which C-FOVE [4] has no lifted solution. As such, we show how the new operations perform inference with complexity polynomial in the domain size, where C-FOVE cannot avoid the exponential complexity of propositional inference.

5.1 Counting Conversion

Counting formulas are introduced into the model by *counting conversion*. By rewriting the model (replacing an *atom*) with a counting formula, this operation allows us to compactly represent and manipulate a high dimensional factor on a set of interchangeable randvars. Intuitively this conversion achieves the equivalent of multiplying groundings of a single parfactor with each other, and thus functions as an enabling operation for lifted sum-out. We generalize this operation to a rewrite rule that replaces a *tuple* of atoms with a counting formula. By removing the restriction on the number of counted atoms, this generalization provides more opportunities for lifting.

Example. Consider the model consisting of the parfactor $\phi(A(X), B(X), C(Y), D(Y))$. Lifted sum-out is not applicable here, as no atom contains all the logvars. Counting conversion removes a logvar from the set of free logvars, hence preparing the model for lifted sum-out. Here we perform counting conversion on logvar X by introducing a counting formula on the tuple of atoms $A(X), B(X)$, and rewrite the parfactor as $\phi'(\#_X[A(X), B(X)], C(Y), D(Y))$, where ϕ' is such that for each histogram $h(.)$ in the range of $\#_X[A(X), B(X)]$, $\phi'(h(.), c, d)$ is equal to

$$\phi(t,t,c,d)^{h(tt)} \cdot \phi(t,f,c,d)^{h(tf)} \cdot \phi(f,t,c,d)^{h(ft)} \cdot \phi(f,f,c,d)^{h(ff)}$$

Note that after this conversion, logvar Y is the only free logvar in the parfactor, which enables lifted sum-out of both $C(Y)$ and $D(Y)$ from the model.[1] □

We can now formally define this operator.

Operation: COUNT-CONVERT
Input: (1) a parfactor $g = \forall \mathbf{L} : C.\phi(\mathcal{A})$ (2) a logvar $X \in logvar(\mathcal{A})$
Preconditions: (1) there is no counting formula in the set $\mathcal{A}_X = \{A \in \mathcal{A} | X \in logvar(A)\}$ (2) There is no counting formula $\gamma = \#_{X_i:C_i}[\ldots]$ in $\mathcal{A}$, such that $(X_i \neq X) \in C_i$
Output: $g' = \forall \mathbf{L}' : C'.\phi'(\mathcal{A}')$, such that:
(1) $\mathbf{L}' = \mathbf{L} \setminus \{X\}$ (2) C' is the projection of C on $\mathbf{L}'$ (3) $\mathcal{A}' = \mathcal{A} \setminus \mathcal{A}_X \cup \{\#_X[\mathcal{A}_X]\}$, and (4) for each valuation $(h(.), \mathbf{a})$ to $(\#_X[\mathcal{A}_X], \mathcal{A} \setminus \mathcal{A}_X)$:

$$\phi'(h(.), \mathbf{a}) = \prod_{\mathbf{a}' \in range(\mathcal{A}_X)} \phi(\mathbf{a}'; \mathbf{a})^{h(\mathbf{a}')}$$

The preconditions for counting conversion of a logvar X require that it does not appear inside an existing counting formula. This means that X cannot appear (1) in an atom inside a counting formula, or (2) in the constraint associated with a counting formula. Excluding the first case ensures that the result of conversion can be represented by our counting formulas, and does not require more complicated structures like nested or overlapping counting formulas, which are not well defined in our formulation. In the second case, where X is in an inequality constraint with a counted logvar, counting conversion is still possible, but by the operation of *merge-counting*, which will be introduced in Section 5.3. Note that the precondition for counting conversion as defined above, is weaker than the original one of C-FOVE, which requires X to appear in exactly one atom in the parfactor [4].

5.2 Merging Counting Formulas

Two counting formulas can count over tuples of randvars with overlapping randvars between them. When

[1] We further show in Section 6 that $A(X)$ and $B(X)$ can also be eliminated with a lifted sum-out.

such formulas appear in a parfactor together, to sum-out the common randvars, we need to first merge these counting formulas into one.

Example. Consider the counting formulas in the parfactor $\phi(\#_X[S(X)], \#_Y[S(Y), A(Y)])$. The first argument, $\gamma_1 = \#_X[S(X)]$, aggregates the state of randvars $RV(S(X))$, and the second argument, $\gamma_2 = \#_Y[S(Y), A(Y)]$, the state of $RV(S(X), A(X))$. As the second group of randvars is a superset of the first set, given a histogram $h_2(.)$ for γ_2, we can infer the value h_1 of γ_1. We can therefore merge the two counting formulas into one $\#_X[S(X), A(X)]$ and rewrite the parfactor as $\phi'(\#_X[S(X), A(X)])$ where $\phi'(h_2) = \phi(h_1, h_2)$ and h_1 is the histogram that results from projecting h_2 on the assignments to the $S()$ randvars. Concretely, having the counts $(n_{tt}, n_{tf}, n_{ft}, n_{ff})$ for h_2, the value of h_1 is the histogram with counts $(n_{tt} + n_{tf}, n_{ft} + n_{ff})$. □

Although in the above example one CRV was a superset of the other, merging can be applied to any pair of formulas with overlapping randvars. For instance, by merging, the parfactor $\phi(\#_X[A(X), B(X)], \#_Y[B(Y), C(Y))])$ is transformed into a parfactor $\phi(\#_X[A(X), B(X), C(X)])$. To formalize this operator we introduce the notion of *compatible* valuations to atoms, and then the *projection* of histograms.

A pair of valuations $(\mathbf{a}_1, \mathbf{a}_2)$ to $(\mathcal{A}_1, \mathcal{A}_2)$ are *compatible*, denoted by $\mathbf{a}_1 \sim \mathbf{a}_2$, if each atom $A_i \in \mathcal{A}_1 \cap \mathcal{A}_2$ is assigned with the same value a_i in both $\mathbf{a}_1$ and $\mathbf{a}_2$.

Given a counting formula $\gamma = \#_X[\mathcal{A}]$, the *projection* of a histogram $h \in range(\gamma)$ on $\mathcal{A}' \subseteq \mathcal{A}$, is a histogram $h' \in range(\#_X[\mathcal{A}'])$ such that for each $\mathbf{a}' \in range(\mathcal{A}')$: $h'(\mathbf{a}') = \sum_{\mathbf{a} \sim \mathbf{a}'} h(\mathbf{a})$. We denote the projection of h on $\mathcal{A}'$ by $h_{[\mathcal{A}']}$.

We can now formally define this operator.

Operation: MERGE
Input: (1) a parfactor $g = \forall \mathbf{L} : C.\phi(\mathcal{A})$ (2) a pair of counting formulas $(\gamma_1, \gamma_2) = (\#_{X_1:C_1}[\mathcal{A}_1], \#_{X_2:C_2}[\mathcal{A}_2])$ in $\mathcal{A}$
Precondition: $gr(X_1 : C \wedge C_1) = gr(X_2 : C \wedge C_1)$
Output: $g' = \forall \mathbf{L} : C.\phi'(\mathcal{A}')$, such that:
(1) $\mathcal{A}' = \mathcal{A} \setminus \{\gamma_1, \gamma_2\} \cup \{\#_{X_1}[\mathcal{A}_{12}]\}$, with $\mathcal{A}_{12} = \mathcal{A}_1 \cup \mathcal{A}_2\theta$ and $\theta = \{X_2 \to X_1\}$ (2) for each valuation $(h(.), \mathbf{a})$ to $(\#_{X_1}[\mathcal{A}_{12}], \mathcal{A} \setminus \{\gamma_1, \gamma_2\})$:

$$\phi'(h, \mathbf{a}) = \phi(h_{[\mathcal{A}_1]}, h_{[\mathcal{A}_2\theta]}; \mathbf{a})$$

5.3 Merge-counting

This operation is applicable when counting conversion cannot replace a tuple of atoms by a new counting formula, but needs to merge them into an existing counting formula. Concretely, this happens during counting conversion on a logvar X, in a parfactor with a counting formula $\#_{Y:Y \neq X}[\mathcal{A}_Y]$, that is, when an existing counted logvar is in an inequality constraint with X. In such cases, we cannot convert the parfactor to an equivalent one with two separate counting formulas $\#_Y[\mathcal{A}_Y]$ and $\#_X[\mathcal{A}_X]$. Intuitively, this is because the histograms of these two counting formulas do not determine the value of the original parfactor. Instead, we can apply *merge-counting*, which incorporates the atoms $\mathcal{A}_X$ inside the existing counting formula. Consider the following example:

Example. Consider the parfactor g of the form $\phi(\#_{X:X \neq Y}[A(X)], B(Y))$. We show how merge-counting on logvar Y rewrites g as an equivalent parfactor $g' = \phi'(\#_X[A(X), B(X)])$. For this we need to properly define the potential ϕ' based on ϕ. Note that g' represents a single factor, while g represents n factors, one for each $y \in D(Y)$. The potential ϕ' should thus be defined such that g' evaluates the product of these n factors, for any valuation of randvars $A(X)$ and $B(X)$. Consider a valuation that yields histogram $h(.)$ for $\#_X[A(X), B(X)]$. To compute $\phi'(h)$, we compute the value of g at this valuation, based on the following observations: in $gr(g)$, each factor $g_y = \phi(\gamma_y, B(y))$, has a *distinct* counting formula $\gamma_y = \#_{X:X \neq y}[A(X)]$, which covers all the randvars $RV(A(X))$ *except* $A(y)$, due to the inequality constraint. Since the histogram of $\#_X[A(X)]$ is $h_{[A]} = (n_t, n_f)$, each CRV γ_y, which excludes the value of one randvar $A(y)$, takes on one of the two histograms

- $h^{-t}_{[A]} = (n_t - 1, n_f)$, when $A(y) = t$, or
- $h^{-f}_{[A]} = (n_t, n_f - 1)$, when $A(y) = f$

Each factor g_y in $gr(g)$ thus evaluates to one of the four values $\phi(h^-, b)$, for $(h^-, b) \in \{h^{-t}_{[A]}, h^{-f}_{[A]}\} \times \{t, f\}$. Knowing the number $\#(h^-, b)$ of factors with each value $\phi(h^-, b)$, we can compute the desired potential ϕ' as:

$$\phi'(h(.)) = \prod_{(h^-, b)} \phi(h^-, b)^{\#(h^-, b)}$$

The numbers $\#(h^-, b)$ are inferred directly from the histogram $h(.)$. For instance, $\#(h^{-t}_{[A]}, t)$, the number of ys with $\gamma_y = h^{-t}_{[A]}$ and $B(y) = t$, by definition equals $h(tt)$, that is, the number of ys with $A(y) = B(y) = t$. With similar reasoning, we determine all the numbers $\#(h^-, b)$ from $h(.)$ and define the desired potential ϕ' as:

$$\phi'(h(.)) = \phi(h^{-t}_{[A]}, t)^{h(tt)} \cdot \phi(h^{-f}_{[A]}, t)^{h(ft)} \cdot \phi(h^{-t}_{[A]}, f)^{h(tf)} \cdot \phi(h^{-f}_{[A]}, f)^{h(ff)}$$

As such, merge-counting replaces the parfactor g with the equivalent parfactor $g' = \phi'(\#_X[A(X), B(X)])$, by directly computing ϕ' from ϕ. □

The operator is formally defined as follows.

Operation: MERGE-COUNT
Input: (1) a parfactor $g = \forall \mathbf{L} : C.\phi(\mathcal{A})$ (2) a counting formula $\gamma = \#_{X_1:C_1}[\mathcal{A}_1]$ in $\mathcal{A}$ (3) a logvar X_2 in $logvar(\mathcal{A})$
Precondition: (1) there is no counting formula in $\mathcal{A}_2 = \{A \in \mathcal{A} | X_2 \in logvar(A)\}$ (2) γ is the only counting formula whose counted logvar X_1 is in an inequality constraint with X_2
Output: $g' = \forall \mathbf{L}' : C'.\phi'(\mathcal{A}')$, such that:
(1) $\mathbf{L}' = \mathbf{L} \setminus \{X_2\}$ (2) C' is the projection of C on $\mathbf{L}'$ (3) $\mathcal{A}' = \mathcal{A} \setminus \{\gamma, \mathcal{A}_2\} \cup \{\#_{X_1:C_{12}}[\mathcal{A}_{12}]\}$, with $\mathcal{A}_{12} = \mathcal{A}_1 \cup \mathcal{A}_2\{X_2 \to X_1\}$, and $C_{12} = C_1 \setminus (X_1 \neq X_2)$ (4) for each valuation $(h(.), \mathbf{a})$ to $(\#_{X_1}[\mathcal{A}_{12}], \mathcal{A}' \setminus \#_{X_1}[\mathcal{A}_{12}])$:

$$\phi'(h(.), \mathbf{a}) = \prod_{(\mathbf{a}_{12}) \in range(\mathcal{A}_{12})} \phi(h^{-\mathbf{a}_1}_{[\mathcal{A}_1]}, \mathbf{a}_2; \mathbf{a})^{h(\mathbf{a}_{12})}$$

where $\mathbf{a}_i$ denotes the projection of the valuation $\mathbf{a}_{12}$ on $\mathcal{A}_i\{X_2 \to X_1\}$, and the histogram $h^{-\mathbf{r}}$ is such that $h^{-\mathbf{r}}(\mathbf{r}) = h(\mathbf{r}) - 1$, and $h^{-\mathbf{r}}(\mathbf{r}') = h(\mathbf{r}')$, for $\mathbf{r}' \neq \mathbf{r}$.

Note that the second precondition can be established by *merging* counting formulas. Merging is thus an enabling operator for this conversion operator.

6 Elimination Operations

In the previous section, we presented the operators that introduce or merge counting formulas. In this section, we present a lifted *sum-out* operator that eliminates these formulas, as well as an operator that aggregates the results after lifted sum-out.

6.1 Sum-out by Counting

We present an operation for summing out an atom inside counting formulas. This lifted operation sums out all the randvars represented by the atom from the model. It functions as a rewrite rule that removes the atom from a counting formula, and has the sum-out operation of C-FOVE as a special case.

Example. Consider summing-out the PRV $A(X)$ from the model defined by the parfactor $\phi(\#_X[A(X), B(X)])$. By lifted sum-out we derive the parfactor $\phi'(\#_X[B(X)])$, for which we define the potential function ϕ' such that for each histogram $h' \in range(\#_X[B(X)])$:

$$\phi'(h') = \sum_{h \sim h'} Num(h|h')\phi(h)$$

where $h \sim h'$ denotes a histogram $h \in range(\#_X[A(X), B(X)])$ that is compatible with histogram h', that is, $h' = h_{[B]}$. The quantity $Num(h|h')$ equals the number of possible ways that a valuation to $RV(A(X))$ with the counts h', can be extended to a valuation to $RV(A(X), B(X))$ with the counts h. □

The coefficient $Num(h|h')$ is defined based on the number $Num(h)$ of possible valuations to the randvars that result in a histogram h: For each histogram $h = \{(r_i, n_i)\}_{i=1}^r$, with $\sum_i n_i = n$, we define $Num(h) = \frac{n!}{(n_1!)\ldots(n_r!)}$, and $Num(h|h') = \frac{Num(h)}{Num(h')}$.

Note that sum-out removes an atom from the counting formula. Summing-out the last atom, such as $B(X)$ in the above example, results in an empty counting formula, for which we define the range as $\{(Null, 0)\}$, $Num((Null, 0)) = 1$, and which we trivially remove from the list of arguments after sum-out. The operation is formally defined below.

Operation: SUM-OUT
Input: (1) a parfactor $g = \forall \mathbf{L} : C.\phi(\mathcal{A})$ in model G (2) a counting formula $\gamma = \#_{X_1:C_1}[\mathcal{A}_1]$ in $\mathcal{A}$ (3) an atom $A \in \mathcal{A}_1$
Precondition: (1) $logvar(A) = \mathbf{L}$ (2) for all PRVs $(A'|C')$ in the model, other than $(A|C \wedge C_1)$: $RV(A|C \wedge C_1) \cap RV(A'|C') = \emptyset$
Output: $g' = \forall \mathbf{L} : C.\phi'(\mathcal{A}')$, such that:
(1) $\mathcal{A}' = \mathcal{A} \setminus \{\gamma\} \cup \{\gamma'\}$, with $\gamma' = \#_{X_1:C_1}[\mathcal{A}_1 \setminus \{A\}]$ (2) for each valuation $(h'(.), \mathbf{a})$ to $(\gamma', \mathcal{A}' \setminus \gamma')$:

$$\phi'(h'(.), \mathbf{a}) = \sum_{h \sim h'} Num(h|h') \cdot \phi(h(.), \mathbf{a})$$

This operation has the sum-out operation of C-FOVE as a special case, namely when $\mathcal{A}_1 = \{A\}$. It can also be further generalized to sum-out a group of atoms $\{A_1, \ldots, A_n\} \subseteq \mathcal{A}$ in one operation, if the two preconditions are satisfied for all atoms.

6.2 Aggregation

In lifted inference, after dividing a problem into isomorphic subproblems, first the result of one prototypical instance of this problem is computed and then the result is *aggregated*, usually with the trivial operation of exponentiation. This operator in fact multiplies a group of identical factors and is applied when a logvar disappears from the parfactor after a sum-out operation. Aggregation can, however, be extended to cases where a simple exponentiation does not work. In this section we extend this operator and show how this allows for more efficient lifted computations.

Example. Consider the parfactor g of the form $\forall Y.\phi(\#_{X:X \neq Y}[P(X)], Q(Y))$. Summing-out $Q(Y)$ results in the parfactor $g' = \forall Y.\phi'(\#_{X:X \neq Y}[P(X)])$, on

which sum-out is no longer applicable. Note that the logvar Y is still part of the parfactor, although it does not appear in any atom. We show how, by *aggregation*, we can rewrite g' as an equivalent parfactor $g'' = \phi''(\#_X[P(X)])$, which is free of logvar Y. Assume $D(X) = D(Y) = \{ann, bob, carl, dave\}$. Then g' represents four ground factors, one for each person in $D(Y)$, e.g., for $Y = ann$ there is a factor $\phi'(\#_{X:X \neq ann}[P(X)])$ in $gr(g')$. g'' should be defined such that $\phi''(\#_X[P(X)])$ equals the product of these four factors. Note that these factors all have the same potential, but each on a CRV that excludes one distinct randvar from the group $R_P = \{P(ann), P(bob), P(carl), P(dave)\}$. Given any assignment to R_P with n_t true and n_f false, each of these CRVs has either one less $true$ than n_t or one less $false$ than n_f. Specifically, there are n_t histograms with counts $(n_t - 1, n_f)$ and n_f histograms with counts $(n_t, n_f - 1)$. Since the value $\phi'(h)$ is the same for all factors with the same histogram h, to compute the product it suffices to know (n_t, n_f). Aggregation rewrites g' as $\phi''(\#_X[P(X)])$, where the potential ϕ'' is such that:

$$\phi''((n_t, n_f)) = \phi'((n_t - 1, n_f))^{n_t} \cdot \phi'((n_t, n_f - 1))^{n_f}$$

The logvar Y is now removed from g', and we can eliminate $P(X)$ from the model by lifted sum-out. □

The more expensive alternative to aggregation is to apply counting on both atoms, and work with the parfactor $\phi^*(\#_X[P(X), Q(X)])$. This alternative solution eliminates both P and Q atoms with counting sum-out, in poly time, while the above solution uses counting only for the P atoms, and runs in linear time.

We formalize this operator as follows.

Operation: AGGREGATE
Input: (1) a parfactor $g = \forall \mathbf{L} : C.\phi(\mathcal{A})$ in model G (2) a counting formula $\gamma = \#_{X_1:C_1}[\mathcal{A}_1]$ in $\mathcal{A}$ (3) a logvar $X_2 \in \mathbf{L} \setminus logvar(\mathcal{A})$
Precondition: $\mathcal{A}$ has no counting formula $\#_{X_i:C_i}[.]$ other than γ, such that $(X_i \neq X_2) \in C_i$
Output: $g' = \forall \mathbf{L}' : C'.\phi'(\mathcal{A}')$, such that:
(1) $\mathbf{L}' = \mathbf{L} \setminus \{X_2\}$ (2) C' is the projection of C on L' (3) $\mathcal{A}' = \mathcal{A} \setminus \{\gamma\} \cup \{\#_{X_1:C_1'}[\mathcal{A}_1]\}$, with $C_1' = C_1 \setminus \{X_1 \neq X_2\}$ (4) for each valuation $(h(.), \mathbf{a})$ to $(\#_{X_1:C_1'}[\mathcal{A}_1], \mathcal{A}' \setminus \{\#_{X_1:C_1'}[\mathcal{A}_1]\})$:

$$\phi'(h(.), \mathbf{a}) = \prod_{\mathbf{a}_1 \in range(\mathcal{A}_1)} \phi(h^{-\mathbf{a}_1}(.), \mathbf{a})^{h(\mathbf{a}_1)}$$

7 Relation to Joint Conversion

Our contributions are closely related to, and target similar problems as, *joint conversion* and just-different counting conversion [1]. Our approach, however, can provide more efficient solutions than those based on the mentioned methods.

Joint conversion enables counting the states of a group of tuples of randvars, without modifying Milch et al.'s definition of counting formulas [4]. For instance, to enable counting *tuples* of randvars $(A(x), B(x))$, joint conversion replaces each occurrence of atoms $A(X)$ and $B(X)$ in the model with a joint atom $J_{AB}(X)$, whose state is the Cartesian product of the two atoms. Counting conversion can then derive a counting formula like $\#_X[J_{AB}(X)]$, which corresponds to a formula $\#_X[A(X), B(X)]$ in our formulation. When combined with just-different counting [1], joint conversion may also enable counting on logvars that are constrained to be unequal, similar to our approach.

However, there are differences between the two methods. Joint conversion is a global operation on the model, which introduces more dependencies by coupling two randvars into a joint randvar. After this operation, inference deals solely with the joint atom, and never directly with its constituents. Our method, however, uses a more fine grained formulation, by which it not only can simulate joint conversion, but also provide more efficient solutions than those possible by joint conversion. This primarily happens when the operations can divide the problem into independent parts, by eliminating a subset of the atoms that joint conversion couples in a joint atom. This allows for more efficient computations by avoiding the dependencies induced by unnecessary joint conversions. We illustrate this advantage in the following example.

Example. Consider summing-out all the randvars, i.e., computing the partition function, in the model consisting of the following parfactors

$$\begin{aligned} g_1 &= \phi_1(P_1(X), P_2(Y)) | X \neq Y \\ g_2 &= \phi_2(P_2(Y), P_3(X)) | X \neq Y \\ &\dots \\ g_m &= \phi_m(P_m(X), P_{m+1}(Y)) | X \neq Y \end{aligned}$$

Our approach. To sum out all the randvars, i.e., to eliminate all the atoms, our approach can proceed as follows. It first performs a counting conversion on Y in g_1 to derive $g_1' = \phi'(P_1(X), \#_{Y:Y \neq X}[P_2(Y)])$. Next, it eliminates $P_1(X)$ from the model by lifted sum-out, and aggregation. This results in $g_1^* = \phi_1^*(\#_Y[P_2(Y)])$. To prepare the model for summing-out $P_2(Y)$, we first perform the following operations:

1. COUNT-CONVERT on logvar Y in g_2 to derive the parfactor $g_2' = \phi_2'(\#_{Y:Y \neq X}[P_2(Y)], P_3(X))$
2. MERGE-COUNT on logvar X in g_2' to derive the

parfactor $g_2'' = \phi_2''(\#_Y[P_2(Y), P_3(Y)])$

3. MULTIPLY g_1^* and g_2'' to derive the parfactor $g_{12} = \phi_{12}(\#_Y[P_2(Y), P_3(Y)]), \#_Y[P_2(Y)])$
4. MERGE the counting formulas to derive a $g_{12}' = \phi_{12}'(\#_Y[P_2(Y), P_3(Y)])$

Now from this parfactor, we sum-out $P_2(Y)$ and get a parfactor $g_2^* = \phi_{12}^*(\#_Y[P_3(Y)])$. Inference continues by eliminating P_3 from parfactors g_2^* and g_3, in a similar way as it eliminated P_2 from g_1^* and g_2. We repeat this procedure for all the remaining atoms P_i, until we eliminate the last atom P_{m+1}, which concludes the inference. The complexity of the procedure is proportional to the size of the largest potential it handles. For elimination of each atom P_i, the size of the largest potential we handle is $O(n^4)$, proportional to the number of histograms in the range of a counting formula $\#_X[P_i(X), P_{i+1}(X)]$. As there are m atoms P_i, the whole procedure is in time $O(mn^4)$.

Joint conversion. Any solution based on *joint conversion* and *just different* counting conversion is less efficient than the above method. Here we present one such typical solution. Joint conversion first replaces the atoms P_1 and P_2 with a *joint* atom J_{12}, which represents the joint state of the atoms. This changes g_1 and g_2 respectively into $\phi_1'(J_{12}(X), J_{12}(Y))|X \neq Y$ and $\phi_2'(J_{12}(Y), P_3(X))|X \neq Y$. Still J_{12} cannot be summed-out from the model, due to the free logvar X in g_2. Just-different conversion, to derive a $\phi_1''(\#_X[J_{12}(X)])$ is not helpful either. The only option is to continue applying joint conversions between atoms such as $J_{12\ldots k}$ and P_{k+1}, to finally have only one joint atom $J_{1\ldots m+1}$ in the model. Note that $range(J_{1\ldots m+1}) = \{true, false\}^{m+1}$. The model would then consist of parfactors of the form $\phi_i(J_{1\ldots m+1}(X), J_{1\ldots m+1}(Y))|X \neq Y$. By multiplying these m parfactors into one, and then just-different conversion, we derive a parfactor $g^* = \phi^*(\#_X[J_{1\ldots m+1}(X)])$. Finally we can sum-out the counting formula $\gamma^* = \#_X[J_{1\ldots m+1}]$ from g^*. The complexity of these operations is dominated by the manipulation of the counting formula γ^* and is proportional to $|range(\gamma^*)|$ which is $O(n^{2^m})$. Comparing this complexity to the complexity of our approach, $O(mn^4)$, we see that our method can be much more efficient than Apsel and Brafman (2011)'s approach [1]. For example, for $m = 10$, the latter approach has complexity $O(n^{1024})$, in contrast to our approach which has complexity $O(10n^4)$.

8 Conclusion

Counting is one of the fundamental techniques used in lifted inference to exploit the symmetries among interchangeable randvars. In lifted variable elimination this technique works by manipulation of counting formulas. In this paper we showed how generalizing the structure of counting formulas, along with the lifted operators that manipulate them, provides more opportunities for lifting. Our approach is closely related to the method of joint conversion [1], but can offer more efficient solutions than this method.

Further generalizations of counting formulas, to capture a greater range of symmetries, are also conceivable. However, manipulation of such formulas, and especially counting the number of isomorphic states, can easily lead to non-trivial combinatorial problems. Future research on such problems can bring valuable insights for lifted inference.

Acknowledgements

NT is supported by GOA 08/008 'Probabilistic Logic Learning'. JD is partially supported by the Research Fund K.U.Leuven (CREA/11/015 and OT/11/051), EU FP7 Marie Curie Career Integration Grant (#294068), and FWO-Vlaanderen (G.0356.12). We wish to thank Daan Fierens for many insightful comments and discussions.

References

[1] Udi Apsel and Ronen I. Brafman. Extended lifted inference with joint formulas. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 11–18. AUAI Press, 2011.

[2] Rodrigo de Salvo Braz. *Lifted first-order probabilistic inference*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2007.

[3] Lise Getoor and Ben Taskar, editors. *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[4] Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted probabilistic inference with counting formulas. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1062–1608, 2008.

[5] David Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 985–991, 2003.

[6] David Poole and Nevin Lianwen Zhang. Exploiting contextual independence in probabilistic inference. *J. Artif. Intell. Res. (JAIR)*, 18:263–313, 2003.