Faster Lifting for Two-Variable Logic Using Cell Graphs

Timothy van Bremen[1] **Ondřej Kuželka**[2]

[1]KU Leuven, Belgium
[2]Czech Technical University in Prague, Czech Republic

Abstract

We consider the *weighted first-order model counting* (WFOMC) task, a problem with important applications to inference and learning in structured graphical models. Bringing together earlier work [Van den Broeck et al., 2011, 2014], a formal proof was given by Beame et al. [2015] showing that the two-variable fragment of first-order logic, $\mathbf{FO}^2$, is *domain-liftable*, meaning it admits an algorithm for WFOMC whose runtime is polynomial in the given domain size. However, applying this theoretical upper bound is often impractical for real-world problem instances. We show how to adapt their proof into a fast algorithm for lifted inference in $\mathbf{FO}^2$, using only off-the-shelf tools for knowledge compilation, and several careful optimizations involving the *cell graph* of the input sentence, a novel construct we define that encodes the interactions between the cells of the sentence. Experimental results show that, despite our approach being largely orthogonal to that of FORCLIFT [Van den Broeck et al., 2011], our algorithm often outperforms it, scaling to larger domain sizes on more complex input sentences.

1 INTRODUCTION

Given a sentence ϕ in first-order logic and a domain size $n \in \mathbb{N}$, the *first-order model counting* (FOMC) problem asks for the number of models of ϕ over the domain $\{1, \ldots, n\}$. Closely related to FOMC is its weighted variant, *weighted first-order model counting* (WFOMC), in which each predicate R_i appearing in ϕ is associated with positive and negative weights $(w_{R_i}, \bar{w}_{R_i})$, and the task becomes to compute the weighted sum of models of ϕ over $\{1, \ldots, n\}$.

Efficient algorithms for WFOMC have applications in areas as diverse as probabilistic inference and weight learning in Markov logic networks [Van den Broeck et al., 2014, Van Haaren et al., 2016], and enumeration problems in combinatorics. In the context of many applications—especially those in probabilistic inference—it is often desirable to compute the weighted first-order model count in a manner whose runtime grows efficiently with n, the input domain size.

Unfortunately, it has been shown that (assuming $\mathsf{E} \neq \mathsf{NE}$) an algorithm that is efficient in terms of domain size for *any* first-order sentence does not exist, via a reduction to spectrum membership [Jaeger, 2015]. Therefore, over the past several years, an effort has been made to map out fragments of first-order logic that do allow for efficient inference in the size of the input domain: so-called *domain-liftable* fragments. In a seminal result, Van den Broeck et al. [2014] showed that the syntactic fragment of first-order logic limited to two variables, $\mathbf{FO}^2$, is domain-liftable. This result was very recently extended to $\mathbf{C}^2$, the two-variable fragment with counting quantifiers, allowing many interesting properties on graphs to be modelled and efficiently counted [Kuzelka, 2021].

However, in spite of the theoretical domain-liftability of two-variable logic, computing the WFOMC for sentences in this fragment is still quite challenging in practice. As we show later in the paper, FORCLIFT [Van den Broeck et al., 2011] struggles to scale given larger input sentences that are common in modelling problems with a richer structure, and other lifted inference systems such as ALCHEMY2 [Gogate and Domingos, 2011] and GC-FOVE [Taghipour et al., 2013] focus on Markov logic networks and parfactors respectively as input models. While translations from WFOMC to inference in these models do exist, the structure in the original problem is typically lost, limiting their utility for highly structured combinatorial problems. In this paper, we seek to close this gap by proposing a fast and easy-to-implement algorithm for WFOMC in $\mathbf{FO}^2$. The algorithm only requires access to an off-the-shelf d-DNNF compiler, and most importantly scales to large domains even on complex input sentences.

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

Note that, despite the relatively limited expressivity of two-variable logic as an input language, practical algorithms for this fragment have important consequences for efficient lifting of more expressive input languages: in particular, Kuzelka [2021] showed that WFOMC in the far more expressive fragment $\mathbf{C}^2$ can be reduced to a series of calls to an $\mathbf{FO}^2$ WFOMC oracle, thus underlining the importance of an efficient method for two-variable logic like the one proposed here. Indeed, later in the paper we show how our algorithm can be applied in this way to quickly count k-regular graphs on n nodes, a property expressible in $\mathbf{C}^2$ but not $\mathbf{FO}^2$.

2 RELATED WORK

Poole [2003] proposed using an algorithm for reasoning about populations in the setting of Bayesian networks, forming the foundation for research in lifted inference. Various extensions to this technique were put forth over the years (see, for example, [de Salvo Braz et al., 2005, Taghipour et al., 2013, Braun and Möller, 2016], among many others), but in most cases these works focused on reasoning over first-order counterparts of graphical models directly.

More recently, the same ideas have been applied in the framework of first-order logic [Gogate and Domingos, 2011, Van den Broeck et al., 2011], with the latter paper first introducing the formal notion of weighted first-order model counting, and proposing a practical approach, FORCLIFT, for performing lifted inference in this setting. This tool was improved and eventually extended to operate on the full fragment of two-variable logic [Van den Broeck et al., 2014]. On the other hand, Beame et al. [2015] examined the WFOMC problem from a theoretical perspective and showed various lower and upper bounds for the problem, including $\#\mathsf{P}_1$-completeness of the three-variable fragment $\mathbf{FO}^3$, and gave a consolidated and formal proof of the tractability of $\mathbf{FO}^2$ which we build upon later in this paper. Since then, the "frontier of tractability" has been further mapped out [Kazemi et al., 2016, Kuusisto and Lutz, 2018, Kuzelka, 2021].

One limitation in the existing literature is that presentations of implementations in previous work, where they exist, have usually dealt with relatively simple models in the fragment they lift: for example, Van den Broeck et al. [2011] evaluate FORCLIFT over examples such as the classic "friends and smokers" and "friends-smokers-drinkers", which each contain only a few predicates and clauses. Thus, the focus so far has primarily been on *proving* domain-liftability of new fragments, rather than studying how these algorithms scale in practice. In this paper, we hope to shift this focus, and examine how these ideas can be applied to more complex models while still scaling quickly.

To this end, very recent work by van Bremen and Kuzelka [2020] investigated a practical *approximate* approach to the WFOMC problem, geared towards dealing with complex non-liftable models. However, as their approach is not strictly domain-lifted and requires grounding out the input sentence, we do not consider it further here.

3 BACKGROUND

In this section we provide some background on (weighted) model counting in propositional and first-order logic, and review the domain-liftability result for $\mathbf{FO}^2$.

3.1 FIRST-ORDER LOGIC

We deal with the function-free, finite domain fragment of first-order logic. An *atom* of arity k takes the form $P(t_1, \ldots, t_k)$, where P/k comes from a vocabulary of *predicates*, and each argument t_i is either a constant from a finite domain $\mathbf{D}$, or a logical variable from a vocabulary of variables. A *literal* is an atom or its negation. A *formula* is formed by connecting one or more literals together using conjunction or disjunction. A formula may optionally be surrounded by one or more quantifiers of the form $\exists x$ or $\forall x$, where x is a logical variable. A logical variable in a formula is said to be *free* if it is not bound by any quantifier. A formula with no free variables is called a *sentence*. We follow the usual semantics of first-order logic.

Definition 1. *The first-order model count (FOMC) of a sentence ϕ over a domain of size n is defined as:*

$$\mathsf{FOMC}(\phi, n) = |\mathsf{models}_n(\phi)|$$

where $\mathsf{models}_n(\phi)$ denotes the set of all models of ϕ over the domain $\mathbf{D} = \{1, \ldots, n\}$.

We define the *weighted* first-order model count of a sentence in terms of weightings.

Definition 2. *Denote the set of predicates appearing in a sentence ϕ by P_ϕ. A weighting on ϕ is a pair of mappings $w : P_\phi \to \mathbb{R}$ and $\bar{w} : P_\phi \to \mathbb{R}$.*

Definition 3. *Let $(w, \bar{w})$ be a weighting on a sentence ϕ. The weighted first-order model count (WFOMC) of ϕ over a domain of size n under $(w, \bar{w})$ is:*

$$\mathsf{WFOMC}(\phi, n, w, \bar{w}) = \sum_{\mu \in \mathsf{models}_n(\phi)} \prod_{L \in \mu_T} w(\mathsf{pred}(L)) \cdot \prod_{L \in \mu_F} \bar{w}(\mathsf{pred}(L))$$

where μ_T denotes the set of true ground atoms in the model μ, and μ_F the false ground atoms. The notation $\mathsf{pred}(L)$ maps an atom L to its corresponding predicate name.

In general, the weighted first-order model count of any sentence ϕ can always be computed by grounding the sentence

out over the input domain to a propositional formula ϕ_G, and computing the *propositional* weighted model count $\mathsf{WMC}(\phi_G, w_G, \bar{w}_G)$, extending the original weight functions to the literals in ϕ_G in the natural way. However, such an approach is usually intractable for all but the smallest domains, motivating the search for more tractable algorithms as described in the following section.

Finally, to motivate Definition 3 we point out that inference in Markov logic networks [Richardson and Domingos, 2006] can be reduced to WFOMC using the reduction described in [Van den Broeck et al., 2011]. A similar reduction for probabilistic logic programs was also shown in [Van den Broeck et al., 2014], but this reduction is constrained to work only on a rather limited class of "tight" logic programs. More recent results have also shown how to apply WFOMC algorithms to weight learning in Markov logic networks as well [Van Haaren et al., 2016, Kuzelka and Kungurtsev, 2019, Kuzelka et al., 2020]. In this paper for the sake of simplicity (and ease of checking correctness) we instead focus on combinatorial applications[1]; however, all of our results can also easily be applied to probabilistic inference and learning.

3.2 DOMAIN-LIFTABILITY OF $\mathbf{FO}^2$

Given a fragment $\mathcal{F}$ of first-order logic, we may consider its *data complexity* for WFOMC: fixing the input sentence as some $\phi \in \mathcal{F}$ and weights $(w, \bar{w})$, what is the complexity of computing $\mathsf{WFOMC}(\phi, n, w, \bar{w})$ when treating the domain size n as the input? In Appendix C of [Beame et al., 2015], the authors consolidate work from earlier papers to prove that the two-variable fragment $\mathbf{FO}^2$ enjoys FP data complexity, or in other words is *domain-liftable*. The proof for this result is important, as we will build on it—and certain concepts therein, such as that of *cells*—throughout the remainder of the paper. We therefore repeat the high-level ideas of their argument here[2], and refer the reader to their paper for the details.

Definition 4. *A cell of a first-order formula ψ is a maximally consistent set of literals formed from atoms in ψ using only a single variable x.*

Cells are also referred to as *1-types* in the logic literature (see, for example, [Libkin, 2004]), but we use the term "cells" for the sake of consistency with [Beame et al., 2015].

Example 1. *Consider the formula $\psi(x,y) = (F(x,y) \vee G(x)) \wedge (\neg G(x) \vee \neg H(x))$. Then there are $2^3 = 8$ possible cells on $\psi(x,y)$; one such example is $\neg F(x,x) \wedge G(x) \wedge H(x)$.*

[1] As Kuusisto and Lutz [2018] point out, WFOMC provides a *logic-based way of classifying combinatorial problems.*

[2] To improve clarity, we diverge slightly from the presentation of Beame et al. [2015] by including reflexive binary atoms in cells.

Theorem 1 ([Beame et al., 2015])**.** *The fragment of first-order logic limited to two variables, $\mathbf{FO}^2$, is domain-liftable.*

Proof sketch. Suppose we wish to compute $\mathsf{WFOMC}(\phi, n, w, \bar{w})$ for some input sentence $\phi \in \mathbf{FO}^2$, domain size $n \in \mathbb{N}$, and weights $(w, \bar{w})$. Begin by applying the reduction in [Grädel et al., 1997] and eliminating existential quantifiers as shown in [Van den Broeck et al., 2014] to get a universally quantified sentence $\phi = \forall x \forall y \psi(x,y)$ such that all atoms in ψ have arity at most 2.

Suppose ψ has u distinct predicates appearing in it. Take the 2^u cells on ψ and denote them $C_1, \ldots, C_{2^u}$. Now, consider the possible partitions of $[n]$ into 2^u disjoint sets. Each of these partitions can be thought of as representing a series of assignments of subsets of the n domain elements to each of the cells. Then the models of ϕ over the domain $[n]$ correspond to the models of the following sentence:

$$\eta = \bigwedge_{i,j \in [2^u], i<j} \forall x : S_i \forall y : S_j \ (\psi(x,y) \wedge \psi(y,x)) \wedge \bigwedge_{k \in [2^u]} \forall x : S_k \forall y : S_k \ \psi(x,y)$$

where S_i denotes the elements of $[n]$ assigned to the cell C_i, and the notation $\forall x : S_i$ denotes universal quantification limited to the set S_i. Since we know the truth values of the unary and reflexive binary atoms given by each cell assignment C_i, we may simplify the body of each conjunct by replacing every unary and reflexive binary atom with true or false as appropriate. Write $\psi_i(x,y)$ for the simplified version of $\psi(x,y)$ when both x and y belong to the same cell C_i, and $\psi_{ij}(x,y)$ for the simplified version of $\psi(x,y) \wedge \psi(y,x)$ when x and y belong to C_i and C_j respectively. We then have:

$$\eta = \bigwedge_{i,j \in [2^u], i<j} \forall x : S_i \forall y : S_j \ (\psi_{ij}(x,y)) \wedge \bigwedge_{k \in [2^u]} \forall x : S_k \forall y : S_k \ \psi_k(x,y)$$

Observe that each conjunct in the formula above is independent (that is, they do not share any propositional variables when grounded out). Denote $r_{ij} = \mathsf{WMC}(\psi_{ij}(a,b), w, \bar{w})$, $s_k = \mathsf{WMC}(\psi_k(a,b) \wedge \psi_k(b,a), w, \bar{w})$, and $w_k = \mathsf{WMC}(\psi_k(c,c), w, \bar{w})$. Summing across the different possible configurations of cell cardinalities, and multiplying by a multinomial coefficient to account for the different possible selections of domain elements for a given configuration, we get:

$$\mathsf{WFOMC}(\phi, n, w, \bar{w}) = \sum_{n_1 + \cdots + n_{2^u} = n} \binom{n}{n_1, \ldots, n_{2^u}} \cdot \prod_{i,j \in [2^u], i<j} r_{ij}^{n_i n_j} \prod_{i \in [2^u]} s_i^{n_i(n_i-1)/2} w_i^{n_i} \quad (1)$$

Clearly, evaluating this equation can be done in time polynomial in the domain size, and so we have that $\mathbf{FO}^2$ is domain-liftable. □

4 ALGORITHM

We are now ready to present our algorithm for computing the WFOMC of a given two-variable sentence. Although it is tempting to apply the approach presented in the proof of Theorem 1 directly, there are several practical barriers to doing so; namely:

1. We must make a propositional weighted model counter call for each r_{ij}, s_i, and w_i term. We can avoid this using *knowledge compilation*.
2. The number of possible cells grows exponentially in the number of predicates in the input sentence (2^m for m predicates). We can reduce the number of cells we need to consider using *model enumeration*.
3. In turn, the number of cell pairs included in each of the sum terms of Equation (1) grows quadratically in the number of cells: for each cell j, we must consider the terms r_{ij} for all $i < j$ (as well as the s_i and w_i terms). We address this problem by making our algorithm *weight-aware* using *independent sets*.
4. The number of terms in the outermost sum of Equation (1) grows in the number of partitions of the domain size n, and there is no mechanism in place to *reuse* results computed within any of the terms. We address these problems through the use of *dynamic programming*.
5. In many cases, we find that certain groups of cells are identical in how they interact with other cells (the r_{ij} terms) and themselves (the s_i and w_i terms), but there is no procedure taking advantage of this. We solve this by showing how to identify and exploit *cell symmetries*.

The latter three optimizations all rely on a construct, which we call the *cell graph*, that can be computed from the input sentence.

Definition 5. *The cell graph G_ϕ of a sentence ϕ is a complete graph (V, E) where:*

1. *V is the set of cell labels ($\{1, \ldots, 2^u\}$) on ϕ*
2. *Each node $i \in V$ has a self-loop labelled with the tuple (s_i, w_i)*
3. *Each edge from node i to j ($i < j$) is labelled with r_{ij}*

Although this construct seems simple, we will see later that it is helpful to think of the optimizations presented here as being performed over this cell graph. Each of the improvements are explained individually below.

4.1 KNOWLEDGE COMPILATION

The first, and simplest, improvement is to make use of knowledge compilation to make the repeated calls to a propositional weighted model counter faster. Observe that each of the r_{ij} and s_i terms corresponds to a different conditioning of the unary and reflexive binary ground atoms of the propositional formula $\psi(a, b) \wedge \psi(b, a)$. Thus, after preprocessing the input sentence, we can employ the d-DNNF compiler DSHARP [Muise et al., 2012] to construct a smooth d-DNNF from which we may efficiently condition on the different cells to extract the values of the r_{ij} and s_i terms.

4.2 MODEL ENUMERATION

Another important observation is that for most sentences, there will be several cells that are impossible. We therefore do not need to consider any cell cardinality configuration that assigns a non-zero value to any of these cells.

Example 2. *Consider the formula $\psi(x, y) = (F(x, y) \vee G(x)) \wedge (\neg G(x) \vee \neg H(x))$. Here, it is clear that after conditioning on any cell setting both $G(x)$ and $H(x)$ to true, the residual formula will always be unsatisfiable.*

We will call cells that are guaranteed to lead to a satisfiable residual formula *valid* cells. To easily obtain the set of valid cells of a formula, we can enumerate[3] the models of the d-DNNF corresponding to $\psi(c, c)$ and obtain a set M of valid cells. We may then consider only those $|M|$ cells for the remainder of the algorithm.

We can also update our cell graph by simply deleting any node (along with any edges connecting to it) that corresponds to a cell that is not valid. From now on, we will assume that our cell graph has been defined accordingly.

4.3 WEIGHT-AWARENESS

In the previous section, we considered cells that lead to unsatisfiability of the entire formula when we condition on them. Here, we consider a similar idea, but instead we find a group of cells C such that when conditioning the formula by setting the cell for x as $c \in C$, any further conditioning on y with another cell in C determines a weighted model count of 1. Concretely, this means that $s_c = 1$, and moreover for any other cell $d \in C$, we have that $r_{cd} = 1$ (or $r_{dc} = 1$) as well. We can find such cells by computing an independent set of its *independent cell graph*.

Definition 6. *The independent cell graph $I(G_\phi)$ is a graph formed from the cell graph G_ϕ by:*

[3] Note that d-DNNFs support fast (polytime) model enumeration, just like they do for model counting [Darwiche and Marquis, 2002].

1. *Deleting edges (i, j) such that $r_{ij} = 1$*
2. *Deleting any self-loop (i, i) such that $s_i = 1$*

This yields the following result, which can be easily obtained through some algebraic manipulations on Equation (1) in combination with the properties identified in Definition 6 (a full proof is given in the supplementary material).

Theorem 2. *Let ϕ be a sentence, and denote some maximal independent set of $I(G_\phi)$ by I_1. Now, let $k = |I_1|$, and suppose without loss of generality we reorder the $|M|$ cells such that the first k cells are those of the independent set. Then*[4]*:*

$$\mathsf{WFOMC}(\phi, n, w, \bar{w}) = \sum_{n_{k+1}+\cdots+n_{|M|}\leq n} \binom{n}{n_{k+1}, \ldots, n_{|M|}} \cdot \prod_{i,j:i,j\notin\{1,2,\ldots,k\},i<j} r_{ij}^{n_i n_j} \prod_{i\notin\{1,2,\ldots,k\}} w_i^{n_i} s_i^{n_i(n_i-1)/2} \cdot \left(\sum_{i=1}^{k} w_i \prod_{j\notin\{1,2,\ldots,k\}} r_{i,j}^{n_j}\right)^{n-n_{k+1}-\cdots-n_{|M|}} \quad (2)$$

Applying this yields a speedup on the order of n^{k-1}.

4.4 DYNAMIC PROGRAMMING

Unfortunately, for many cells, we may have that $s_i \neq 1$. In this case, we can identify a second set of cells whose interactions with cells in I_1 is limited.

Definition 7. *The reduced independent cell graph of $I(G_\phi)$ with respect to an independent set I_1 of $I(G_\phi)$, denoted by $R(I_1, I(G_\phi))$, is formed from $I(G_\phi)$ by:*

1. *Removing all vertices present in I_1 and their neighbours*
2. *Removing all self-loops*

From $R(I_1, I(G_\phi))$ we can again compute a maximal independent set I_2. Note that, by construction, the cells in I_2 have limited interaction with one another (i.e. we have $r_{ij} = 1$ for $i, j \in I_2$), and also with those in I_1 ($r_{ij} = 1$ for $i \in I_1$, $j \in I_2$). Suppose we again reorder the cells, so that those in I_1 are precisely the cells $\{1, \ldots, k\}$, and those in I_2 the cells $\{k + 1, \ldots, k + l\}$. Now, define inductively:

$$g_0(n_{k+l+1}, \ldots, n_{|M|}, N) = \left(\sum_{i=1}^{k} w_i \prod_{j\notin\{1,2,\ldots,k+l\}} r_{i,j}^{n_j}\right)^{n-N-n_{k+l+1}-\cdots-n_{|M|}} \quad (3)$$

and

$$g_p(n_{k+l+1}, \ldots, n_{|M|}, N) = \sum_{n_{k+p}=0}^{n-N-n_{k+l+1}-\cdots-n_{|M|}} \binom{n-N-n_{k+l+1}-\cdots-n_{|M|}}{n_{k+p}} \cdot w_{k+p}^{n_{k+p}} s_{k+p}^{n_{k+p}(n_{k+p}-1)/2} \cdot \left(\prod_{j\notin\{1,2,\ldots,k+l\}} r_{(k+p),j}^{n_{k+p}n_j}\right) \cdot g_{p-1}(n_{k+l+1}, \ldots, n_{|M|}, N + n_{k+1}) \quad (4)$$

for $0 < p \leq l$. Using these definitions along with the idea applied in Theorem 2, we have the following result.

Theorem 3. *Let ϕ be a sentence, and I_1 and I_2 be independent sets of G_ϕ and $R(I_1, G_\phi)$ respectively. Suppose, without loss of generality, that we reorder the cells in ϕ such that $\{1, \ldots, k\}$ are precisely the cells in I_1 and $\{k+1, \ldots, k+l\}$ the cells in I_2. Then:*

$$\mathsf{WFOMC}(\phi, n, w, \bar{w}) = \sum_{n_{k+l+1}+\cdots+n_{|M|}\leq n} \binom{n}{n_{k+l+1}, \ldots, n_{|M|}} \cdot \prod_{i,j:i,j\notin\{1,2,\ldots,k+l\},i<j} r_{ij}^{n_i n_j} \prod_{i\notin\{1,2,\ldots,k+l\}} w_i^{n_i} s_i^{n_i(n_i-1)/2} \cdot g_l(n_{k+l+1}, \ldots, n_{|M|}, 0)$$

Here, the individual g_p functions can be memoized, allowing for a dynamic programming algorithm that saves computation when we have multiple cells in I_2.

4.5 CELL SYMMETRIES

We can do even better than the improvements outlined above. The final observation that we make is that one often finds groups of cells that not only interact with each other in the same way, but also interact with cells outside of the group in the same way too. We will call such groups *symmetric cliques*[5].

Definition 8. *Let ϕ be a sentence, and let C denote the set of (valid) cells on ϕ. Then a subset $B \subseteq C$ of cells is called a symmetric clique if it satisfies the following conditions:*

1. *$s_i = s_j$ for all $i, j \in B$, $i \neq j$*
2. *$w_i = w_j$ for all $i, j \in B$, $i \neq j$*
3. *$r_{ik} = r_{jk}$ for all $i, j \in B$, $k \in C \setminus B$, $i \neq j$, $i, j < k$*
4. *$r_{ij} = r_{kl}$ for all $i.j, k, l \in B$, $i < j$, $k < l$*

[4] We abuse notation slightly here and throughout the paper by using $\binom{N}{k_1,\ldots,k_n}$ to denote $\binom{N}{k_1,\ldots,k_n,N-\sum_i k_i}$ when $\sum_i k_i < N$.

[5] Strictly speaking, any group of nodes in the cell graph forms a clique in the graph-theoretic sense, since the cell graph is complete. However, we still use this name in a slightly different sense as we feel it best conveys the way they behave.

In order to efficiently partition the set of cells into a family of symmetric cliques, we can use a simple greedy algorithm like the one shown in Algorithm 1. We start a clique with some cell, and continue trying to add cells that satisfy the symmetric clique property w.r.t. the cells in the current clique (line 5). Once we have checked all cells, we start a new clique and continue until all of the cells have been assigned to a clique.

Algorithm 1 An algorithm for finding symmetric cliques

Input: A set of cells C of a sentence ϕ
Output: A family $\mathcal{F}_C$ of symmetric cliques on C

1: $\mathcal{F}_C \leftarrow \emptyset$
2: **while** $C \neq \emptyset$ **do**
3: $\quad cur \leftarrow \emptyset$
4: $\quad$ **for** cell $c \in C$ **do**
5: $\quad\quad$ **if** Compatible(c, cur) **then**
6: $\quad\quad\quad cur \leftarrow cur \cup \{c\}$
7: $\quad\quad\quad C \leftarrow C \setminus \{c\}$
8: $\quad \mathcal{F}_C \leftarrow \mathcal{F}_C \cup \{cur\}$
9: **return** $\mathcal{F}_C$

After identifying the symmetric cliques of the input sentence, we can view each of these cliques as forming a new "pseudo-node" in the cell graph, replacing all of the cells in the clique.

Definition 9. *Let ϕ be a sentence with (valid) cells C. Denote by $\mathcal{F}_C$ the family of symmetric cliques on C. Then the collapsed cell graph of ϕ, denoted by $Col(G_\phi, \mathcal{F}_C)$ is the cell graph formed by deleting all nodes from G_ϕ, except for one cell per clique in $\mathcal{F}_C$, and relabelling the cells in some consistent order $\{1, \ldots, |\mathcal{F}_C|)\}$.*

Using these symmetric cliques, we can still evaluate the WFOMC using the techniques from Sections 4.3 and 4.4. We will now view I_1 and I_2 as being computed over the *collapsed* (reduced) (independent) cell graph, and the equations given in those sections as being evaluated over the collapsed cell graph formed by the symmetric cliques rather than the cells of the input sentence directly. This means that the symmetric cliques will act like new cells, inheriting the corresponding r_{ij}, s_i, and w_i values of their constituents. The only difference is that when encountering a collapsed clique node, we must take care to account for the s_i-terms for all of the cells in the clique, as well as the *internal* r_{ij} terms for cells $i < j$ inside the clique. We can do this by defining an auxiliary term that deals with this. Suppose that c is a symmetric clique comprising the cells $\{1, \ldots, k\}$. We can define:

$$J_c(\widehat{n}) = \sum_{n_1+\cdots+n_k=\widehat{n}} \binom{\widehat{n}}{n_1, \ldots, n_k} s_1^{\binom{n_1}{2}+\cdots+\binom{n_k}{2}} \cdot \prod_{i,j:1\leq i,j\leq k, i<j} r_{ij}^{n_i n_j}$$

Intuitively, $\widehat{n}$ represents the number of elements assigned to the symmetric clique c, and the J-function above encodes the different ways the possible assignments to its constituent cells affect the internal s_i and r_{ij} terms.

Using this, we may then rewrite Equation (4) as:

$$g_p(n_{k+l+1}, \ldots, n_{|M|}, N) = \sum_{n_{k+p}=0}^{n-N-n_{k+l+1}-\cdots-n_{|M|}} \binom{n-N-n_{k+l+1}-\cdots-n_{|M|}}{n_{k+p}} \cdot w_{k+p}^{n_{k+p}} J_{k+p}(n_{k+p}) \cdot \left(\prod_{j\notin\{1,2,\ldots,k+l\}} r_{(k+p),j}^{n_{k+p}n_j}\right) \cdot g_{p-1}(n_{k+l+1}, \ldots, n_{|M|}, N + n_{k+1})$$

Note that Equation (3) does not change.

Going back to Theorem 3, we can rewrite the expression for the WFOMC to be defined over the symmetric cliques instead:

$$\mathsf{WFOMC}(\phi, n, w, \bar{w}) = \sum_{n_{k+l+1}+\cdots+n_{|\mathcal{F}_C|}\leq n} \binom{n}{n_{k+l+1}, \ldots, n_{|\mathcal{F}_C|}} \cdot \prod_{i,j:i,j\notin\{1,2,\ldots,k+l\}, i<j} r_{ij}^{n_i n_j} \prod_{i\notin\{1,2,\ldots,k+l\}} w_i^{n_i} J_i(n_i) \cdot g_l(n_{k+l+1}, \ldots, n_{|\mathcal{F}_C|}, 0) \quad (5)$$

For the equation above to hold, we must ensure that only J-functions evaluating to 1 are allowed into the expression for g_0. In other words, I_1 must not contain any symmetric clique i such that $J_i(\widehat{n}) \neq 1$ for some $\widehat{n}$. To do this, we may adapt the construction of the collapsed cell independent graph by removing the self-loop from the clique node for c only when we have $J_c(\widehat{n}) = 1$ for all $\widehat{n} \in \{1, \ldots, n\}$.

The final point we address is how to evaluate the J-functions efficiently. It turns out dynamic programming can be used again here. Given some symmetric clique $c = \{1, \ldots, k\}$, we may define inductively:

$$d_{k,c}(\widehat{n}) = \left(\frac{s}{r}\right)^{\widehat{n}(\widehat{n}-1)/2}$$

and

$$d_{i,c}(\widehat{n}) = \sum_{n_i=0}^{\widehat{n}} \binom{\widehat{n}}{n_i} \left(\frac{s}{r}\right)^{n_i(n_i-1)/2} d_{i+1,c}(\widehat{n}-n_i)$$

for $1 \leq i < k$, where s is the value s_k for some arbitrary node k in the clique, and r the value r_{ij} for some two nodes $i < j$ in the clique.[6] One can verify then that:

$$J_c(\widehat{n}) = r^{\widehat{n}(\widehat{n}-1)/2} d_{1,c}(\widehat{n}) \quad (6)$$

[6] We implicitly assume here that the clique comprises at least two nodes; otherwise evaluation of the J-function is trivial.

Just as in Section 4.4, we can again employ dynamic programming to memoize the values of the individual d-function calls. This means that we can evaluate any J-function over all $\widehat{n} \in \{1, \ldots, n\}$ at the same time using a number of arithmetic operations which is quadratic in the domain size n.

4.6 FINAL ALGORITHM

The pseudocode for our algorithm incorporating all of the improvements outlined above, which we dub FASTWFOMC, is summarized in Algorithm 2. Lines 2 to 4 preprocess the input sentence to obtain a Skolemized formula containing only unary and binary predicates, and compile the requisite d-DNNFs. The set of valid cells is computed on line 5, and then the r_{ij}, s_i, and w_i terms are calculated by conditioning on the d-DNNFs for each of the cells (lines 6–11). On the basis of these values the cell set gets partitioned into a family of symmetric cliques (line 12), and the remainder of the code essentially implements Equation (5). The functions GetGTerm and GetJTerm are computed using Equations (4) and (6) respectively, but their complete pseudocode is given in the supplementary material to this paper.

Algorithm 2 FASTWFOMC

Input: $\mathbf{FO}^2$ sentence ϕ, weights $(w, \bar{w})$, domain size n
Output: $\mathsf{WFOMC}(\phi, n, w, \bar{w})$

1: **/* Initialization */**:
2: $\psi \leftarrow \text{Preprocess}(\phi)$
3: $\text{d-DNNF}_1 \leftarrow \text{Compile}(\psi(a, b) \wedge \psi(b, a))$
4: $\text{d-DNNF}_2 \leftarrow \text{Compile}(\psi(c, c))$
5: $C \leftarrow \text{Models}(\text{d-DNNF}_2)$
6: **for** $i, j \in \{1, \ldots, |\mathcal{F}_C|\}$ **do**
7: $\quad r_{ij} \leftarrow \text{WeightedModels}(\text{d-DNNF}_1(i, j))$
8: **for** $i \in \{1, \ldots, |\mathcal{F}_C|\}$ **do**
9: $\quad s_i \leftarrow \text{WeightedModels}(\text{d-DNNF}_1(i, i))$
10: **for** $i \in \{1, \ldots, |\mathcal{F}_C|\}$ **do**
11: $\quad w_i \leftarrow \text{WeightedModels}(\text{d-DNNF}_2(i))$
12: $\mathcal{F}_C \leftarrow \text{PartitionIntoCliques}(C)$
13: $sum \leftarrow 0$
14: **/* Main loop */**:
15: **for** $\{n_{k+l+1}, \ldots, n_{|\mathcal{F}_C|}\}$ s.t. $\sum_i n_i \leq n$ **do**
16: $\quad term \leftarrow \binom{n}{n_{k+l+1}, \ldots, n_{|\mathcal{F}_C|}}$
17: $\quad$ **for** $i, j \in \{k + l + 1, \ldots, |\mathcal{F}_C|\}$ **do**
18: $\quad\quad term \leftarrow term \cdot r_{ij}^{n_i n_j}$
19: $\quad$ **for** $i \in \{k + l + 1, \ldots, |\mathcal{F}_C|\}$ **do**
20: $\quad\quad term \leftarrow term \cdot w_i^{n_i} \cdot \text{GetJTerm}(i, n_i)$
21: $\quad term \leftarrow term \cdot \text{GetGTerm}(l, 0, n_{k+l+1}, \ldots, n_{|\mathcal{F}_C|})$
22: $\quad sum \leftarrow sum + term$
23: **return** sum

5 EXPERIMENTS

We implemented FASTWFOMC in Python[7] and used the DSHARP [Muise et al., 2012] compiler to construct the d-DNNFs. All experiments were performed on a computer with a six-core Intel i7 2.2GHz processor and 16 GB of RAM.

We compared FASTWFOMC to FORCLIFT [Van den Broeck et al., 2011], which to our knowledge is the only other tool capable of directly computing the WFOMC of a first-order sentence using lifted inference.

5.1 BENCHMARKS

We tested our algorithm on the benchmarks below. We focused on problems from enumerative combinatorics since their solutions are easily checked against *The On-line Encyclopedia of Integer Sequences* (OEIS)[8], but one could easily adapt these into, for example, Markov logic networks using the usual encoding described in [Van den Broeck et al., 2014]:

- `3-regular`: counting 3-regular graphs on n nodes. A graph is said to be k-regular if each node has precisely k neighbours.
- `4-coloured`: counting 4-coloured graphs on n nodes. A k-colourable graph is a graph whose nodes are coloured with one of k colours, such that no two nodes of the same colour are adjacent to one another. A k-coloured graph is a k-colourable graph along with a valid colouring function.
- `derangements`: counting derangements on n items. A derangment is a permutation that maps no item to itself. This example models the following classic combinatorial word problem: n people go to the theatre and leave their hats in the cloakroom. Something goes wrong and the hats get mixed up. What is the probability that no person goes home with the same hat they came with?
- `3-matchings`: counting the number of ways of constructing three non-overlapping maximal matchings on K_{2n}, the complete graph with $2n$ vertices.

For the three benchmarks above whose sentences are in $\mathbf{C}^2$ but not expressible in $\mathbf{FO}^2$ (all but `4-coloured`), we employed the encoding of Kuzelka [2021] which splits the WFOMC task into several oracle calls of a two-variable sentence containing no counting quantifiers, with varying weights. We measure the time taken for one such call. In addition, for all of the benchmarks above, we used the standard technique described by Van den Broeck et al. [2014]

[7] Source code for FASTWFOMC is available online at `https://people.cs.kuleuven.be/~timothy.vanbremen/`.
[8] `https://oeis.org/`

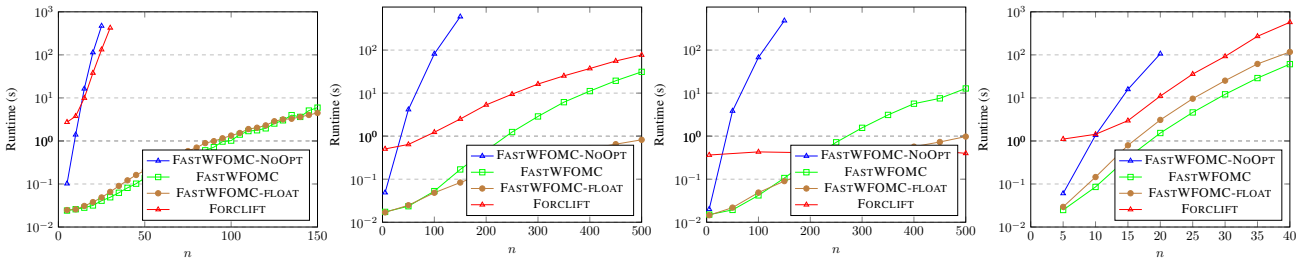

Figure 1: (a) A comparison of the runtime of FastWFOMC and Forclift on `3-regular` for various domain sizes. (b) The same, for `4-coloured`. (c) The same, for `derangements`. (d) The same, for `3-matchings`.

to Skolemize the input sentence where appropriate. These transformations can result in a sentence passed to FastWFOMC and Forclift with more predicates and clauses than their original encoding. Both the original and (where applicable) transformed sentences are given in the supplementary material.

Finally, as Forclift does not support negative weights, when weights of -1 were present due to Skolemization, we simply adjusted these to 1 for Forclift for comparison purposes (note that this should have no impact on performance).

5.2 RESULTS

We compared the performance on the four benchmarks in Figure 1. Note that Forclift does not support exact arithmetic and instead performs approximate calculations in log mode, whereas FastWFOMC by default uses arbitrary precision integers, which is important for combinatorial applications like the ones here. Since solutions at this scale often have tens of thousands of digits, performing this arithmetic can become a dominating factor for FastWFOMC as the domain size grows large. Therefore, for illustrative purposes, we also graphed the performance of a floating point version of FastWFOMC. Unlike Forclift, we employ arbitrary-precision (rather than native) floats through the MPFR library [Fousse et al., 2007], meaning FastWFOMC still suffers a small performance handicap here. We also point out that unlike Forclift (Scala), FastWFOMC is implemented in an interpreted language (Python), which again accounts for a small performance difference. Lastly, as a second baseline in order to better measure the impact of several of the improvements explained in the paper, we also considered an "unoptimized" version of FastWFOMC which sets $I_1 = I_2 = \emptyset$ in all cases and does not exploit symmetric cliques.

In Figure 1a, we compare the performance of FastWFOMC with Forclift on the `3-regular` problem for varying values of the domain size n. Forclift takes over 400 seconds on a domain size of $n = 30$, whereas FastWFOMC is far more resilient, taking a fraction of a second for the same domain size, and scaling comfortably in under 6 seconds to 150 nodes. FastWFOMC is thus the clear winner here, especially considering that in this application, due to counting quantifiers several such oracle calls would actually be necessary to count 3-regular graphs.

In Figure 1b, we repeat the exercise for `4-coloured`, and find that FastWFOMC also performs better here, with the *exact* version running over twice as fast as Forclift's floating point implementation even at 500 nodes (31.2 vs 76.9 seconds). On the floating point version of FastWFOMC, which is a more direct comparison, the improvement is far more pronounced (< 1 vs 76.9 seconds). FastWFOMC therefore again seems to be the clear winner on this benchmark.

Next, in Figure 1c, we compare the algorithms on `derangements`. Here Forclift is able to solve the problem in near-constant time regardless of the domain size (we suspect its runtime is linear, though), whereas the runtime of FastWFOMC seems to grow polynomially with the domain size. We conclude that Forclift performs better on this benchmark for large domain sizes. We suspect that this might be due to the presence of the domain recursion rule used in Forclift, for which no analogue exists in our algorithm. In future work, we would like to investigate this more closely.

In Figure 1d, we compare the algorithms on `3-matchings`. The difficulty of this problem grows rapidly with the domain size, but FastWFOMC seems to outperform Forclift by an increasing margin as the domain size grows: on a domain size of 20 (corresponding to K_{40}), FastWFOMC takes 3.1 seconds while Forclift takes 11 seconds, for a roughly 3-fold speedup. When the domain size is increased to 40, this improvement grows to a factor of about 5 (116.9 vs 571.2 seconds). As a result, FastWFOMC once again comes out ahead here.

Finally, we observe that in virtually all cases, our unoptimized implementation of FastWFOMC was outperformed by both Forclift and our original implementation of Fast-

WFOMC, showing the importance of the improvements presented in the paper.

6 CONCLUSION

We presented FASTWFOMC, an algorithm for computing the weighted first-order model count of a two-variable sentence in a domain-lifted manner, and showed its improvement over FORCLIFT. There are several avenues for future work, including support for closer integration with more expressive fragments such as $\mathbf{S}^2\mathbf{FO}^2$ [Kazemi et al., 2016] and $\mathbf{C}^2$ [Kuzelka, 2021], as well as alternative heuristics for optimally choosing the independent sets I_1 and I_2.

Acknowledgements

TvB was supported by the Research Foundation – Flanders (G095917N). OK was supported by Czech Science Foundation project "Generative Relational Models" (20-19104Y) and by the OP VVV project *CZ.02.1.01/0.0/0.0/16_019/0000765* "Research Center for Informatics".

References

Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. Symmetric weighted first-order model counting. In *PODS*, pages 313–328. ACM, 2015.

Tanya Braun and Ralf Möller. Lifted junction tree algorithm. In *KI*, volume 9904 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2016.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *IJCAI*, pages 1319–1325. Professional Book Center, 2005.

Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2):13, 2007.

Vibhav Gogate and Pedro M. Domingos. Probabilistic theorem proving. In *UAI*, pages 256–265. AUAI Press, 2011.

Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bull. Symb. Log.*, 3(1):53–69, 1997.

Manfred Jaeger. Lower complexity bounds for lifted inference. *Theory Pract. Log. Program.*, 15(2):246–263, 2015.

Seyed Mehran Kazemi, Angelika Kimmig, Guy Van den Broeck, and David Poole. New liftable classes for first-order probabilistic inference. In *NIPS*, pages 3117–3125, 2016.

Antti Kuusisto and Carsten Lutz. Weighted model counting beyond two-variable logic. In *LICS*, pages 619–628. ACM, 2018.

Ondrej Kuzelka. Weighted first-order model counting in the two-variable fragment with counting quantifiers. *J. Artif. Intell. Res.*, 70:1281–1307, 2021.

Ondrej Kuzelka and Vyacheslav Kungurtsev. Lifted weight learning of markov logic networks revisited. In *AISTATS*, volume 89 of *Proceedings of Machine Learning Research*, pages 1753–1761. PMLR, 2019.

Ondrej Kuzelka, Vyacheslav Kungurtsev, and Yuyi Wang. Lifted weight learning of markov logic networks (revisited one more time). In *PGM*, volume 138 of *Proceedings of Machine Learning Research*, pages 269–280. PMLR, 2020.

Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-DNNF compilation with sharpSAT. In *Canadian Conference on AI*, volume 7310 of *Lecture Notes in Computer Science*, pages 356–361. Springer, 2012.

David Poole. First-order probabilistic inference. In *IJCAI*, pages 985–991. Morgan Kaufmann, 2003.

Matthew Richardson and Pedro M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. Lifted variable elimination: Decoupling the operators from the constraint language. *J. Artif. Intell. Res.*, 47:393–439, 2013.

Timothy van Bremen and Ondrej Kuzelka. Approximate weighted first-order model counting: Exploiting fast approximate model counters and symmetry. In *IJCAI*, pages 4252–4258. ijcai.org, 2020.

Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*, pages 2178–2185. IJCAI/AAAI, 2011.

Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *KR*. AAAI Press, 2014.

Jan Van Haaren, Guy Van den Broeck, Wannes Meert, and Jesse Davis. Lifted generative learning of markov logic networks. *Mach. Learn.*, 103(1):27–55, 2016.