Completeness Results for Lifted Variable Elimination

Nima Taghipour Daan Fierens Guy Van den Broeck Jesse Davis Hendrik Blockeel
Department of Computer Science, KU Leuven, Belgium

Abstract

Lifting aims at improving the efficiency of probabilistic inference by exploiting symmetries in the model. Various methods for lifted probabilistic inference have been proposed, but our understanding of these methods and the relationships between them is still limited, compared to their propositional counterparts. The only existing theoretical characterization of lifting is a completeness result for weighted first-order model counting. This paper addresses the question whether the same completeness result holds for other lifted inference algorithms. We answer this question positively for lifted variable elimination (LVE). Our proof relies on introducing a novel inference operator for LVE.

1 INTRODUCTION

Probabilistic logical models combine graphical models with elements of first-order logic to compactly model uncertainty in structured domains (social networks, citation graphs, etc.) [3, 8]. These domains can involve a large number of objects, making efficient inference a challenge. Lifted probabilistic inference methods address this problem by exploiting symmetries present in the structure of the model [1, 5, 9, 11, 12, 13, 14, 17, 18, 20, 22]. The basic principle is to identify "interchangeable" groups of objects and perform an inference operation once per group instead of once per individual in the group. Researchers have proposed "lifted" versions of many standard propositional inference algorithms, including variable elimination [5, 13, 14], belief propagation [12, 17, 18], recursive conditioning [15], weighted model counting [9], and knowledge compilation [21, 22].

Appearing in Proceedings of the 16^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2013, Scottsdale, AZ, USA. Volume 31 of JMLR: W&CP 31.

Despite the progress made, we have far less insight into lifted inference methods than into their propositional counterparts. Only recently has a definition been proposed for lifted inference. *Domain-lifted* inference requires the time-complexity of inference to be at most polynomial in the domain size (number of objects) of the model [21]. In contrast, standard propositional inference is typically exponential in the domain size in probabilistic logical models. Given this definition, it is possible to characterize, in the form of completeness results, which classes of models always permit lifted inference. *Weighted first-order model counting (WFOMC)* is the first lifted algorithm shown to be complete for a non-trivial model class: Van den Broeck [21] showed that WFOMC is complete for 2-logvar models. These models can express many important regularities that commonly occur in real-world problems, such as (anti-)homophily and symmetry.

This raises the question whether WFOMC is fundamentally more powerful for lifted inference than other approaches, such as lifted variable elimination (LVE). In this paper, we address that question. We prove that LVE is complete for the same class of models that WFOMC was shown to be complete for. This theoretical result advances our understanding about the relationship between these two approaches. The completeness theorem is proven by extending a state-of-the-art algorithm for LVE with a new operator, called group-inversion, and showing that the new algorithm has the completeness property. We also show experimentally that this completeness is not only of theoretical importance: the new algorithm can be orders of magnitude faster than its incomplete predecessor.

2 REPRESENTATION

Many representation languages for probabilistic logical models have been proposed [8]. Like earlier work on LVE [5, 13, 14], we use a representation based on *parametrized random variables* and *parametric factors*. This representation combines random variables and factors (as used in factor graphs) with concepts from first order logic. The goal is to compactly define complex probability distributions over large sets of vari-

ables. We now introduce the necessary terminology.

We use the term 'variable' in both the logical and probabilistic sense. We use *logvar* for logical variables and *randvar* for random variables. We write variables in uppercase and values in lowercase.

Preliminaries. 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 each configuration of $\mathcal{A}_f$ to a real number. A *factor graph* is a set of factors F over randvars $\mathcal{A} = \bigcup_{f \in F} \mathcal{A}_f$ and determines a probability distribution $\mathcal{P}_F(\mathcal{A}) = \frac{1}{Z} \prod_{f \in F} \phi_f(\mathcal{A}_f)$, with Z a normalization constant.

The vocabulary consists of *predicates* (representing properties and relations), *constants* (representing objects) and *logvars*. A *term* is a constant or a logvar. An *atom* is of the form $P(t_1, t_2, \ldots, t_n)$, where P is a predicate and each argument t_i is a term. An atom is *ground* if all its arguments are constants. Each logvar X has a finite *domain* $\mathcal{D}(X)$, which is a set of constants. A *constraint* C 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 $\mathcal{D}(X_i)$ or a logvar in $\mathbf{X}$. A *substitution* $\theta = \{X_1 \to t_1, \ldots, X_n \to t_n\}$ maps logvars to terms. Applying a substitution θ to an atom (or term) a, replaces each occurrence of X_i in a with t_i; the result is denoted $a\theta$. When all t_i's are constants, θ is called a *grounding substitution*. Given a constraint C, we use $gr(\mathbf{X}|C)$ to denote the set of grounding substitutions to $\mathbf{X}$ that are consistent with C.

Parametrized randvars. The representation associates atoms with randvars. To this end, every predicate is assigned a *range*, i.e., a set of possible values, e.g., $range(BloodType) = \{a, b, ab, o\}$. A ground atom then represents a randvar, e.g., $BloodType(joe)$. The randvar/atom has the same range as the involved predicate. Unlike in first-order logic, a range is not limited to $\{true, false\}$ but can be any finite set.

To compactly encode distributions over many randvars, the concept of a *parametrized randvar (PRV)* was introduced [5, 13, 14]. A PRV is of the form $P(\mathbf{X})|C$, where $P(\mathbf{X})$ is an atom and C is a constraint on $\mathbf{X}$. A PRV represents a set of randvars. Concretely, the set of randvars represented by a PRV $\mathcal{V} = P(\mathbf{X})|C$ is denoted $RV(\mathcal{V})$ and is defined as $\{P(\mathbf{X})\theta | \theta \in gr(\mathbf{X}|C)\}$.

Example 1. Suppose $\mathcal{D}(X) = \mathcal{D}(Y) = \{a, b, c, d\}$, where a stands for the person *ann*, b for *bob*, etc. The PRV $Friends(X, Y)|X \neq Y$ represents a set of 12 randvars, namely $\{Friends(a, b), Friends(a, c), \ldots, Friends(d, c)\}$. Similarly, the (unconstrained) PRVs $Smokes(X)$ and $Drinks(X)$ each represent a set of 4 randvars. □

Parametric factors (parfactors). Like PRVs compactly encode sets of randvars, *parfactors* compactly encode sets of factors. A parfactor is of the form $\forall \mathbf{L} : \phi(\mathcal{A})|C$, 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 omit $\mathbf{L}$ and 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 2. We use the following as our running example. Below we abbreviate *Drinks* to D, *Friends* to F and *Smokes* to S. The parfactor

$$g_1 = \phi_1(S(X), F(X, Y), D(Y))|X \neq Y \qquad (1)$$

represents a set of 12 factors, namely $gr(g_1) = \{\phi_1(S(a), F(a, b), D(b)), \ldots, \phi_1(S(d), F(d, c), D(c))\}$. If we choose the entries in the potential ϕ_1 appropriately, we can use this parfactor to encode, for instance, that if X is a smoker and is friends with Y, then Y is likely to be a drinker. The parfactor

$$g_2 = \phi_2(F(X, Y), F(Y, X))|X \neq Y \qquad (2)$$

also represents 12 factors, and can be used to encode, for instance, that friendship is likely to be symmetric.□

Parfactor models. 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 [1, 5, 13, 14, 19, 20]. This section reviews the algorithm that we build on, namely C-FOVE [13].

Variable elimination calculates a marginal distribution by *eliminating* randvars in a specific order from the model until reaching the desired marginal [16]. To eliminate a single randvar V, it first *multiplies* all 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 *sets* of randvars) from parfactors (i.e., *sets* of factors). The outer loop of LVE is shown in Algorithm 1. As this shows, LVE works by applying a set of lifted *operators*. We now discuss the most basic operators. Beside these, LVE has *conversion* operators, which we discuss in Section 6.

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 G
end while
return G

Algorithm 1: The outer loop of LVE.

Lifted Sum-out. This operator sums-out a PRV, and hence all the randvars represented by that PRV, from the model. Lifted sum-out is applicable only under a precondition (each randvar represented by the PRV appears in exactly one grounding of exactly one parfactor in the model). 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. This operator 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 particular PRV by a single equivalent product parfactor.

Splitting and Shattering. These operators rewrite the model into a *normal* form in which, e.g., each pair of PRVs represent either identical or disjoint randvars.

4 COMPLETENESS OF LVE

Lifting can yield significant speedups over standard inference. This has been demonstrated empirically in large models where a lifted algorithm can conclude inference without grounding. There, a central feature of lifted inference is scalability w.r.t. the *domain size* (the number of objects in the model). This is formally captured in the definition of *domain-lifted* inference:

Definition 1 (Domain-lifted algorithm [21]) *A probabilistic inference algorithm is* domain-lifted *for a model G, query $\mathcal{Q}$ and evidence $\mathcal{E}$ iff it runs in polynomial time in $|\mathcal{D}_1|$, ..., $|\mathcal{D}_k|$, with $\mathcal{D}_i$ the domain of logvar $X_i \in logvar(G, \mathcal{Q}, \mathcal{E})$.*

Note that this definition of 'lifting' requires time *polynomial* in the domain size. This is to contrast with standard propositional inference, which is often exponential in the domain size for common probabilistic logical models. This definition allows us to evaluate lifted algorithms not only by empirical evaluation on specific models, but by theoretically characterizing their completeness w.r.t. useful model classes.

Definition 2 (Completeness [21]) *An algorithm is* complete *for a class $\mathcal{M}$ of models, if it is domain-lifted for all models $G \in \mathcal{M}$ and all ground queries $\mathcal{Q}$ and evidence $\mathcal{E}$.*

Intuitively, this means that we can analyze a model syntactically and know a priori whether lifting is possible. Among all the lifted inference algorithms, the only existing completeness results belongs to WFOMC, which was shown to be complete for 2-logvar models [21]. This refers to any model where each parfactor contains at most 2 logvars. While C-FOVE has a lifted solution for some 2-logvar models, it is not complete w.r.t. this class [21]. Consider the 2-logvar model from Example 2. C-FOVE can handle the model consisting only of the first parfactor g_1 in a lifted way (i.e., without grounding). However, including the second parfactor g_2 forces C-FOVE to ground the model and run inference with exponential complexity in the domain size. This raises the question whether this is due to an inherent limitation of LVE.

In this paper, we answer this question negatively by presenting a lifted inference solution for LVE for all 2-logvar models. For this, we introduce a novel lifted operator in C-FOVE, resulting in the C-FOVE$^+$ algorithm. We then derive the first completeness results for LVE, and prove that C-FOVE$^+$ is complete in the same sense as WFOMC.

Theorem 1 *C-FOVE$^+$ is a complete domain-lifted algorithm for 2-logvar models.*

Importance of the Result. Our completeness result furthers our understanding of the relation between LVE and lifted search based methods, which is an important problem in the field [9, 10, 15, 21]. Van den Broeck [21] showed that WFOMC is complete for the class of 2-WFOMC models. Any such model can be represented as a 2-logvar model, and vice versa (see the appendix). Our completeness result for LVE is thus equally strong as that of WFOMC.

The class of 2-logvar models includes many useful and often employed models in statistical relational learning. It can model multiple kinds of relations, including: *homophily* between linked entities, e.g., $\phi(Property(X), Related(X,Y), Property(Y))$; *symmetry*, e.g., $\phi(Friend(X,Y), Friend(Y,X))$; *antisymmetry*, e.g., $\phi(Smaller(X,Y), Smaller(Y,X))$; and *reflexivity*, e.g., $\phi(Knows(X,X))$. Theorem 1 guarantees that for these models, LVE can perform inference in time polynomial in the domain size.

Secondary Result. Beside our main result (Theorem 1), we also present a second result (Theorem 2), which is in line with a known result for lifted recursive conditioning [15]. This theorem applies to models that restrict the number of logvars per *atom*, whereas Theorem 1 restricts the number of logvars per *parfactor*.

Theorem 2 *C-FOVE$^+$ is a complete domain-lifted algorithm for the class of models in which each atom has at most 1 logvar.*

The next two sections develop the machinery that allows us to prove our completeness results.

5 A NEW OPERATOR: GROUP INVERSION

We now introduce a new lifted operator called *group inversion*. This operator is required to make LVE complete for important classes of models, as argued above. Group inversion generalizes the existing inversion operator of LVE [5, 14] and is inspired by the concept of disconnected groundings in lifted recursive conditioning [15]. We first review inversion, and then define group inversion.

5.1 Inversion

Lifted sum-out eliminates a PRV, i.e., a whole set of randvars, in a single operation. An important principle that it relies on is *inversion*, which consists of turning a sum of products into a product of sums [5, 14]. Consider the sum of products $\sum_i \sum_j i \cdot j$. If the range of j does not depend on i, it can be rewritten as $(\sum_j j)(\sum_i i)$, which is a product of sums. More generally, given n variables $x_1, \ldots, x_n$, with independent ranges, we have

$$\sum_{x_1} \sum_{x_2} \cdots \sum_{x_n} \prod_{i=1}^{n} f(x_i) = \prod_{i=1}^{n} \sum_{x_i} f(x_i).$$

Furthermore, if all x_i have the same range, this equals $(\sum_{x_1} f(x_1))^n$. That is, the summation can be performed for *only one* representative x_1 and the result used for all x_i.

Exactly the same principle can be applied in lifted inference for summing out randvars. Suppose we need to sum out $F(X,Y)$ from the parfactor $g_1 = \phi_1(S(X), F(X,Y), D(Y))$, of our running example (Example 2, Section 2). For each instantiation (x,y) of (X,Y), $F(x,y)$ has the same range, hence applying inversion yields

$$\sum_{F(a,a)} \sum_{F(a,b)} \cdots \sum_{F(d,d)} \prod_{\theta \in \Theta} g_1\theta = \prod_{\theta \in \Theta} \Big(\sum_{F(X,Y)\theta} g_1\theta \Big) \quad (3)$$

with $\Theta = gr(X,Y)$. This shows that we can perform the sum-out operations independently for each $F(X,Y)\theta = F(x,y)$, and multiply the results. Furthermore, since all the factors $g_1\theta$ are groundings of the same parfactor and have the same potential ϕ_1, the result of summing out their second argument $F(X,Y)\theta$ is also the same potential, denoted ϕ_1'. It thus suffices to only perform one instance of these sum-out operations and rewrite Expression 3 as

$$\prod_{(x,y)} \Big(\sum_{F(x,y)} \phi_1(S(x), F(x,y), D(y)) \Big) = \prod_{(x,y)} \Big(\phi_1'(S(x), D(y)) \Big) = gr(g_1')$$

where $g_1' = \phi_1'(S(X), D(Y))$. This is what lifted sum-out by inversion does: it directly computes the parfactor g_1' from $g_1 = \phi_1(S(X), F(X,Y), D(Y))$ by summing out $F(X,Y)$ from g_1 in a single operation. This single lifted operation replaces $|\Theta|$ sum-out operations on the ground level.

5.2 Group Inversion: Principle

Inversion only works when the summations are independent. Our first contribution is based on the following observation. When we cannot apply inversion because of dependencies between factors, we can still partition the factors (and the summations) into *groups* such that dependencies exist only among factors within a group, but not between groups. Furthermore, we can do this at the lifted level: we can compute the result for one group and use it for all groups, provided that these groups are *isomorphic*, i.e., that there exists a one-to-one-mapping of the randvars from one group to the others such that exactly the same sum of products is obtained.

Consider the problematic parfactor from Example 2, $g_2 = \phi_2(F(X,Y), F(Y,X)) | X \neq Y$. Figure 1 depicts this model graphically. Consider summing out $F(X,Y)$ from this model. If we focus on the part of the computation related to one particular instantiation (a,b), the sum looks as follows.

$$\ldots \sum_{F(a,b)} \sum_{F(b,a)} \phi_2(F(a,b), F(b,a)) \cdot \phi_2(F(b,a), F(a,b))$$

The product contains two factors over the considered pair of randvars $F(a,b)$ and $F(b,a)$. Inversion is not applicable here, since the product of these two cannot be moved out of the summation over either randvar. Still, the two summations are independent of all other factors, and can be isolated from the rest of the computation. The same can be done for each pair of instantiations $\{(x,y),(y,x)\}$ of (X,Y), corresponding to each pair of factors *grouped* in the same box in Figure 1. This means that summing out $F(X,Y)$ from g_2 can be done using *group inversion*:

$$\sum_{F(a,b)} \sum_{F(a,c)} \cdots \sum_{F(d,c)} \Big(\prod_{\theta_{xy} \in \Theta} g_2\, \theta_{xy} \Big)$$

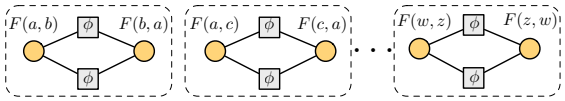

Figure 1: Group inversion on pairs of randvars. Circles represent randvars, squares represent factors. Dashed boxes indicate the partitioning into groups.

$$= \prod_{\{\theta_{xy},\theta_{yx}\}\in\Theta} \Big(\sum_{F(X,Y)\theta_{xy}} \sum_{F(X,Y)\theta_{yx}} g_2\,\theta_{xy} \cdot g_2\,\theta_{yx} \Big)$$

where θ_{xy} is a grounding substitution $\{X \to x, Y \to y\}$ in $\Theta = gr(X, Y|X \neq Y)$. Lifting is now possible again since the groups are isomorphic (see Figure 1): for all distinct pairs of substitutions $(\theta_{xy}, \theta_{yx})$ in Θ, the pairs of factors $(g_2\theta_{xy}, g_2\theta_{yx})$ share the same potential ϕ_2. Thus, the multiplicands of each pair also have the same potential ϕ_2', and summing out their arguments results in the same potential ϕ_2''. Hence, it suffices to perform only one (lifted) instance of these operations as follows

$$\prod_{\{(x,y),(y,x)\}} \Big(\sum_{F(x,y)} \sum_{F(y,x)} \phi_2'(F(x,y), F(y,x)) \Big)$$
$$= \prod_{\{(x,y),(y,x)\}} \phi_2''() = \prod_{(x,y)} \phi_2''()^{1/2} = gr(g_2'),$$

where g_2' is the parfactor $\forall X, Y : \phi_2''()|\ X \neq Y$ with ϕ_2'' a potential function with no arguments, i.e., a constant, because both arguments have been summed-out.

Group inversion partitions the set of factors (and randvars) into independent and isomorphic groups. An important question is what such a partitioning looks like. Figure 1 shows this for the above example. In general, let us call two factors *directly linked* if they share a randvar, and let *linked* be the transitive closure of this relation. Factors that are linked end up in the same group. Sometimes this yields useful partitionings, sometimes not. As a 'negative' example, consider a parfactor $\phi(P(X), P(Y))$. Any two factors $\phi(P(a), P(b))$ and $\phi(P(c), P(d))$ are linked, since both are directly linked to $\phi(P(b), P(c))$. Hence the only option is the trivial partition in which all factors are in a single, large group, which is not practically useful. As a 'positive' example, consider the case where each atom uses all the logvars in the parfactor, as in the earlier example $\phi(F(X,Y), F(Y,X))|X \neq Y$. In such cases, we can always partition the randvars into groups whose size is independent of the domain size. The reason is that in such cases, the arguments of the atoms in a linked group are necessarily *permutations* of each other. Hence the size of a linked group can be no larger than the number of possible permutations, which is independent of the domain size. We use this property in our group inversion operator.

5.3 The Group Inversion Operator

In- and output. Group inversion takes a parfactor $g = \phi(\mathcal{A})|C$ and a set of atoms $\{A_1, \ldots, A_n\} \subseteq \mathcal{A}$ as input. It returns a new parfactor that is the result of summing out $\{A_1, \ldots, A_n\}$ from g.

Preconditions. Group inversion is applicable when **(i)** for all i, j: $RV(A_i|C) = RV(A_j|C)$, **(ii)** each A_i has all the logvars $\mathbf{L}$ in the parfactor, **(iii)** for each pair of logvars X_i, X_j, with $\mathcal{D}(X_i) = \mathcal{D}(X_j)$, there is an inequality constraint $X_i \neq X_j$ in C, and **(iv)** for each PRV $\mathcal{V}$ outside g: $RV(A_i|C) \cap RV(\mathcal{V}) = \emptyset$. The key observation is that, in such a parfactor, due to conditions (i) and (ii), for each $i \neq j$, randvar A_i is a permutation of A_j. That is, $\lambda_{ij}(A_i) = A_j$, where λ_{ij} is a permutation of the logvars and $\lambda(A_i)$ represents the result of applying λ on the arguments of A_i.

Operator. When the preconditions hold, group inversion applies the following four steps. We further explain these steps below.

1. *Partition.* Find the set Λ of permutations λ_{ij} such that $\lambda_{ij}(A_i) = A_j$. Then find the closure $[\Lambda]$ of Λ, i.e., the minimal set $[\Lambda]$ of permutations such that $\Lambda \subseteq [\Lambda]$, and $[\Lambda]$ is closed under composition.
2. *Multiply* to compute the parfactor $g_{[\Lambda]} = \phi'(\mathcal{A}')|C$ as the product $\prod_{\lambda\in[\Lambda]} g_\lambda$, where $g_\lambda = \phi(\lambda(\mathcal{A}))|C$.
3. *Sum-out* to compute $g' = \phi''(\mathcal{A}'')|C = \sum_{\mathcal{A}_{[\Lambda]}} g_{[\Lambda]}$, where $\mathcal{A}_{[\Lambda]} = \{\lambda(A_1)|\lambda \in [\Lambda]\}$.
4. *Scale.* Return $g'' = \phi''(\mathcal{A}'')^{1/m}|C$, with $m = |[\Lambda]|$.

Step 1 (partition). The goal here is to find, on the lifted level, the set of factors and randvars that need to be put (and summed-out) in the *same group*. Because of the preconditions of the operator, we know that each pair of factors $(g\theta, g\theta')$ in $gr(g)$ that are directly linked can be derived from each other by a *permutation* of constants. Concretely, if randvar $A_i\theta$ in $g\theta$ is the same as $A_j\theta'$ in $g\theta'$, then we have $\theta = \lambda_{ij}(\theta')$.[1] All factors that are directly linked to a factor $g\theta$ are thus in the set $\{g\theta'|\theta' = \lambda(\theta), \lambda \in \Lambda\}$. In Step 1, we find the set of permutations Λ that convert directly linked factors to each other. Since directly linked factors are derived from each other by a permutation in Λ, each factor linked to $g\theta$ can be written as $g\lambda(\theta)$, where λ is a *composition* of permutations in Λ. We can thus find all such factors by computing the *closure* of Λ under the operation of composition, that is, the minimal set $[\Lambda] \supseteq \Lambda$, such that $\forall\lambda_1, \lambda_2 \in [\Lambda] : \lambda_1.\lambda_2 \in [\Lambda]$. Note that all the permutations are computed on the lifted level, i.e., as permutations of logvars, not constants.

[1] Note that for any atom A_i, and a permutation λ of its logvars, $A_i\lambda(\theta) = \lambda(A_i)\theta$.

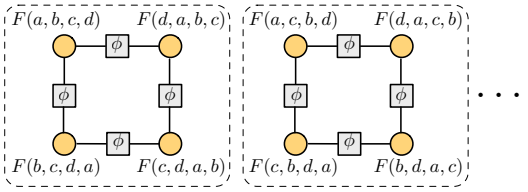

Figure 2: Group inversion for the parfactor $\phi(F(W,X,Y,Z),F(Z,W,X,Y))|W \neq X \neq Y \neq Z$, which partitions the randvars and factors into groups of size four.

Example 3. Consider the model shown in Figure 2, defined by a parfactor with two atoms $A_1 = F(W,X,Y,Z)$ and $A_2 = F(Z,W,X,Y)$. We partition this model based on the closure $[\Lambda]$ for A_1 and A_2. Let us denote a permutation $\lambda = \{W \to W', \ldots Z \to Z'\}$ as $\lambda = (W', \ldots, Z')$. Note that a right-shift of A_1's logvars yields A_2. Thus, Λ includes the three permutations $\lambda_{11} = \lambda_{22} = (W,X,Y,Z)$, $\lambda_{12} = (Z,W,X,Y)$ and $\lambda_{21} = (X,Y,Z,W)$ (respectively identity, right- and left-shift). These permutations map each randvar $F(a,b,c,d)$ to $F(d,a,b,c)$, and $F(b,c,d,a)$ that are directly linked to it, but not to $F(c,d,a,b)$ that is in turn directly linked to these two (see Figure 2). This is because Λ is *not* closed under composition: the composition $\lambda' = \lambda_{12}.\lambda_{12} = (Y,Z,W,X)$ is not in Λ. Adding λ' to Λ, however, yields a closed set of permutations, as no other possible composition results in a permutation out of this set. As such, we have $[\Lambda] = \Lambda \cup \{\lambda'\}$. Note that $[\Lambda]$ includes only 4 of all the possible $4! = 24$ permutations. Correspondingly, the model is partitioned into groups of size 4, as shown in Figure 2. □

Step 2 (multiply). Next we apply lifted multiplication to compute a parfactor $g_{[\Lambda]} = \prod_{\lambda \in [\Lambda]} g_\lambda$, where $g_\lambda = \phi(\lambda(\mathcal{A}))|C$. This parfactor has the same form as the product of all the factors in a representative group.

Step 3 (sum-out). Next we perform lifted sum-out, which sums-out the set of atoms $\{\lambda(A_i)|\lambda \in [\Lambda]\}$ from parfactor $g_{[\Lambda]}$ and yields parfactor g'. At this point, all randvars $RV(A_i|C)$ have been eliminated from the model, which is the goal of the entire procedure.

Step 4 (scale). For correctness, we still need to scale g', as it represents multiple equivalent factors. g' has the same constraint as g and hence the same set of all possible grounding substitutions Θ. Hence $gr(g')$ represents $|[\Lambda]|$ equivalent factors instead of one factor, for each group of the factors in $gr(g)$.[2] Hence, we replace the potential ϕ'' of g' with $\phi''^{1/|[\Lambda]|}$ to preserve the distribution.

[2]This equivalence follows from the fact that each of them can be derived from a same set of factors, but with a different order of multiplications in Step 2.

The following illustrates all these steps in a complete group inversion procedure on our running example.

Example 4. Consider eliminating the F atoms from the model defined in Example 2. To sum-out, we first multiply parfactors g_1 and g_2 to compute the product $g = \phi(S(X),F(X,Y),F(Y,X),D(Y))\ |X \neq Y$. Next we use group inversion. In Step 1, we find the permutation group $[\Lambda]$ for $A_1 = F(X,Y)$ and $A_2 = F(Y,X)$. The group $[\Lambda] = \{\lambda, \lambda'\}$ consists of the identity permutation $\lambda = \{X \to X, Y \to Y\}$, and the permutation $\lambda' = \{X \to Y, Y \to X\}$. In Step 2, we multiply $g_\lambda = g$ and $g_{\lambda'} = \phi(S(Y),F(Y,X),F(X,Y),D(X))|X \neq Y$, which results in the parfactor $g_{[\Lambda]} = \phi'(S(X),D(X),F(X,Y),F(Y,X),S(Y),D(Y))|X \neq Y$. In Steps 3 and 4, we respectively sum-out atoms $F(X,Y),F(Y,X)$ from $g_{[\Lambda]}$ and scale the resulting potential, to compute the parfactor $g'' = \phi''(S(X),D(X),S(Y),D(Y))^{1/2}|X \neq Y$. This concludes group inversion for elimination of F atoms. In Section 6, we see how LVE can eliminate the remaining atoms in g'' in a lifted way. □

As mentioned before, group inversion has inversion as a special case (namely when there is only one atom in the parfactor that covers the randvars $RV(A_i|C)$, i.e., when $n = 1$ in the operator). In this case, the closure $[\Lambda]$ consists only of the identity permutation.

Theorem 3 *Lifted sum-out with the group inversion operator is sound, i.e., is equivalent to summing out the randvars on the ground level.*

The proof relies on showing that the corresponding ground operations are independent and isomorphic, see the appendix.

6 EXTENSION FOR COUNTING

A central concept in LVE (and in our completeness proofs) that we have not discussed yet is *counting*, which is essentially a tool for allowing more lifting to take place. This requires an extension of the parfactor representation with *counting formulas* [13]. We first briefly review counting formulas and the existing operators for handling them in LVE [1, 13, 19]. Then we extend our new group inversion operator with support for counting formulas.

Representation. A *counting formula* is of the form $\#_{X:C}[P(\mathbf{X})]$, with $X \in \mathbf{X}$ and C a constraint on X. We call X the *counted logvar*. A *ground counting formula* is a counting formula in which all arguments except the counted logvar are constants. Such a formula represents a *counting randvar* (CRV). The value of a CRV is a histogram of the form $\{(v_i,n_i)\}_{i=1}^{|range(P)|}$, showing for each value $v_i \in range(P)$ the number n_i of covered randvars whose state is v_i.

Example 5. $\#_{Y:Y\neq X}[F(X,Y)]$ is a counting formula. Assume $\mathcal{D}(X) = \mathcal{D}(Y) = \{a,b,c,d\}$, then $\#_{Y:Y\neq a}[F(a,Y)]$ is a ground counting formula. It represents a CRV that counts how many people are (and are not) friends with a. The value of this CRV depends on the value of the three randvars $\{F(a,b), F(a,c), F(a,d)\}$. For instance, if $F(a,b) = true$, $F(a,c) = false$ and $F(a,d) = true$, the value of the CRV is the histogram $\{(true,2),(false,1)\}$, meaning that a has two friends and one 'non-friend'. Note that while there are $2^{|\mathcal{D}(X)|-1}$ different joint values for the covered randvars, there are only $|\mathcal{D}(X)|$ different histograms for the CRV. □

Counting formulas can be introduced in the model by the following two operators.

(Just-different) Counting Conversion [1, 13]. This operator replaces an atom (in a particular factor) by a counting formula, e.g., replace $F(X,Y)$ by $\#_Y[F(X,Y)]$. This is applicable on a set of logvars that only appear in a single atom or in *just-different* atoms [1], i.e., pairs of atoms $P(X_1,\mathbf{X}), P(X_2,\mathbf{X})$ with a constraint $X_1 \neq X_2$.

Joint Conversion [1]. This auxiliary operator works on a pair of atoms $A(X), B(X)$ and replaces any occurrence of $A(.)$ or $B(.)$ with a *joint* atom $J_{AB}(.)$, whose range is the Cartesian product of the range of A and B. This is useful because it can enable counting conversion, namely when the result of the joint conversion is a model with just-different atoms.

Example 6. Consider the parfactor $g'' = \phi(S(X), D(X), S(Y), D(Y))|X \neq Y$, from Example 4, Section 5.3. Joint conversion on atoms $S(.)$ and $D(.)$ rewrites this parfactor as $\phi'(J_{SD}(X), J_{SD}(X), J_{SD}(Y), J_{SD}(Y))|X \neq Y$, which can be simplified to $\phi''(J_{SD}(X), J_{SD}(Y))|X \neq Y$. Note that $J_{SD}(X)$ and $J_{SD}(Y)$ are now just-different atoms. Next, just-different counting conversion rewrites this parfactor as $\phi'''(\#_X[J_{SD}(X)])$. This parfactor is now ready for application of lifted sum-out. □

Extension of Group Inversion. The above are existing operators. Now that we have counting formulas, we also need to support them in our new group inversion operator. Counting formulas, like atoms, can be eliminated by lifted sum-out. This can be done by group inversion with a small modification of the operator. Suppose $\{A_i\}_{i=1}^n$ is a group of counting formulas $A_i = \#_{X_i:C}[P(\mathbf{L}, X_i)]$, and let $\mathcal{A}_{[\Lambda]} = \{A'_1, \dots A'_m\}$. Our operator eliminates all the formulas using the same four steps as in Section 5.3, with the exception of the *sum-out* step (Step 3), which becomes:

$$\sum_{(h_1,\dots,h_m)\in range(A'_1,\dots,A'_m)} \Big(\big(\prod_{i=1}^{m} \text{NUM}(h_i)\big)\ g_{[\Lambda]} \Big)$$

where, for a histogram $h_i = \{(v_j, n_j)\}_{j=1}^r$ with $\sum_j n_j = n$, the coefficient $\text{NUM}(h_i)$ is the multinomial coefficient $\frac{n!}{\prod_j n_j!}$ representing the number of possible assignments to $RV(A'_i|C)$ that yield this histogram. This operator generalizes the existing sum-out operation of LVE [13].

7 PROOF OF COMPLETENESS

By including group inversion in LVE, we can now prove the correctness of Theorem 1 (Section 4), i.e., we can show that LVE is complete domain-lifted for the subclass of 2-logvar parfactor models. This is the first completeness result for LVE and the second for exact lifted methods in general (after WFOMC [21]). The concrete LVE algorithm considered is C-FOVE [13] extended with joint formulas [1] and our new group-inversion operator. We refer to it as C-FOVE$^+$.

Proof of Theorem 1. We show the proof here since it is 'constructive': it shows how C-FOVE$^+$ deals with 2-logvar models: first sum-out all 2-logvar atoms (atoms A with $|logvar(A)| = 2$), then 1-logvar atoms, then 0-logvar atoms. We then show that this procedure is domain-lifted.[3]

Step 1. We eliminate all **2-logvar** atoms in two steps. (a) We multiply the parfactors until there are no distinct pairs (A_i, A_j) of 2-logvar atoms in distinct parfactors (g_i, g_j), such that $RV(A_i|C_i) = RV(A_j|C_j)$. The resulting equivalent model M^* is a 2-logvar model, since multiplication preserves the number of logvars in the product [13]. (b) We eliminate each 2-logvar atom in M^* by group inversion. This is possible since all 2-logvar atoms that represent the same randvars are in the same parfactor, and have both of the logvars. The result is a 1-logvar model.

Step 2. We eliminate all **1-logvar** atoms in four steps. (a) We (repeatedly) perform joint conversion on a pair of atoms $P_1(X_1, \mathbf{c}_1)$ and $P_2(X_2, \mathbf{c}_2)$ (of distinct predicates). This replaces them with joint atoms $J_{12}(X_1, \mathbf{c}_{12})$ and $J_{12}(X_2, \mathbf{c}_{12})$, respectively. When no more such conversions are possible, any pair of logvars X_1, X_2 that are constrained as $X_1 \neq X_2$, appear only in pairs of just-different atoms (otherwise, a joint conversion is still possible between them). (b) We perform (just-different) counting conversion on all the logvars. (c) We multiply all the parfactors into one, which is trivially possible since the model contains no free logvars. In the resulting parfactor each argument is either a ground atom, or a counting formula of the form

[3] We assume that the model M is preemptively shattered and in normalized form [13, 15]. Any 2-logvar model M can be rewritten in poly time as an equivalent 2-logvar model M' that satisfies these conditions [15].

$\gamma_i = \#_{X_i}[J_{1\ldots k_i}(X_i)]$. (d) We sum-out the counting formulas. The result of this step is a model in which all arguments are ground, i.e., a 0-logvar model.

Step 3. We eliminate all the remaining (so **0-logvar**) non-query randvars. Inference is now performed at the ground level, i.e., with standard VE. This step concludes the inference process.

Complexity. All the above operations run in time polynomial in the domain of the logvars. The most expensive step is handling the counting formulas produced in Step 2b. The largest size for the range of these formulas is $O(n^r)$ where r is the largest range size among the (joint) atoms, and n is the largest domain size among logvars. As such the exponent r is independent of the domain size. The complexity of Step 2 is thus polynomial in the domain size. Step 3 has worst case complexity $O(m^c)$, with c the total number of symbols appearing in $(M, \mathcal{Q}, \mathcal{E})$, and m the largest size of range among the randvars. This complexity satisfies the definition of a domain-lifted algorithm [21]. As such, C-FOVE$^+$ is a domain-lifted algorithm for any 2-logvar model, and hence *complete* for this class of models. □

Proof of Theorem 2. The proof, which builds on the proof of Theorem 1, is provided in the appendix.

8 EMPIRICAL EVALUATION

In addition to our theoretical analysis, we also provide an empirical evaluation to illustrate the benefit of group inversion. We use the publicly available implementation of C-FOVE[4] and augment it with the group inversion operator, yielding the C-FOVE$^+$ system. We test both algorithms on two synthetic domains. The first is the symmetric *friends and smokers* model [21]. The second is a collective classification model containing the following parfactors: $\forall i, j \in \mathit{Classes}$: $\phi_{ij}(\mathit{Class}_i(P_1), \mathit{Link}(P_1, P_2), \mathit{Class}_j(P_2))\,|P_1 \neq P_2$, and $\phi_2(\mathit{Link}(P_1, P_2), \mathit{Link}(P_2, P_1))|P_1 \neq P_2$. We set $|\mathit{Classes}| = 2$ so this model has five parfactors in total.

Figure 3 illustrate the runtime performance for varying domain sizes. The original C-FOVE algorithm cannot solve either model in a lifted manner due to the presence of the ϕ_2 factors. Thus it resorts to ground VE, which quickly becomes intractable (it runs out of memory on a machine with 16 GBs of RAM). By employing group inversion, C-FOVE$^+$ can exploit the symmetry expressed in the ϕ_2 factors and solve both models on the lifted level. Inference is more expensive in the second model than in the first model for the same domain size. This occurs because of the number of unary predicates involved in the ϕ_{ij} parfactor

[4] `http://people.csail.mit.edu/milch/blog/`

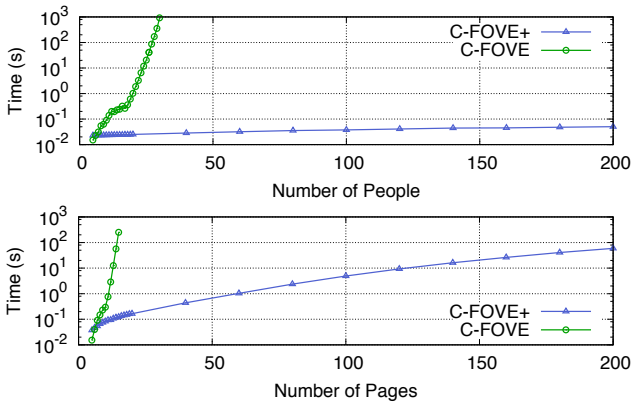

Figure 3: Comparison of performance, with varying domain size, on two models: symmetric *friends and smokers* (top), and *collective classification* (bottom).

in the collective classification model, which increases the complexity of the inference. See the appendix for a more detailed explanation.

9 CONCLUSION

We showed how introducing a new inference operator, called group inversion, makes lifted variable elimination a complete domain-lifted algorithm for 2-logvar models. A corollary of the completeness result is that lifted variable elimination and WFOMC are currently known to be domain-lifted complete for the same subclass of models.

An interesting direction for future work is derivation of (positive or negative) completeness results for useful models that fall outside of the 2-logvar class. An example are 3-logvar models containing a transitive relation, e.g. $\phi(\mathit{Like}(X, Y), \mathit{Like}(Y, Z), \mathit{Like}(X, Z))$, for which no domain-lifted inference procedure is known. We further believe that future research on the relationships between the various lifted inference algorithms will yield valuable theoretical insights, similar to those about the propositional inference methods [2, 6, 7].

Acknowledgements

NT is supported by the research fund KU Leuven (GOA/08/008 and CREA/11/015). JD is partially supported by the research fund KU Leuven (CREA/11/015 and OT/11/051), and EU FP7 Marie Curie Career Integration Grant (#294068). DF and GVB are supported by FWO-Vlaanderen.

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)*, pages 11–18, 2011.

[2] Adnan Darwiche. Recursive conditioning. *Artif. Intell.*, 126(1-2):5–41, 2001.

[3] Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors. *Probabilistic Inductive Logic Programming: Theory and Applications*. Springer-Verlag, Berlin, Heidelberg, 2008.

[4] Rodrigo de Salvo Braz. *Lifted First-order Probabilistic Inference*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2007.

[5] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1319–1325, 2005.

[6] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

[7] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007.

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

[9] Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 256–265, 2011.

[10] Manfred Jaeger and Guy Van den Broeck. Liftability of probabilistic inference: Upper and lower bounds. In *Proceedings of the 2nd International Workshop on Statistical Relational AI (StaRAI)*, pages 55–62, 2012.

[11] Abhay Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. Lifted inference seen from the other side : The tractable features. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, pages 973–981. 2010.

[12] Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 277–284, 2009.

[13] 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)*, pages 1062–1608, 2008.

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

[15] David Poole, Fahiem Bacchus, and Jacek Kisynski. Towards completely lifted search-based probabilistic inference. *CoRR*, abs/1107.4035, 2011.

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

[17] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Bisimulation-based approximate lifted inference. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI09)*, pages 496–505, 2009.

[18] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1094–1099, 2008.

[19] Nima Taghipour and Jesse Davis. Generalized counting for lifted variable elimination. In *Proceedings of the 2nd International Workshop on Statistical Relational AI (StaRAI)*, pages 1–8, 2012.

[20] Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. Lifted variable elimination with arbitrary constraints. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1194–1202, 2012.

[21] Guy Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Proceedings of the 24th Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 1386–1394, 2011.

[22] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2178–2185, 2011.