First-Order Decomposition Trees

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

Abstract

Lifting attempts to speedup probabilistic inference by exploiting symmetries in the model. Exact lifted inference methods, like their propositional counterparts, work by recursively decomposing the model and the problem. In the propositional case, there exist formal structures, such as decomposition trees (dtrees), that represent such a decomposition and allow us to determine the complexity of inference a priori. However, there is currently no equivalent structure nor analogous complexity results for lifted inference. In this paper, we introduce FO-dtrees, which upgrade propositional dtrees to the first-order level. We show how these trees can characterize a lifted inference solution for a probabilistic logical model (in terms of a sequence of lifted operations), and make a theoretical analysis of the complexity of lifted inference in terms of the novel notion of *lifted width* for the tree.

1 Introduction

Probabilistic logical modes (PLMs) combine elements of first-order logic with graphical models to succinctly model complex, uncertain, structured domains [5]. These domains often involve a large number of objects, making efficient inference a challenge. To address this, Poole [12] introduced the concept of *lifted probabilistic inference*, i.e., inference that exploits the symmetries in the model to improve efficiency. Various lifted algorithms have been proposed, mainly by *lifting* propositional inference algorithms [3, 6, 8, 9, 10, 13, 15, 17, 18, 19, 21, 22]. While the relation between the propositional algorithms is well studied, we have far less insight into their lifted counterparts.

The performance of propositional inference, such as variable elimination [4, 14] or recursive conditioning [2], is characterized in terms of a corresponding *tree decomposition* of the model, and their complexity is measured based on properties of the decomposition, mainly its *width*. It is known that standard (propositional) inference has complexity exponential in the treewidth [2, 4]. This allows us to measure the complexity of various inference algorithms only based on the structure of the model and its given decomposition. Such analysis is typically done using a secondary structure for representing the decomposition of graphical models, such as decomposition trees (*dtrees*) [2].

However, the existing notion of treewidth does not provide a tight upper bound for the complexity of lifted inference, since it ignores the opportunities that lifting exploits to improve efficiency. Currently, there exists no notion analogous to treewidth for lifted inference to analyze inference complexity based on the model structure. In this paper, we take a step towards filling these gaps.

Our work centers around a new structure for specifying and analyzing a lifted solution to an inference problem, and makes the following contributions. First, building on the existing structure of dtrees for propositional graphical models, we propose the structure of First-Order dtrees (FO-dtrees) for PLMs. An FO-dtree represents both the decomposition of a PLM and the symmetries that lifting exploits for performing inference. Second, we show how to determine whether an FO-dtree has a lifted solution, from its structure alone. Third, we present a method to read a lifted solution (a sequence of lifted inference operations) from a liftable FO-dtree, just like we can read a propositional inference solution from a dtree. Fourth, we show how the structure of an FO-dtree determines the complexity of inference using its corresponding solution. We formally analyze the complexity of lifted inference in terms of the novel, symmetry-aware notion of *lifted width* for FO-dtrees. As such, FO-dtrees serve as the first formal tool for finding, evaluating, and choosing among lifted solutions.[1]

2 Background

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 their values in lowercase. Applying a substitution $\theta = \{s_1 \to t_1, \ldots, s_n \to t_n\}$ to a structure S means replacing each occurrence of s_i in S by the corresponding t_i. The result is written $S\theta$.

2.1 Propositional and first-order graphical models

Probabilistic graphical models such as Bayesian networks, Markov networks and factor graphs compactly represent a joint distribution over a set of randvars $\mathcal{V} = \{V_1, \ldots, V_n\}$ by factorizing the distribution into a set of local distribution. For example, factor graphs represent the distribution as a product of *factors*: $Pr(V_1, \ldots, V_n) = \frac{1}{Z}\prod \phi_i(\mathcal{V}_i)$, where ϕ_i is a *potential* function that maps each configuration of $\mathcal{V}_i \subseteq \mathcal{V}$ to a real number and Z is a normalization constant.

Probabilistic logical models use concepts from first-order logic to provide a high-level modeling language for representing propositional graphical models. While many such languages exist (see [5] for an overview), we focus on *parametric factors* (parfactors) [12] that generalize factor graphs.

Parfactors use *parametrized randvars* (PRVs) to represent entire sets of randvars. For example, the PRV $BloodType(X)$, where X is a logvar, represents one $BloodType$ randvar for each object in the domain of X (written $\mathcal{D}(X)$). Formally, a PRV is of the form $P(\mathbf{X})|C$ where C is a *constraint* consisting of a conjunction of inequalities $X_i \neq t$ where $t \in \mathcal{D}(X_i)$ or $t \in \mathbf{X}$. It represents the set of all randvars $P(\mathbf{x})$ where $\mathbf{x} \in \mathcal{D}(\mathbf{X})$ and $\mathbf{x}$ satisfies C; this set is denoted $rv(P(\mathbf{X})|C)$.

A *parfactor* uses PRVs to compactly encode a set of factors. For example, the parfactor $\phi(Smoke(X), Friends(X, Y), Smoke(Y))$ could encode that friends have similar smoking habits. It imposes a symmetry in the model by stating that the probability that, among two friends, both, one or none smoke, is the same for all pairs of friends, in the absence of any other information.

Formally, a parfactor is of the form $\phi(\mathcal{A})|C$, where $\mathcal{A} = (A_i)_{i=1}^n$ is a sequence of PRVs, C is a constraint on the logvars appearing in $\mathcal{A}$, and ϕ is a potential function. The set of logvars occurring in $\mathcal{A}$ is denoted $logvar(\mathcal{A})$. A *grounding substitution* maps each logvar to an object from its domain. A parfactor g represents the set of all factors that can be obtained by applying a grounding substitution to g that is consistent with C; this set is called the grounding of g, and is denoted $gr(g)$. A parfactor model is a set G of parfactors. It compactly defines a factor graph $gr(G) = \{gr(g)|g \in G\}$.

Following the literature, we assume that the model is in a *normal form*, such that (i) each pair of logvars have either identical or disjoint domains, and (ii) for each pair of co-domain logvars X, X' in a parfactor $\phi(\mathcal{A})|C$, $(X \neq X') \in C$. Every model can be written into this form in poly time [13].

2.2 Inference

A typical inference task is to compute the marginal probability of some variables by summing out the remaining variables, which can be written as: $Pr(\mathcal{V}') = \sum_{\mathcal{V}\setminus\mathcal{V}'} \prod_i \phi_i(\mathcal{V}_i)$. This is an instance of the general sum-product problem [1]. Abusing notation, we write this sum of products as $\sum_{\mathcal{V}\setminus\mathcal{V}'} M(\mathcal{V})$.

Inference by recursive decomposition. Inference algorithms exploit the factorization of the model to recursively decompose the original problem into smaller, independent subproblems. This is achieved by a decomposition of the sum-product, according to a simple *decomposition rule*.

Definition 1 (The decomposition rule) *Let $\mathcal{P}$ be a sum-product computation $\mathcal{P} : \sum_{\mathcal{V}} M(\mathcal{V})$, and let $\mathbb{M} = \{M_1(\mathcal{V}_1), \ldots M_k(\mathcal{V}_k)\}$ be a partitioning (decomposition) of $M(\mathcal{V})$. Then, the* decomposi-

[1] Similarly to existing studies on propositional inference [2, 4], our analysis only considers the model's *global structure*, and makes no assumptions about its *local structure*.

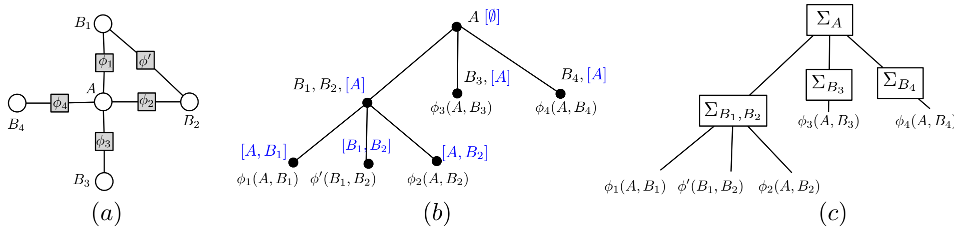

Figure 1: (a) a factor graph model; (b) a dtree for the model, with its node clusters shown as $cutset, [context]$; (c) the corresponding factorization of the sum-product computations.

tion of $\mathcal{P}$, w.r.t. $\mathbb{M}$ *is an equivalent sum-product formula* $\mathcal{P}_{\mathbb{M}}$*, defined as follows:*

$$\mathcal{P}_{\mathbb{M}} : \sum_{\mathcal{V}'} \Big[\big(\sum_{\mathcal{V}'_1} M_1(\mathcal{V}_1) \big) \dots \big(\sum_{\mathcal{V}'_k} M_k(\mathcal{V}_k) \big) \Big]$$

where $\mathcal{V}' = \bigcup_{i,j} \mathcal{V}_i \cap \mathcal{V}_j$*, and* $\mathcal{V}'_i = \mathcal{V}_i \setminus \mathcal{V}'$.

Most exact inference algorithms recursively apply this rule and compute the final result using top-down or bottom-up dynamic programming [1, 2, 4]. The complexity is then exponential only in the size of the largest sub-problem solved. Variable elimination (VE) is a bottom-up algorithm that computes the nested sum-product by repeatedly solving an innermost problem $\sum_V M(V, \mathcal{V}')$ to *eliminate* V from the model. At each step, VE eliminates a randvar V from the model by *multiplying* the factors in $M(V, \mathcal{V}')$ into one and *summing-out* V from the resulting factor.

Decomposition trees. A single inference problem typically has multiple solutions, each with a different complexity. A *decomposition tree* (dtree) is a structure that represents the decomposition used by a specific solution and allows us to determine its complexity [2]. Formally, a dtree is a rooted tree in which each leaf represents a factor in the model.[2] Each node in the tree represents a decomposition of the model into the models under its child subtrees. Properties of the nodes can be used to determine the complexity of inference. $Child(T)$ refers to T's child nodes; $rv(T)$ refers to the randvars under T, which are those in its factor if T is a leaf and $rv(T) = \cup_{T' \in Child(T)} rv(T')$ otherwise. Using these, the important properties of *cutset*, *context*, and *cluster* are defined as follows:

- $cutset(T) = \cup_{\{T_1,T_2\} \in child(T)} rv(T_1) \cap rv(T_2) \setminus acutset(T)$, where $acutset(T)$ is the union of cutsets associated with ancestors of T.
- $context(T) = rv(T) \cap acutset(T)$
- $cluster(T) = rv(T)$, if T is a leaf; otherwise $cluster(T) = cutset(T) \cup context(T)$

Figure 1 shows a factor graph model, a dtree for it with its clusters, and the corresponding sum-product factorization. Intuitively, the properties of dtree nodes help us analyze the size of subproblems solved during inference. In short, the time complexity of inference is $O(n \exp(w))$ where n is the size (number of nodes) of the tree and w is its *width*, i.e., its maximal cluster size minus one.

3 Lifted inference: Exploiting symmetries

The inference approach of Section 2.2 ignores the symmetries imposed by a PLM. Lifted inference aims at exploiting symmetries among a model's *isomorphic* parts. Two constructs are isomorphic if there is a structure preserving bijection between their components. As PLMs make assertions about whole groups of objects, they contain many isomorphisms, established by a bijection at the level of objects. Building on this, symmetries arise between constructs at different levels [11], such as between: randvars, value assignments to randvars, factors, models, or even sum-product problems. All exact lifted inference methods use two main tools for exploiting symmetries, i.e., for lifting:

1. Divide the problem into isomorphic subproblems, solve one instance, and aggregate
2. Count the number of isomorphic configurations for a group of interchangeable variables instead of enumerating all possible configurations.

[2]We use a slightly modified definition for dtrees, which were originally defined as full binary rooted trees.

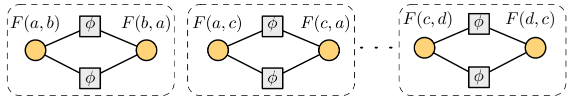

Figure 2: Isomorphic decomposition of a model. Dashed boxes indicate the partitioning into groups.

Below, we show how these tools are used by lifted variable elimination (LVE) [3, 10, 12, 17, 18].

Isomorphic decomposition: exploiting symmetry among subproblems. The first lifting tool identifies cases where the application of the decomposition rule results in a product of isomorphic sum-product problems. Since such problems all have isomorphic answers, we can solve *one* problem and reuse its result for all the others. In LVE, this corresponds to lifted elimination, which uses the operations of *lifted multiplication* and *lifted sum-out* on parfactors to evaluate a single representative problem. Afterwards, LVE also attempts to aggregate the result (compute their product) by taking advantage of their isomorphism. For instance, when the results are identical, LVE computes their product simply by *exponentiating* the result of one problem.

Example 1. Figure 2 shows the model defined by $\phi(F(X,Y),F(Y,X))|X \neq Y$, with $\mathcal{D}(X) = \mathcal{D}(Y) = \{a,b,c,d\}$. The model asserts that the friendship relationship (F) is likely to be symmetric. To sum-out the randvars F using the decomposition rule, we partition the ground factors into six groups of the form $\{\phi(F(x,y),F(y,x)), \phi(F(y,x),F(x,y))\}$, i.e., one group for each 2-subset $\{x,y\} \subseteq \{a,b,c,d\}$. Since no randvars are shared between the groups, this decomposes the problem into the product of six isomorphic sums $\sum_{F(x,y),F(y,x)} \phi(F(x,y),F(y,x)) \cdot \phi(F(y,x),F(x,y))$. All six sums have the same result c (a scalar). Thus, LVE computes c only once (lifted elimination) and computes the final result by exponentiation as c^6 (lifted aggregation).

Counting: exploiting interchangeability among randvars. Whereas isomorphic decomposition exploits symmetry among problems, counting exploits symmetries within a problem, by identifying *interchangeable* randvars. A group of (k-tuples of) randvars are *interchangeable*, if permuting the assignment of values to the group results in an equivalent model. Consider a sum-product subproblem $\sum_{\mathcal{V}} M(\mathcal{V},\mathcal{V}')$ that contains a set of n interchangeable (k-tuples of) randvars $\mathcal{V} = \{(V_{i1}, V_{i2}, \dots V_{ik})\}_{i=1}^{n}$. The interchangeability allows us to rewrite $\mathcal{V}$ into a single *counting randvar* $\#[\mathcal{V}]$, whose value is the histogram $h = \{(\mathbf{v}_1, n_1), \dots, (\mathbf{v}_r, n_r)\}$, where n_i is the number of tuples with joint state $\mathbf{v}_i$. This allows us to replace a sum over all possible joint states of $\mathcal{V}$ with a sum over the histograms for $\#[\mathcal{V}]$. That is, we compute $M(\mathcal{V}') = \sum_{i=1}^{m} \textsc{Mul}(h_i) \times M(h_i,\mathcal{V}')$, where $\textsc{Mul}(h_i)$ denotes the number of assignments to $\mathcal{V}$ that yield the same histogram h_i for $\#[\mathcal{V}]$. Since the number of histograms is $O(n^{\exp(k)})$, when $n \gg k$, we gain exponential savings over enumerating all the possible joint assignments, whose number is $O(\exp(n^k))$. This lifting tool is employed in LVE by *counting conversion*, which rewrites the model in terms of counting randvars.

Example 2. Consider the model defined by the parfactor $\phi(S(X),S(Y))|X \neq Y$, which is $\prod_{i \neq j} \phi(S(x_i),S(x_j))$. The group of randvars $\{S(x_1),\dots,S(x_n)\}$ are interchangeable here, since under any value assignment where n_t randvars are $true$ and n_f randvars are $false$, the model evaluates to the same value $\phi'(n_t,n_f) = \phi(t,t)^{n_t \cdot (n_t-1)} \cdot \phi(t,f)^{n_t \cdot n_f} \cdot \phi(f,t)^{n_f \cdot n_t} \cdot \phi(f,f)^{n_f \cdot (n_f-1)}$. By counting conversion, LVE rewrites this model into $\phi'(\#_X[S(X)])$.

4 First-order decomposition trees

In this section, we propose the structure of FO-dtrees, which compactly represent a recursive decomposition for a PLM and the symmetries therein.

4.1 Structure

An FO-dtree provides a compact representation of a propositional dtree, just like a PLM is a compact representation of a propositional model. It does so by explicitly capturing isomorphic decomposition, which in a dtree correspond to a node with isomorphic children. Using a novel node type, called a *decomposition into partial groundings (DPG)* node, an FO-dtree represents the *entire set* of

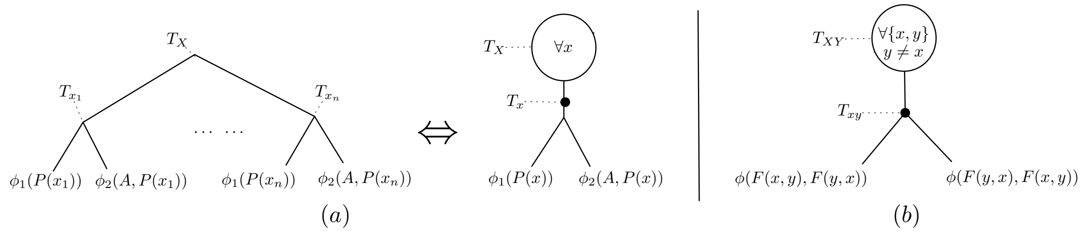

Figure 3: (a) dtree (left) and FO-dtree (right) of Example 3; (b) FO-dtree of Example 1

isomorphic child subtrees with a *single representative subtree*. To formally introduce the structure, we first show how a PLM can be decomposed into isomorphic parts by DPG.

DPG of a parfactor model. The DPG of a parfactor g is defined w.r.t. a k-subset $\mathbf{X} = \{X_1, \ldots, X_k\}$ of its logvars that all have the same domain $D_{\mathbf{X}}$. For example, the decomposition used in Example 1, and shown in Figure 2, is the DPG of $\phi(F(X, Y), F(Y, X))|X \neq Y$ w.r.t. logvars $\{X, Y\}$. Formally, $DPG(g, \mathbf{X})$ partitions the model defined by g into $\binom{|D_{\mathbf{X}}|}{k}$ parts: one part $G_{\mathbf{x}}$ for each k-subset $\mathbf{x} = \{x_1, \ldots, x_k\}$ of the objects in $D_{\mathbf{X}}$. Each $G_{\mathbf{x}}$ in turn contains all $k!$ (partial) groundings of g that can result from replacing $(X_1, \ldots, X_k)$ with a permutation of $(x_1, \ldots, x_k)$. The key intuition behind DPG is that for any $\mathbf{x}, \mathbf{x}' \subseteq_k \mathcal{D}_{\mathbf{X}}$, $G_{\mathbf{x}}$ is isomorphic to $G_{\mathbf{x}'}$, since any bijection from $\mathbf{x}$ to $\mathbf{x}'$ yields a bijection from $G_{\mathbf{x}}$ to $G_{\mathbf{x}'}$.

DPG can be applied to a whole model $G = \{g_i\}_{i=1}^m$, if G's logvars are (re-)named such that (i) only co-domain logvars share the same name, and (ii) logvars $\mathbf{X}$ appear in all parfactors.

Example 3. Consider $G = \{\phi_1(P(X)), \phi_2(A, P(X))\}$. $DPG(G, \{X\}) = \{G_i\}_{i=1}^n$, where each group $G_i = \{\phi_1(P(x_i)), \phi_2(A, P(x_i))\}$ is a grounding of G (w.r.t. X).

FO-dtrees simply add to dtrees special nodes for representing DPGs in parfactor models.

Definition 2 (DPG node) *A DPG node $T_{\mathbf{X}}$ is a triplet $(\mathbf{X}, \mathbf{x}, C)$, where $\mathbf{X} = \{X_1, \ldots X_k\}$ is a set of logvars with the same domain $D_{\mathbf{X}}$, $\mathbf{x} = \{x_1, \ldots, x_k\}$ is a set of* representative objects*, and C is a constraint, such that for all $i \neq j$: $x_i \neq x_j \in C$. We denote this node as $\forall \mathbf{x} : C$ in the tree.*

A representative object is simply a placeholder for a domain object.[3] The idea behind our FO-dtrees is to use $T_{\mathbf{X}}$ to graphically indicate a $DPG(G, \mathbf{X})$. For this, each $T_{\mathbf{X}}$ has a single child distinguished as $T_{\mathbf{x}}$, under which the model is a representative instance of the isomorphic models $G_{\mathbf{x}}$ in the DPG.

Definition 3 (FO-dtree) *An FO-dtree is a rooted tree in which*

1. *non-leaf nodes may be DPG nodes*
2. *each leaf contains a factor (possibly with representative objects)*
3. *each leaf with a representative object x is the descendent of exactly one DPG node $T_{\mathbf{X}} = (\mathbf{X}, \mathbf{x}, C)$, such that $x \in \mathbf{x}$*
4. *each leaf that is a descendent of $T_{\mathbf{X}}$ has all the representative objects $\mathbf{x}$, and*
5. *for each $T_{\mathbf{X}}$ with $\mathbf{X} = \{X_1, \ldots, X_k\}$, $T_{\mathbf{x}}$ has $k!$ children $\{T_i\}_{i=1}^{k!}$, which are isomorphic up to a permutation of the representative objects $\mathbf{x}$.*

Semantics. Each FO-dtree defines a dtree, which can be constructed by recursively *grounding* its DPG nodes. Grounding a DPG node $T_{\mathbf{X}}$ yields a (regular) node $T'_{\mathbf{X}}$ with $\binom{|\mathcal{D}_{\mathbf{X}}|}{k}$ children $\{T_{\mathbf{x} \to \mathbf{x}'} | \mathbf{x}' \subseteq_k D_{\mathbf{X}}\}$, where $T_{\mathbf{x} \to \mathbf{x}'}$ is the result of replacing $\mathbf{x}$ with objects $\mathbf{x}'$ in $T_{\mathbf{x}}$.

Example 4. Figure 3 (a) shows the dtree of Example 3 and its corresponding FO-dtree, which only has one instance T_x of all isomorphic subtrees T_{x_i}. Figure 3 (b) shows the FO-dtree for Example 1.

[3] As such, it plays the same role as a logvar. However, we use both to distinguish between a whole group of randvars (a PRV $P(X)$), and a representative of this group (a representative randvar $P(x)$).

4.2 Properties

Darwiche [2] showed that important properties of a recursive decomposition are captured in the properties of dtree nodes. In this section, we define these properties for FO-dtrees. Adapting the definitions of the dtree properties, such as cutset, context, and cluster, for FO-dtrees requires accounting for the semantics of an FO-dtree, which uses DPG nodes and representative objects. More specifically, this requires making the following two modifications (i) use a function $Child_\theta(T)$, instead of $Child(T)$, to take into account the semantics of DPG nodes, and (ii) use a function $\cap_\theta$ that finds the intersection of two sets of *representative* randvars. First, for a DPG node $T_\mathbf{X} = (\mathbf{X}, \mathbf{x}, C)$, we define: $Child_\theta(T_X) = \{T_{\mathbf{x}\to\mathbf{x}'} | \mathbf{x}' \subseteq_k \mathcal{D}_\mathbf{X}\}$. Second, for two sets $A = \{a_i\}_{i=1}^n$ and $B = \{b_i\}_{i=1}^n$ of (representative) randvars we define: $A \cap_\theta B = \{a_i | \exists\theta \in \Theta : a_i\theta \in B\}$, with Θ the set of grounding substitutions to their representative objects. Naturally, this provides a basis to define a '$\setminus_\theta$' operator as : $A \setminus_\theta B = A \setminus (A \cap_\theta B)$.

All the properties of an FO-dtree are defined based on their corresponding definitions for dtrees, by replacing $Child$, $\cap$, $\setminus$ with $Child_\theta$, $\cap_\theta$, $\setminus_\theta$. Interestingly, all the properties can be computed without grounding the model, e.g., for a DPG node T_X, we can compute $rv(T_X)$ simply as $rv(T_x)\theta_X^{-1}$, with $\theta_X^{-1} = \{\mathbf{x} \to \mathbf{X}\}$.[4] Figure 4 shows examples of FO-dtrees with their node clusters.

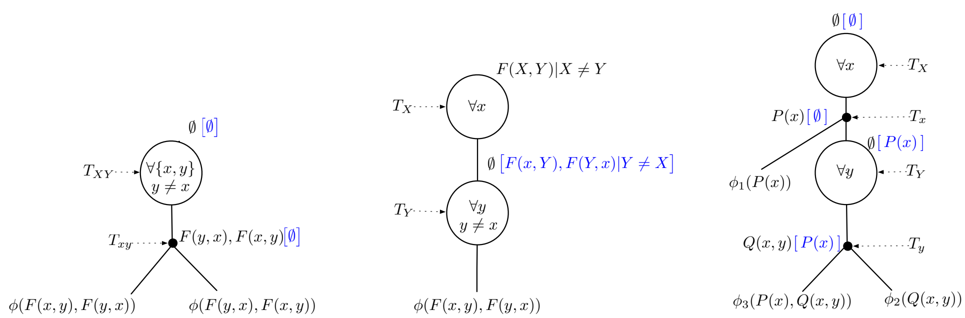

Figure 4: Three FO-dtree with their clusters (shown as $cutset$, [$context$]).

Counted FO-dtrees. FO-dtrees capture the first lifting tool, isomorphic decomposition, explicitly in DPG nodes. The second tool, counting, can be simply captured by rewriting interchangeable randvars in clusters of the tree nodes with counting randvars. This can be done in FO-dtrees similarly to the operation of counting conversion on logvars in LVE. We call such a tree a *counted* FO-dtree. Figure 5(a) shows an FO-dtree (left) and its counted version (right).

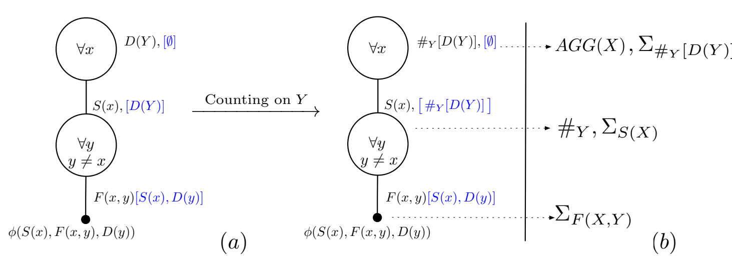

Figure 5: (a) an FO-dtree (left) and its counted version (right); (b) lifted operations of each node.

5 Liftable FO-dtrees

When inference can be performed using the lifted operations (i.e., without grounding the model), it runs in polynomial time in the domain size of logvars. Formally, this is called a domain-lifted inference solution [19]. Not all FO-dtrees have a lifted solution, which is easy to see since not

[4]The only non-trivial property is $cutset$ of DPG nodes. We can show that $cutset(T_X)$ excludes from $rv(T_X) \setminus acutset(T_X)$ only those PRVs for which $\mathbf{X}$ is a root binding class of logvars [8, 19].

all models are liftable [7], though each model has at least one FO-dtree.[5] Fortunately, we can structurally identify the FO-dtrees for which we know a lifted solution.

What models can the lifting tools handle? Lifted inference identifies isomorphic problems and solves only one instance of those. Similar to propositional inference, for a lifted method the difficulty of each sub-problem increases with the number of variables in the problem– those that appear in the *clusters* of FO-dtree nodes. When each problem has a bounded (domain-independent) number of those, the complexity of inference is clearly independent of the domain size. However, a sub-problem can involve a large group of randvars— when there is a PRV in the cluster. While traditional inference is then intractable, lifting may be able to exploit the interchangeability among the randvars and reduce the complexity by *counting*. Thus, whether a problem has a lifted solution boils down to whether we can rewrite it such that it only contains a bounded (domain-independent) number of counting randvars and ground randvars. This requires the problem to have enough symmetries in it such that all the randvars $\mathcal{V} = V_1, \ldots V_n$ in each cluster can be divided into k groups of interchangeable (tuples of) randvars $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k$, where k is independent of the domain size.

Theorem 1 *A (non-counted) FO-dtree has a lifted inference solution if its clusters only consist of (representative) randvars and* 1*-logvar PRVs. We call such an FO-dtree a* liftable *tree.*[6]

Proof sketch. Such a tree has a corresponding LVE solution: (i) each sub-problem that we need to solve in such a tree can be formulated as a (sum-out) problem on a model consisting of a parfactor with 1-logvar PRVs, and (ii) we can count-convert all the logvars in a parfactor with 1-logvar PRVs [10, 16], to rewrite all the PRVs into a (bounded) number of counting randvars.[7]

6 Lifted inference based on FO-dtrees

A dtree can prescribe the operations performed by propositional inference, such as VE [2]. In this section, we show how a liftable FO-dtree can prescribe an LVE solution for the model, thus providing the first formal method for symbolic operation selection in lifted inference.

In VE, each inference procedure can be characterized based on its elimination order. Darwiche [2] shows how we can read a (partial) elimination order from a dtree (by assigning elimination of each randvar to some tree node). We build on this result to read an LVE solution from a (non-counted) FO-dtree. For this, we assign to each node a set of lifted operations, including lifted elimination of PRVs (using multiplication and sum-out), and counting conversion and aggregation of logvars:

- $\sum_{\mathcal{V}}$: A PRV $\mathcal{V}$ is *eliminated* at a node T, if $\mathcal{V} \in cluster(T) \setminus context(T)$.
- $AGG(X)$: A logvar X is *aggregated* at a DPG node $T_{\mathbf{X}} = (\mathbf{X}, \mathbf{x}, C)$, if (i) $X \in \mathbf{X}$, and (ii) $X \notin logvar(cluster(T_{\mathbf{X}}))$.
- $\#_X$: A logvar X is *counted* at $T_{\mathbf{X}}$, if (i) $X \in \mathbf{X}$, and (ii) $X \in logvar(cluster(T_X))$.

A lifted solution can be characterized by a sequence of these operations. For this we simply need to order the operations according to two rules:

1. If node T_2 is a descendent of T_1, and OP_i is performed at T_i, then $OP_2 \prec OP_1$.
2. For operations at the same node, aggregation and counting precede elimination.

Example 5. From the FO-dtree shown in Figure 5 (a) we can read the following order of operations: $\sum F(X,Y) \prec \#_Y \prec \sum S(X) \prec AGG(X) \prec \sum \#_Y[D(Y)]$, see Figure 5 (b). $\square$

7 Complexity of lifted inference

In this section, we show how to compute the complexity of lifted inference based on an FO-dtree. Just as the complexity of ground inference for a dtree is parametrized in terms of the tree's *width*, we define a *lifted width* for FO-dtrees and use it to parametrize the complexity of lifted inference.

[5]A basic algorithm for constructing an FO-dtree for a PLM is presented in the supplementary material.

[6]Note that this only restricts the number of logvars in PRVs appearing in an *FO-dtree's clusters*, not PRVs in the PLM. For instance, all the liftable trees in this paper correspond to PLMs containing 2-logvar PRVs.

[7]For a more detailed proof, see the supplementary material.

To analyze the complexity, it suffices to compute the complexity of the operations performed at each node. Similar to standard inference, this depends on the randvars involved in the node's *cluster*: for each lifted operation at a node T, LVE manipulates a factor involving the randvars in $cluster(T)$, and thus has complexity proportional to $O(|\mathit{range}(\mathit{cluster}(T))|)$, where *range* denotes the set of possible (joint) values that the randvars can take on. However, unlike in standard inference, this complexity need not be exponential in $|rv(\mathit{cluster}(T))|$, since the clusters can contain *counting randvars* that allow us to handle interchangeable randvars more efficiently. To accommodate this in our analysis, we define two *widths* for a cluster: a *ground width* w_g, which is the number of ground randvars in the cluster, and a *counting width*, $w_\#$, which is the number of counting randvars in it. The cornerstone of our analysis is that the complexity of an operation performed at node T is exponential only in w_g, and polynomial in the domain size with degree $w_\#$. We can thus compute the complexity of the entire inference process, by considering the hardest of these operations, and the number of operations performed. We do so by defining a *lifted width* for the tree.

Definition 4 (Lifted width) *The* lifted width *of an FO-dtree T is a pair $(w_g, w_\#)$, where w_g is the largest ground width among the clusters of T and and $w_\#$ is the largest counting width among them.*

Theorem 2 *The complexity of lifted variable elimination for a counted liftable FO-dtree T is:*

$$O(n_T \cdot \log n \cdot \exp(w_g) \cdot n_\#^{(w_\# \cdot r_\#)}),$$

where n_T is the number of nodes in T, $(w_g, w_\#)$ is its lifted width, n (resp., $n_\#$) is the the largest domain size among its logvars (resp., counted logvars), and $r_\#$ is the largest range size among its tuples of counted randvars.

Proof sketch. We can prove the theorem by showing that (i) the largest range size among clusters, and thus the largest factor constructed by LVE, is $O(\exp(w_g) \cdot n^{(w_\# \cdot r_\#)})$, (ii) in case of aggregation or counting conversion, each entry of the factor is exponentiated, with complexity $O(\log n)$, and (iii) there are at most n_T operations. (For a more detailed proof, see the supplementary material.) $\square$

Comparison to ground inference. To understand the savings achieved by lifting, it is useful to compare the above complexity to that of standard VE on the corresponding dtree, i.e., using the same decomposition. The complexity of ground VE is: $O(n_G \cdot \exp(w_g) \cdot \exp(n_\#.w_\#))$, where n_G is the size of the corresponding propositional dtree. Two important observations are:

1. The number of ground operations is linear in the dtree's size n_G, instead of the FO-dtree's size n_T (which is polynomially smaller than n_G due to DPG nodes). Roughly speaking, lifting allows us to perform n_T/n_G of the ground operations by *isomorphic decomposition*.
2. Ground VE, has a factor $\exp(n_\#.w_\#)$ in its complexity, instead of $n_\#^{w_\#}$ for lifted inference. The latter is typically exponentially smaller. These speedups, achieved by *counting*, are the most significant for lifted inference, and what allows it to tackle high treewidth models.

8 Conclusion

We proposed FO-dtrees, a tool for representing a recursive decomposition of PLMs. An FO-dtree explicitly shows the symmetry between its isomorphic parts, and can thus show a form of decomposition that lifted inference methods employ. We showed how to decide whether an FO-dtree is liftable (has a corresponding lifted solution), and how to derive the sequence of lifted operations and the complexity of LVE based on such a tree. While we focused on LVE, our analysis is also applicable to lifted search-based methods, such as lifted recursive conditioning [13], weighted first-order model counting [21], and probabilistic theorem proving [6]. This allows us to derive an order of operations and complexity results for these methods, when operating based on an FO-dtree. Further, we can show the close connection between LVE and search-based methods, by analyzing their performance based on the same FO-dtree. FO-dtrees are also useful to approximate lifted inference algorithms, such as lifted blocked Gibbs sampling [22] and RCR [20], that attempt to improve their inference accuracy by identifying liftable subproblems and handling them by exact inference.

Acknowledgements

This research is supported by the Research Fund K.U.Leuven (GOA 08/008, CREA/11/015 and OT/11/051), and FWO-Vlaanderen (G.0356.12).

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Solving #-SAT and Bayesian inference with backtracking search. *Journal of Artificial Intelligence Research*, 34(2):391, 2009.

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

[3] 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.

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

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

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

[7] 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)*, 2012.

[8] 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.

[9] 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.

[10] 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.

[11] Mathias Niepert. Markov chains on orbits of permutation groups. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 624–633, 2012.

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

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

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

[15] 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.

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

[17] 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.

[18] Nima Taghipour, Daan Fierens, Guy Van den Broeck, Jesse Davis, and Hendrik Blockeel. Completeness results for lifted variable elimination. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

[19] 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.

[20] Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 131–141, 2012.

[21] 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.

[22] Deepak Venugopal and Vibhav Gogate. On lifting the gibbs sampling algorithm. In *Proceedings of the 26th Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 1–6, 2012.