Lifted Variable Elimination with Arbitrary Constraints

Nima Taghipour **Daan Fierens** **Jesse Davis** **Hendrik Blockeel**
KU Leuven
Department of Computer Science
Celestijnenlaan 200A, 3001 Heverlee, Belgium

Abstract

Lifted probabilistic inference algorithms exploit regularities in the structure of graphical models to perform inference more efficiently. More specifically, they identify groups of interchangeable variables and perform inference once for each group, as opposed to once for each variable. The groups are defined by means of constraints, so the flexibility of the grouping is determined by the expressivity of the constraint language. Existing approaches for exact lifted inference rely on (in)equality constraints. We show how inference methods can be generalized to work with arbitrary constraints. This allows them to capture a broader range of symmetries, leading to more opportunities for lifting. We empirically demonstrate that this improves inference efficiency with orders of magnitude, allowing exact inference in cases where until now only approximate inference was feasible.

1 Introduction

Statistical relational learning (SRL) [1, 2] focuses on combining first-order logic with probabilistic graphical models, which permits algorithms to reason about complex, uncertain, structured domains. A major challenge in this area is how to efficiently perform inference. First-order logic can reason on the level of logical variables: one can derive $P(X)$ from $Q(X)$ without knowing what X is. Many approaches to SRL, however, transform their knowledge into a propositional graphical model before performing inference. By doing so, they lose the capacity to reason on the level of logical variables: standard inference methods for graphical models can reason only on the "ground" level, repeating the same inference steps for each different instance x of X, instead of once for all x's.

Addressing this problem, Poole [3] introduced the concept of lifted inference for graphical models. The idea is to group together indistinguishable objects, and perform the inference operations once for each group instead of once for each object. Multiple different algorithms have been proposed, but techniques generally focus on lifting either variable elimination (e.g., [3, 4, 5, 6]) or belief propagation (e.g., [7, 8]).

A group of indistinguishable objects is typically defined by means of a constraint that an object must fulfill in order to belong to that group. The type of constraints that are allowed, and the way in which they are handled, directly influence the granularity of the grouping, and hence, the efficiency of the subsequent lifted inference [9]. Until now, all approaches based on variable elimination use a specific class of constraints, namely, pairwise (in)equalities. This is the bare minimum required to be able to perform lifted inference. However, as we will show, it unnecessarily limits the symmetries the model can capture and exploit.

In this paper, we propose a system that can handle arbitrary constraints. The main contribution is the definition of operators for lifted inference that work correctly for any constraint language. Additionally, we propose a concrete mechanism for representing arbitrary constraints, and briefly discuss how the operators can be implemented with this particular mechanism. The new system performs lifted inference with a much coarser granularity than its predecessors. Due to this, it outperforms existing systems by several orders of magnitude, and solves inference problems that until now could only be solved by approximate inference methods.

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

2 Representation

We assume familiarity with set and relational algebra (union $\cup$, intersection $\cap$, set difference $\setminus$, selection σ_C, projection π_X, attribute renaming ρ, join $\bowtie$) ([10]).

The term "variable" can refer to a logical or a random variable. For clarity, we refer to logical variables as *logvars* and to random variables as *randvars*. Variables are denoted with uppercase letters and their values begin with lowercase letters.

Logical variables are typed and have a finite domain, which for a logvar X is denoted $D(X)$. A *term* is a logvar X or a constant $x \in D(X)$. An n-ary *predicate* P is a mapping from n-tuples of constants onto a range $range(P)$. An *atom* is of the form $P(t_1, t_2, \ldots, t_n)$, where the t_i are terms. A *ground atom* $P(x_1, \ldots, x_n)$ is an atom that contains only constants. We also write $P(\mathbf{x})$, where $\mathbf{x}$ is the tuple $(x_1, \ldots, x_n)$. Each ground atom is associated with one randvar, which can take any value in $range(P)$.

Given a set of logvars $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, a *constraint* $C_\mathbf{X}$ on $\mathbf{X}$ is a relation on $\mathbf{X}$, i.e., a subset of $D(\mathbf{X}) = \times_i D(X_i)$. A constraint may be defined extensionally, by listing the tuples that satisfy it, or intensionally, by means of some logical condition. We may write C for $C_\mathbf{X}$ when the logvars $\mathbf{X}$ are apparent from the context. A constraint satisfied by only one tuple is called *singleton*.

A *parametrized randvar (PRV)* $\mathcal{V}$ is a pair (a, C), where $a = P(X_1, \ldots, X_n)$ is a non-ground atom and C is a constraint on logvars $\mathbf{X} = \{X_1, \ldots, X_n\}$. Each PRV $\mathcal{V} = (a, C)$ represents the set of randvars $\{P(\mathbf{x}) | \mathbf{x} \in C\}$. We denote the set of randvars represented by $\mathcal{V}$ as $RV(\mathcal{V})$.

Example 1. The PRV $\mathcal{V} = (Smokes(X), C)$, with $C = \{x_1, \ldots, x_n\}$, represents n randvars $\{Smokes(x_1), \ldots Smokes(x_n)\}$.

A *factor* $f = (\mathcal{A}_f, \phi_f)$ defines a *potential* function $\phi_f : \times_{i=1}^n range(A_i) \to \mathbb{R}^+$ on a set of randvars $\mathcal{A}_f = \{A_i\}_{i=1}^n$. An *undirected model* is a set of factors F representing a probability distribution $\mathcal{P}_F$ on randvars $\mathcal{A} = \bigcup_{f \in F} \mathcal{A}_f$, where $\mathcal{P}_F(\mathbf{a})$ is proportional to $\omega_F(\mathbf{a}) = \prod_{f \in F} \phi_f(\mathbf{a}_f)$. ω_F is the *weighting function*.

A *parametric factor* or *parfactor* has the form $g = (L, C_\mathbf{X}, \mathcal{A}, \phi)$, with $L \subseteq \mathbf{X}$ a set of logvars, $C_\mathbf{X}$ a constraint, $\mathcal{A} = \{A_i\}_{i=1}^n$ an ordered set of atoms parametrized with $\mathbf{X}$, and ϕ a potential function on $\mathcal{A}$. A factor $\phi(\mathcal{A}')$ is a *grounding* of a parfactor $\phi(\mathcal{A})$ if $\mathcal{A}'$ can be obtained by instantiating $\mathbf{X}$ with some $\mathbf{x} \in C$. The set of groundings of a parfactor g is denoted $gr(g)$. A parfactor g defines a weighting function ω_g on randvars $RV(g) = \bigcup_{A_i \in \mathcal{A}} RV((A_i, C))$, such that $\omega_g = \prod_{f \in gr(g)} \omega_f$. For a set of parfactors G, $\omega_G = \prod_{g \in G} \omega_g$.

Example 2. Parfactor $g = (\{X\}, C_X, Smokes(X), \phi)$ with $C = \{(x_1), \ldots, (x_n)\}$ represents the set of factors $gr(g) = \{\phi(Smokes(x_1)), \ldots, \phi(Smokes(x_n))\}$.

Counting formulas. Milch et al. [5] introduced the idea of counting formulas and we now describe how to incorporate them in our formalism. A *counting formula* has the form $\#_{X_i}[P(\mathbf{X})]$, where $X_i \in \mathbf{X}$ is called the *counted logvar*. A *parametrized counting randvar* (PCRV) is a pair $(\#_{X_i}[P(\mathbf{X})], C)$. For each instantiation of $\mathbf{X} \setminus \{X_i\}$, it creates a separate counting randvar (CRV). The value of this CRV is a *histogram*, and it depends deterministically on the values of $P(\mathbf{X})$. More precisely, given a valuation for $P(\mathbf{X})$, it counts how many different values of X_i occur for each $r \in range(P)$. The result is a histogram of the form $\{(r_1, n_1), (r_2, n_2), \ldots, (r_k, n_k)\}$, with $r_i \in range(P)$ and n_i the corresponding count. Note that the range of a CRV, i.e., the set of all possible histograms it can take as a value, is determined by $k = |range(P)|$ and $\sum_{i=1}^k n_i$.

Example 3. $\mathcal{V} = (\#_Y[Friend(X, Y)], C)$ represents a set of randvars, one for each $x \in \pi_X(C)$, indicating the number of people who are (not) friends with person x. If $C = D(X) \times D(Y)$ with $D(X) = D(Y) = \{ann, bob, carl\}$, we might for instance have $\#_Y[Friend(ann, Y)] = \{(true, 1), (false, 2)\}$ (Ann is friends with 1 person and not with 2 persons).

Counting randvars do not replace any other randvars. They are simply a means of representing potential functions more compactly by exploiting their internal structure. The set of random variables represented by a counting formula is still $RV(\#_{X_i}[P(\mathbf{X})], C) = RV(P(\mathbf{X}), C)$. By definition in a parfactor g, L does not contain the counted logvars. When $\mathcal{A}$ contains a counting formula, factors of $gr(g)$ contain a corresponding counting randvar, whose state in each possible world is determined by the state of the randvars it counts.

3 C-FOVE

The method that we introduce in this paper performs lifted variable elimination in undirected models, specified using the above representation. For reasons of comprehensibility and conciseness, we first briefly explain the state of the art in this area, which is the C-FOVE system [5]. In the next section, we explain

how our method differs from C-FOVE, and which advantages this implies.

Variable elimination (VE) is an exact inference technique that computes the marginal probability distribution for one particular randvar by visiting all other randvars in some order, called the *elimination order*. For each considered randvar V, it first applies *multiplication* (multiplying all the factors containing V into a single factor) and then applies *summing-out* (summing out V from that single factor). Similarly, first-order variable elimination (FOVE) computes the marginal probability distribution for one particular randvar (one grounding of a PRV) by repeatedly applying certain *operators*. Two of these, *lifted multiplication* and *lifted summing-out*, are lifted counterparts of VE's operators, but it also has additional operators, including *counting conversion* that introduces counting randvars, and several constraint manipulation operators.

When none of the lifted operators can be applied, C-FOVE resorts to propositionalization: it completely grounds the P(C)RVs and parfactors and performs inference on the ground level. This is a worst-case scenario; the more often it can be avoided, the better.

Lifted summing-out is, in a sense, the most important operator: this is where randvars or PRVs are eliminated, preferably in a lifted manner. But lifted summing-out can only be applied to parfactors that satisfy certain preconditions. The goal of all other operators, then, is to manipulate the parfactors into a form that satisfies these preconditions. In this sense, all operators except lifted summing-out can be seen as *enabling operators*. The outer level of the C-FOVE algorithm is therefore as follows. GC-FOVE tries to eliminate all (non-query) PRVs in a particular order. When a particular PRV needs to be eliminated, C-FOVE checks whether the preconditions for lifted summing-out hold. If not, C-FOVE applies one or more enabling operators until the preconditions are satisfied, then applies lifted summing-out.

The operators for multiplication, elimination, and counting conversion, as defined in C-FOVE, do not essentially change in our framework, so we do not discuss them in detail. The original descriptions are found in Milch et al. [5], and our reformulation of the operators and their exact preconditions in terms of relational algebra are provided in the online appendix of this paper [11]. What does change, are the constraint manipulation operators. These are the key operators in lifted inference, as they dictate which objects are grouped together. More flexibility in the grouping implies more and better opportunities to apply the lifted multiplication, elimination, and counting conversion operators. C-FOVE uses only (in)equality constraints; e.g., it can represent PRVs $Friend(X,Y)$, $Friend(ann,Y)$, $(Friend(X,Y), X \neq ann)$, but not $(Friend(X,Y), (X,Y) \in \{(ann,bob),(bob,carl)\})$. By developing constraint manipulation operators that can handle arbitrary constraints, we can obtain dramatical improvements in the symmetries we can capture. The following section explains how our operators differ from C-FOVE's, and how these differences affect efficiency.

4 Our Approach: GC-FOVE

The lifted inference operators only apply to groups where the objects are interchangeable. This section describes the operators that ensure that this condition holds, i.e., the operators that manipulate the constraints that define the groupings.

The key factor in the efficiency of lifted inference is the granularity of the groupings: coarser groupings lead to more operations being performed on the lifted level, and hence more efficient inference. We demonstrate that C-FOVE's constraint language, namely conjunction of pairwise (in)equalities, leads to unnecessarily partitioning the objects into overly fine groups. We solve this problem by allowing arbitrary constraints, which leads to much coarser groupings and more efficient inference. We call our approach **GC-FOVE** (generalized C-FOVE).

We focus on explaining the operators necessary for manipulating arbitrary constraints and how they differ from C-FOVE's. The online appendix [11] contains their implementation in terms of relational algebra.

4.1 Splitting and Shattering

In order to multiply two parfactors, they must be *shattered* against each other. Two parfactors, $g_1 = (L_1, C_1, \mathcal{A}_1, \phi_1)$ and $g_2 = (L_2, C_2, \mathcal{A}_2, \phi_2)$, are shattered against each other if any pairing of PRVs in these parfactors is 'proper', i.e., $\forall (A_1, A_2) \in \mathcal{A}_1 \times \mathcal{A}_2$: $RV(A_1, C_1)$ and $RV(A_2, C_2)$ are either identical or disjoint. *Shattering* is the process that establishes this condition by modifying the parfactors. We first explain our approach to shattering, and then compare it to C-FOVE's approach.

Our approach. To shatter, we first check for every pair of PRVs $\mathcal{V}_1 = (A_1, C_1)$ and $\mathcal{V}_2 = (A_2, C_2)$ whether they are proper or not. If the pair is not proper, this means that these PRVs *partially* overlap. This overlap is eliminated in two phases. First, we split the involved constraints. Given constraints C_1 and C_2 on the same logvars, *splitting on their overlap* yields the partitions $\mathbb{P}_1 = \{C_1 \cap C_2, C_1 \setminus C_2\}$ of C_1, and $\mathbb{P}_2 = \{C_1 \cap C_2, C_2 \setminus C_1\}$ of C_2. Second, the parfactors g_1 and g_2 are split

based on the partitionings $\mathbb{P}_1$ and $\mathbb{P}_2$ respectively. Let us illustrate this with an example.

Consider two parfactors g_1 with $\mathcal{A}_1 = N(X,Y), R(X,Y,Z)$ and $C_1 = \{\{x_i\}_{i=1}^{50} \times \{y_i\}_{i=1}^{50} \times \{z_i\}_{i=1}^{5}\}$, and g_2 with $\mathcal{A}_2 = N(X,Y)$ and $C_2 = \{\{x_{2i}\}_{i=1}^{25} \times \{y_i\}_{i=1}^{50}\}$. First, we compare the PRVs $(N(X,Y), \pi_{X,Y}(C_1))$ and $(N(X,Y), \pi_{X,Y}(C_2))$. These PRVs partially overlap, so shattering is necessary. To shatter, we first split the relevant constraints (namely $P_1 = \pi_{X,Y}(C_1)$ and $P_2 = \pi_{X,Y}(C_2)$) on their overlap. Note that P_1 equals $\{\{x_i\}_{i=1}^{50} \times \{y_i\}_{i=1}^{50}\}$ and P_2 is simply C_2. Splitting on overlap first partitions P_1 into two sets, namely the common part (i.e., the shared or overlapping constraints), $P_1^{com} = P_1 \cap P_2 = P_2$, and the remaining ('excluded') part, $P_1^{excl} = P_1 \setminus P_2 = \{\{x_{2i-1}\}_{i=1}^{25} \times \{y_i\}_{i=1}^{50}\}$. Next, C_1 is split into $C_1^{com} = \sigma_{X,Y \in P_1^{com}}(C_1)$, and $C_1^{excl} = C_1 \setminus C_1^{com}$. On the other hand, C_2 does not need to be split because $P_2 = P_2 \cap P_1$. After splitting the constraints, we split the parfactors themselves. Parfactor g_1 is split into two parfactors g_1^{com} and g_1^{excl} with arguments identical to g_1, except for the constraints $C_1^{com} = \{\{x_{2i}\}_{i=1}^{25} \times \{y_i\}_{i=1}^{50} \times \{z_i\}_{i=1}^{5}\}$ and $C_1^{excl} = \{\{x_{2i-1}\}_{i=1}^{25} \times \{y_i\}_{i=1}^{50} \times \{z_i\}_{i=1}^{5}\}$. Parfactor g_2 remains unmodified as its constraint was not split.

In general, our shattering procedure splits any two PRVs that are partially overlapping into at most two partitions each. Similarly, the involved parfactors are split into at most two partitions each. Next, we show how this contrasts with C-FOVE's approach.

C-FOVE's approach. C-FOVE's approach to shattering is equivalent to performing a series of splits as used in our approach. C-FOVE operates per logvar, so consider the case where a pair of PRVs $\mathcal{V}_1 = (A_1, C_1)$ and $\mathcal{V}_2 = (A_2, C_2)$ have only one logvar, X, in common. Then we have $C_X^{com} = \pi_X(C_1) \cap \pi_X(C_2)$ and $C_X^{ex1} = \pi_X(C_1) \setminus \pi_X(C_2)$. C-FOVE will then split the constraint C_1 into the partition $C_X^{com} \cup \bigcup_{x_i \in C_X^{ex1}} \{(x_i)\}$, and does the same for C_2. Each element of C_X^{ex1} is split off into its own separate partition. Once the constraints are split, the parfactors g_1 and g_2 are split accordingly. The result is that g_1 is split into $|C_X^{ex1}| + 1$ different parfactors. This is an unnecessarily fine partition that greatly reduces the degree of lifting that can still take place (in the limit, splitting into $|gr(g)|$ factors boils down to inference at the propositional level). The effect of splitting is even worse when the PRVs have more than one logvar in common. This contrasts with our approach, which always splits into at most two parfactors, yielding much coarser partitions than C-FOVE and hence leaving more opportunities for lifting. The improvement is due to the fact that we allow arbitrary constraints, whereas C-FOVE allows only pairwise (in)equalities, forcing it to split each element off separately.

4.2 Expansion of Counting Formulas

When manipulating parfactors with counting formulas, shattering must be combined with the operation of *expansion*. When shattering splits one group of randvars $RV(\mathcal{V})$ into a partition $\{\mathcal{V}_i\}_{i=1}^m$, any counting randvar γ that counts the values of $RV(\mathcal{V})$ needs to be *expanded*, i.e., replaced by the group of counting randvars $\{\gamma_i\}_{i=1}^m$, where each γ_i counts the values of randvars in $RV(\mathcal{V}_i)$. As we show below, expansion of a counting formula also requires modifying the potential function and its dimensions/size. We show that expansion based on arbitrary constraints yields potential functions that can be exponentially smaller than the potentials obtained from expansion with C-FOVE.

Our approach. For ease of exposition, we explain our approach with an example. Suppose that we need to shatter the following two parfactors, $g_1 = (\{\}, C_1, \#_X[S(X)], \phi_1)$ and $g_2 = (\{X\}, C_2, S(X), \phi_2)$, with $C_1 = \{(x_1), \ldots, (x_{100})\}$ and $C_2 = \{(x_1, \ldots, x_5)\}$. Splitting partitions C_1 into $C_1^{com} = C_1 \cap C_2$ and $C_1^{excl} = C_1 \setminus C_2$. This results in a partitioning of the randvars involved in the parfactors. Concretely, the original group of randvars in parfactor g_1, $\{S(x_1), \ldots S(x_{100})\}$, is partitioned into two groups, $\mathcal{V}_1^{com} = (S(X), C_1^{com}) = \{S(x_1), \ldots S(x_5)\}$ and $\mathcal{V}_1^{excl} = (S(X), C_1^{excl}) = \{S(x_6), \ldots S(x_{100})\}$. In order to preserve the semantics of the original counting formula, we now need two separate counting formulas, one for the group $\mathcal{V}_1^{com}$ and one for $\mathcal{V}_1^{excl}$. In other words, we replace the orginal counting formula that counted over 100 randvars by a combination of two counting formulas that count over 5 and 95 randvars respectively. We also need to replace the original potential $\phi_1(\#_X[S(X)])$ by $\phi_1^*(\#_{X_{com}}[S(X_{com})], \#_{X_{excl}}[S(X_{excl})])$, where $\phi_1^*()$ is defined such that it depends only on the sum of the two new counting randvars $\#_{X_{com}}[S(X_{com})]$ and $\#_{X_{excl}}[S(X_{excl})]$. The end effect is that the parfactor g_1 is replaced by the new parfactor $(\{\}, C_1^*, \langle \#_{X_{com}}[S(X_{com})], \#_{X_{excl}}[S(X_{excl})] \rangle, \phi_1^*)$, where $C_1^* = C_1^{com} \times C_1^{excl}$. This concludes the shattering and expansion.

C-FOVE's approach. C-FOVE uses expansion based on substitution [5]. Suppose that during shattering we again partition the constraint C_1 into $\{C_1^{com}, C_1^{excl}\}$. C-FOVE then splits off all the elements of C^{excl} from C, by adding each of these elements as a *separate* argument of the parfactor and the involved potential function. In the above example, this results in a potential function $\phi_1^*()$ with 96 arguments, namely the counting randvar $\#_{X_{com}}[S(X_{com})]$ (that counts over 5 randvars) and the 95 randvars

$S(x_6), \ldots S(x_{100})$. Clearly this causes an extreme blow up in the size (number of entries) of the potential function, which does not happen using our approach. In general, C-FOVE's expansion yields a potential function of size $O(r^k) \times O((n-k)^r)$, where $n = |C_1|$ is the number of randvars counted over before expansion, $k = |C_1^{excl}|$, and r is the range of the considered randvars (e.g., the range of $S(.)$). In contrast, our expansion yields a potential function of size $O(k^r) \times O((n-k)^r)$. In the likely scenario that $r \ll k$, this is exponentially smaller than C-FOVE's potential function. Given that this potential function will later be used for multiplication or elimination, it is clear that our approach can yield large efficiency gains over C-FOVE. Again, this is due to our use of arbitrary constraints, as opposed to C-FOVE's pairwise (in)equalities.

4.3 Count Normalization

All the lifted operations of elimination, multiplication, and counting conversion require that a *count-normalization* property holds in the constraints. For instance, in lifted elimination this property is a precondition to ensure that parfactors receive the correct exponentiation after elimination. To be precise, for any constraint $C_{\mathbf{X}}$, with $\mathbf{Y} \subset \mathbf{X}$ and $\mathbf{Z} = \mathbf{X} - \mathbf{Y}$, we call $\mathbf{Y}$ *count-normalized* w.r.t. $\mathbf{Z}$ if and only if $\exists n : \forall \mathbf{z} \in \pi_{\mathbf{Z}}(C_{\mathbf{X}}) : |\pi_{\mathbf{Y}}(\sigma_{\mathbf{Z}=\mathbf{z}}(C_{\mathbf{X}}))| = n$. When this normalization property does not hold, it can be achieved by *normalizing* the involved parfactor, which amounts to splitting the parfactor into partitions in which the property does hold.

Our approach. Suppose that logvars $\mathbf{Y}$ need to be count-normalized w.r.t. logvars $\mathbf{Z}$ in a constraint C. Normalization operates on the projected constraint $C' = \pi_{\mathbf{Y},\mathbf{Z}}(C)$. Normalization partitions C' into maximally coarse groups $\{C'_1, \ldots C'_m\}$ such that for every group C_i it holds that all tuples t in that group have the same count $|\pi_{\mathbf{Y}}(\sigma_{\mathbf{Z}=\pi_Z(t)}(C'))|$. Intuitively, this count is the number of instantiations of logvars $\mathbf{Y}$ that are related to one instantiation of logvars $\mathbf{Z}$. As an example, consider the parfactor g with $\mathcal{A} = (Prof(P), Supervises(P,S))$ and constraint $C = \{(p_1,s_1), (p_1,s_2), (p_2,s_2), (p_2,s_3), (p_3,s_5), (p_4,s_3), (p_4,s_4), (p_5,s_6)\}$. Lifted elimination of $Supervises(P,S)$ requires logvar S (student) to be count-normalized with respect to logvar P (professor). Intuitively, we need to partition the professors into groups such that all professors in the same group supervise the same number of students. In our example, C needs to be partitioned into two, namely $C_1 = \sigma_{P \in \{p_3,p_5\}}(C) = \{(p_3,s_5),(p_5,s_6)\}$ (tuples involving professors with 1 student) and $C_2 = \sigma_{P \in \{p_1,p_2,p_4\}}(C) = \{(p_1,s_1),(p_1,s_2),(p_2,s_2),(p_2,s_3),(p_4,s_3),(p_4,s_4)\}$ (professors with 2 students). Next, the parfactor g is split accordingly into two parfactors g_1 and g_2 with constraints C_1 and C_2. These parfactors are now ready for lifted elimination of $Supervises(P,S)$.

C-FOVE's approach. C-FOVE requires a stronger normalization property to hold. Concretely, for every pair of logvars X and Y it requires either (1) $\pi_{X,Y}(C) = \pi_X(C) \times \pi_Y(C)$ or (2) $\pi_X(C) = \pi_Y(C)$ and $\pi_{X,Y}(C) = (\pi_X(C) \times \pi_Y(C)) \setminus \{\langle x_i, x_i \rangle : x_i \in \pi_X(C)\}$. To enforce this stronger property, C-FOVE requires finer partitions than our approach does. In our example, C-FOVE would require the constraint C to be split into 5 groups $\{C_i\}_{i=1}^5$ with $C_i = \sigma_{P \in \{p_i\}}(C)$, i.e., there is one group per professor. Again, the reason for this (overly) fine partitioning is that the coarser partitioning used in our approach cannot be represented using C-FOVE's constraint language.

4.4 Absorption: Handling Evidence

Evidence or observations of the states of randvars can make probabilistic inference more efficient since observed randvars do not need to be summed out. Furthermore, observations can introduce extra independencies in the probabilistic model, which can be exploited. In lifted probabilistic inference, there is also an undesired effect that observations can break the interchangeability of some randvars. It is crucial to handle observations in a manner that preserves as much interchangeability as possible, as this allows for more operations to take place on the lifted level. In order to effectively handle observations in a lifted manner, we introduce the novel operator of *lifted absorption*.

Our approach. We first explain how absorption works in the propositional setting. Given a factor $f = (A, \phi)$ and an observation $A_i = a_i$ about a randvar occurring in f (i.e., $A_i \in A$), absorption replaces the factor f with a new factor $f' = (A', \phi')$, where $A' = A \setminus A_i$ and $\phi'(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m) = \phi(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m)$. This reduces the size of the factor and might induce extra independencies in the model, which is always beneficial. If n randvars (built from the same predicate) have the same observed value, we can perform absorption on the lifted level by treating these n randvars as one interchangeable group. To better understand lifted absorption, consider a parfactor g with $\mathcal{A} = (P(X), Q(X,Y))$ and constraint $C = \{(x_1,y_1), \ldots, (x_1,y_{50})\}$. Assume that evidence atoms $Q(x_1,y_1)$ to $Q(x_1,y_{10})$ all have the value true. First, g needs to be split into two, namely g_1 with $C_1 = \{(x_1,x_1), \ldots, (x_1,x_{10})\}$ (the parfactor about which we have evidence) and g_2 with $C_2 = \{(x_1,y_{11}), \ldots, (x_1,y_{50})\}$ (no evidence). Next, we absorb the evidence about $Q()$ into parfactor g_1. Note

that performing absorption on the ground level results in ten identical factors $\phi'(p(x_1))$ (note that the logvar Y disappears in the absorption). Hence, lifted absorption (i.e., absorption into parfactor g_1) boils down to first instantiating $Q()$ in ϕ, yielding a reduced potential ϕ', and then exponentiating this potential with power 10. This explains the principle of lifted absorption for parfactors that have no counting formulas. Technicalities, including how to handle counting formulas, are in the online appendix [11].

C-FOVE's approach. C-FOVE handles evidence in a quite different way. C-FOVE introduces an additional so-called *evidence factor* for each ground observation $A = a$. This evidence factor assigns potential 1 to the observed value a and 0 to all other values. Including these factors in the probabilistic model effectively conditions the model on the observations. During inference, these evidence factors are used for multiplication and elimination, like any other factors. Our approach is more efficient in two ways. First, absorption boils down to *instantiating* a randvar in a factor, which means that this randvar no longer needs to be summed out. Hence, in our approach evidence reduces the number of elimination and multiplication operations needed, while in C-FOVE's approach evidence increases the number of operations. Second, we perform absorption on the lifted level, once for each group of randvars built from the same predicate and with the same observed value. In C-FOVE, introducing a separate evidence factor for each ground observation leads to splitting: if we have n randvars with the same observed value, there will be n partitions, one for each ground randvar. Hence C-FOVE will perform (at least) n multiplications and eliminations on these randvars, while we deal with them in a single lifted absorption operation. The splitting done by C-FOVE is clearly unnecessary. Additionally, it may cause further splitting as C-FOVE continues, further reducing the opportunities for lifting. We show in Section 6 that this can make inference with C-FOVE impossible in the presence of evidence.

5 Representing and Manipulating Arbitrary Constraints

We have shown that using arbitrary constraints instead of only pairwise (in)equalities can potentially yield large efficiency gains by allowing more opportunities for lifting. The question remains how we can represent these arbitrary constraints. In principle, we could represent them extensionally, as lists of tuples. This obviously allows any constraint to be represented, but is clearly inefficient when we have many logvars. Instead, we employ a *constraint tree* (as also used in

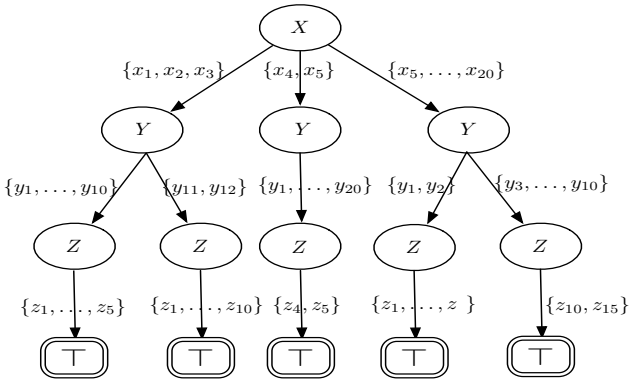

Figure 1: A constraint tree representing a constraint on logvars X, Y, Z.

First Order Bayesball [12]). Hence, the operators currently defined in terms of relational algebra must be translated to operations on the constraint trees. Below, we very briefly explain how this is done.

A constraint tree on logvars $\mathbf{X}$ is a tree in which each internal (non-leaf) node is labeled with a logvar $X \in \mathbf{X}$, each leaf is labeled with a terminal label $\top$, and each edge $e = (X_i, X_j)$ is labeled with a (sub-)domain $D(e) \subseteq D(X_i)$ (see Figure 1 for an example). We use ordered trees, where all nodes in the same level of the tree (with same distance from the root) are labeled with the same logvar, and there is a one-to-one correspondence between the logvars and the levels. Each path from the root to a leaf through edges $(e_1, \ldots, e_{|\mathbf{X}|})$ represents the tuples in the Cartesian product $\times_i D(e_i)$. For example, in Figure 1, the left most path represents the tuples $\{x_1, x_2, x_3\} \times \{y_1, \ldots, y_{10}\} \times \{z_1, \ldots, z_5\}$. The constraint represented by the tree is the union of tuples represented by each root-to-leaf path.

We perform constraint processing on the constraint trees. Given a constraint (in terms of the set of tuples that satisfy it), we construct the corresponding tree in a bottom-up manner by merging compatible edges (similar to the hypercube algorithm [13]). Different logvar orders can result in trees of different sizes. A tree can be *re-ordered* by interchanging nodes in two adjacent levels of the tree and applying the possible merges at those levels. We employ re-ordering to simplify the various constraint handling operations. For *projection* of a constraint, we move the projected logvars to the top of the tree and discard the parts below these logvars. For *splitting*, we perform a pairwise comparison of the two involved constraint trees. First we re-order each tree such that the logvars involved in the split are at the top of the trees. Then we process the trees top-down by comparing the edges leaving the root in the two trees and partitioning their domains

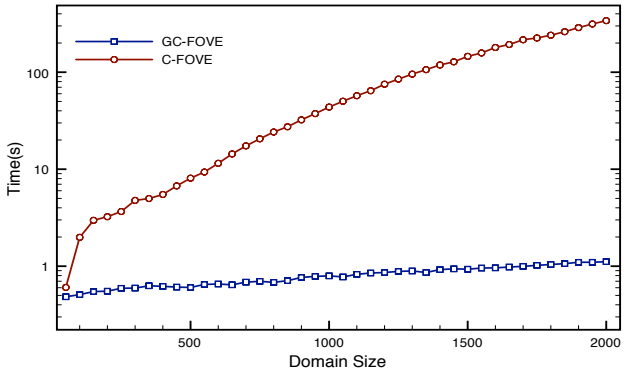

Figure 2: Performance on *workshop attributes*, for increasing domain sizes (evidence: 20%). Y-axis (runtime) is in log scale.

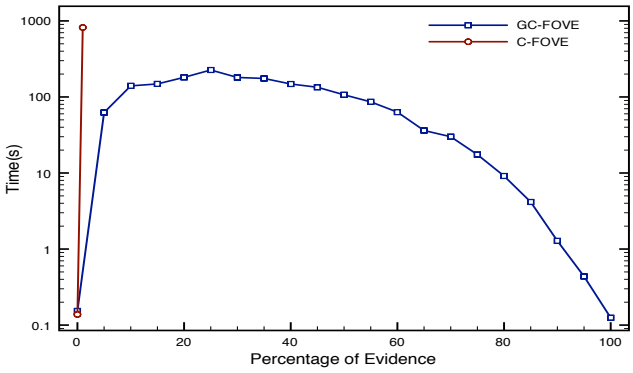

Figure 3: Performance on *social network*, for varying amounts of evidence (domain size: 1000). Y-axis (runtime) is in log scale.

based on their overlap. We recursively repeat this for their children until we reach the last logvar involved in the split. For *count normalization*, we also first apply this re-ordering. Then we partition the tree based on the number of tuples of counted logvars in each branch. For counting this number, we only need to consider the size of the domains associated with the edges.

Constraint trees are close to the *hypercube* representation used in lifted belief propagation [13]. However, for a given constraint, the constraint tree is typically more compact than using hypercubes. The constraint tree of Figure 1 corresponds to a set of five hypercubes. The first hypercube (derived from the left-most root-to-leaf path) represents the tuples $\{x_1, x_2, x_3\} \times \{y_1, \ldots, y_{10}\} \times \{z_1, \ldots, z_5\}$, the second hypercube represents $\{x_1, x_2, x_3\} \times \{y_{11}, y_{12}\} \times \{z_1, \ldots, z_{10}\}$, etc. The hypercube representation does not exploit that the first and second hypercube, for instance, share the part $\{x_1, x_2, x_3\}$. In the constraint tree, this is explicit, making the constraint tree more compact.

6 Experiments

Arbitrary constraints can capture more symmetries in the data, which potentially offers the ability to perform more operations at a lifted level. However, this comes at a cost, as manipulating arbitrary constraints is more computationally demanding. We hypothesize that the ability to perform fewer computations by capturing more symmetries will far outweigh this cost in typical inference tasks. In this section we validate this hypothesis empirically.

We compare our approach GC-FOVE to C-FOVE. We use the version of C-FOVE extended with general parfactor multiplication [14].[1] For implementing GC-FOVE, we started from the publicly available C-FOVE code [16], so the implementations are maximally comparable.[2] To quantify the impact of arbitrary constraints, we study the effect of two factors on the runtime of C-FOVE and GC-FOVE: the domain size (number of entities) and the proportion of randvars with observed values (amount of evidence). The evidence consists of observations on properties of individual entities, i.e., unary atoms. In all experiments, the undirected model has parfactors whose constraints are all representable by C-FOVE. Thus, GC-FOVE has no initial advantage, which makes the comparison conservative. All relevant information not included below (details on the models, additional plots, etc.) is in the online appendix [11].

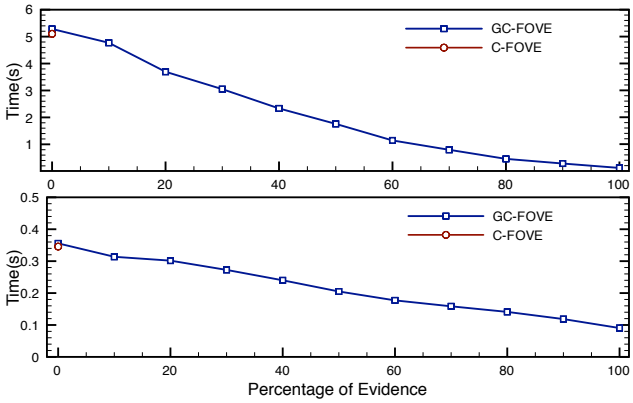

Figure 4: Performance on *Yeast* (top) and *WebKB* (bottom), for varying amounts of evidence. C-FOVE is intractable for any percentage of evidence except zero.

Synthetic data. We use three standard benchmarks from the lifted inference literature: *workshop*

[1] This allows C-FOVE to handle some tasks in an entirely lifted way, where otherwise it would have to resort to grounding, e.g. on the *social network* domain [15].

[2] Implementation of GC-FOVE available from `http://dtai.cs.kuleuven.be/ml/systems/gc-fove`.

attributes [5], *competing workshops* [5] and *social network* [15].

In the first set of experiments, we measure the effect of domain size on runtime. We vary the domain size from 50 to 2000 objects, holding the proportion of observed randvars constant at 20%. Figure 2 shows the results for the *workshop attributes* model. GC-FOVE outperforms C-FOVE as it better preserves the symmetries present in the model. GC-FOVE can treat all interchangeable elements, observed or not, as a single unit. The gain is more pronounced for larger domains because the number of partitions induced by C-FOVE grows linearly with the domain size and it has a costly elimination operation for each partition.

In the second set of experiments, we measure the effect of the proportion of observed randvars among unary atoms (binary atoms are unobserved). We fix the domain size and vary the percentage of observed randvars from 0% to 100% (observed randvars are assigned random values). Figure 3 shows the performance on the *social network* model. Without evidence, GC-FOVE is comparable to C-FOVE. This is the best scenario for C-FOVE as (i) the initial model only contains (in)equality constraints and (ii) there is no evidence, so no symmetries are broken when the inference operators are applied. In this case, the only difference in runtime between the two algorithms is the overhead associated with constraint processing, which is almost negligible. As the proportion of observations increases and the symmetries among the objects are broken, GC-FOVE achieves a much coarser grouping than C-FOVE. Furthermore, GC-FOVE's lifted absorption operator allows it to eliminate the evidence through instantiation and in a lifted fashion. The effect is striking: GC-FOVE consistently finishes in under 200 seconds, whereas C-FOVE cannot handle evidence proportions larger than 1%, as it runs out of memory.

Real-world data. We also used two real-world datasets: *WebKB* [17] and *Yeast* [18]. *WebKB* contains data about more than 1200 webpages, including their class (e.g., 'course page') and textual content (set of words), and the hyperlinks between the pages. The models consist of multiple parfactors, stating for instance how the classes of two linked pages depend on each other. The inference task that we consider is related to *link prediction*. Here, the class information is observed for a subset of all pages and the task is to compute the probability of having a hyperlink between pairs of pages. The *Yeast* dataset contains data about more than 7800 yeast genes, their functions and locations, and the interactions between these genes. The model and task are similar to those in *WebKB* (gene functions correspond to page classes, gene-to-gene interactions to hyperlinks). We use one class/function predicate in the model for each run, and average the runtime over multiple runs for each class/function.

We varied the percentage of observed classes from 0% to 100%. On both datasets, C-FOVE failed to run for any non-zero percentage of observations. Its failure is primarily due to the large number of observations, which often forces it to resort to perform inference at the ground level for a large number of objects. GC-FOVE, on the other hand, always runs successfully, in a few seconds. Figure 4 shows the performance of GC-FOVE with varying percentage of observed classes/functions. As on the synthetic data, GC-FOVE's performance improves with increasing number of observations, as this allows more randvars to be eliminated through absorption, instead of the more expensive operations of multiplication and summation.

These experiments confirm our hypothesis that in many typical settings the higher cost of constraint processing caused by using arbitrary constraints is more than compensated for by the additional symmetries that can be exploited.

7 Conclusions

Constraints play a crucial role in lifted probabilistic inference as they determine the degree of lifting that takes place. Surprisingly, most lifted inference algorithms use the same class of constraints based on pairwise (in)equalities [3, 4, 5, 15, 19, 20] (the main exception is the work on approximate inference using lifted belief propagation [8]). In this paper we have shown that this class of constraints is overly restrictive. We proposed using *arbitrary constraints*, which can capture more symmetries among the objects and allow for more operations to occur on a lifted level. We defined the relevant constraint handling operations (e.g., splitting, shattering and normalization) in terms of arbitrary constraints and implemented them for performing lifted variable elimination. We made use of constraint trees to efficiently represent and manipulate the constraints. We empirically evaluated our system on several domains. Our approach resulted in up to three orders of magnitude improvement in runtime. Furthermore, GC-FOVE can solve several tasks that are intractable for C-FOVE.

Acknowledgements

Nima Taghipour is supported by GOA/08/008 'Probabilistic Logic Learning'. Daan Fierens is a postdoctoral fellow of the Research Foundation-Flanders (FWO-Vlaanderen).

References

[1] Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors. *Probabilistic inductive logic programming: theory and applications*. Springer-Verlag, Berlin, Heidelberg, 2008.

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

[3] David Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI03)*, pages 985–991, 2003.

[4] 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 (IJCAI05)*, pages 1319–1325, 2005.

[5] 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 (AAAI08)*, pages 1062–1608, 2008.

[6] Jaesik Choi, David Hill, and Eyal Amir. Lifted inference for relational continuous models. In *UAI'10: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 126–134, Corvallis, Oregon, USA, 2010. AUAI Press.

[7] Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In J. Bilmes A. Ng, editor, *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI–09)*, Montreal, Canada, June 18–21 2009.

[8] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI08)*, pages 1094–1099, 2008.

[9] Jacek Kisynski and David Poole. Constraint processing in lifted probabilistic inference. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI09)*, 2009.

[10] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

[11] Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. Lifted variable elimination with arbitrary constraints. 2011. `http://dtai.cs.kuleuven.be/ml/systems/gc-fove`.

[12] Wannes Meert, Nima Taghipour, and Hendrik Blockeel. First-order bayes-ball. In José L. Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *ECML/PKDD (2)*, volume 6322 of *Lecture Notes in Computer Science*, pages 369–384. Springer, 2010.

[13] Parag Singla, Aniruddh Nath, and Pedro Domingos. Approximate Lifted Belief Propagation. In *AAAI Workshop on Statistical Relation AI*, pages 92–97, 2010.

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

[15] Abhay Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. Lifted inference seen from the other side : The tractable features. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 973–981. 2010.

[16] Brian Milch. BLOG, 2008. `http://people.csail.mit.edu/milch/blog/`.

[17] Mark Craven and Sean Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1/2):97–119, 1997.

[18] Jesse Davis, Elizabeth S. Burnside, Ines de Castro Dutra, David Page, and Vitor Santos Costa. An integrated approach to learning bayesian networks of rules. In *Proceedings of 16th European Conference on Machine Learning*, pages 84–95, 2005.

[19] Jacek Kisynski and David Poole. Lifted aggregation in directed first-order probabilistic models. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence (IJCAI09)*, 2009.

[20] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In Toby Walsh, editor, *International Joint Conference on Artificial Intelligence, Barcelona, Spain, 16-22 July 2011*, July 2011.