Neural Networks for Approximate DNF Counting: An Abridged Report[*]

Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz
Department of Computer Science
University of Oxford, UK
{ralph.abboud,ismail.ceylan,thomas.lukasiewicz}@cs.ox.ac.uk

Abstract

Weighted model counting (WMC) has emerged as a prevalent approach for probabilistic inference. In its most general form, WMC is #P-hard. Weighted DNF counting (weighted #DNF) is a special case where approximations with probabilistic guarantees are obtained in $O(nm)$, where n denotes the number of variables, and m the number of clauses of the input DNF, but this is not scalable in practice. In this paper, we propose a novel approach for weighted #DNF that combines approximate model counting with deep learning, and accurately approximates model counts in linear time when width is bounded. We conduct experiments to validate our method, and show that our model learns and generalizes very well to large-scale #DNF instances.

1 Introduction

Propositional *model counting (MC)*, or *#SAT*, is the task of counting the number of satisfying assignments for a given propositional formula (Gomes, Sabharwal, and Selman 2009). *Weighted model counting (WMC)*, or *weighted #SAT,* additionally incorporates a *weight function* over the set of all possible assignments. Offering an elegant formalism for encoding various probabilistic inference problems, WMC is a unifying approach for inference in a wide a range of probabilistic models. In particular, *probabilistic graphical models* (Koller and Friedman 2009), *probabilistic planning* (Domshlak and Hoffmann 2007), *probabilistic logic programming* (De Raedt, Kimmig, and Toivonen 2007), *probabilistic databases* (Suciu et al. 2011), and *probabilistic ontologies* (Borgwardt, Ceylan, and Lukasiewicz 2017) can greatly benefit from advances in WMC.

Two important special cases of WMC are *weighted #CNF* and *weighted #DNF*, which require the input formula to be in conjunctive normal form (CNF) and disjunctive normal form (DNF), respectively. Inference in probabilistic graphical models typically reduces to solving weighted #CNF instances, while query evaluation in probabilistic databases reduces to solving weighted #DNF instances (Suciu et al. 2011). However, both weighted #CNF and weighted #DNF are known to be #P-hard (Valiant 1979), and this computational complexity is a major bottleneck for solving large-scale WMC instances. To overcome this bottleneck, approaches have been developed based on *knowledge compilation (KC)* (Cadoli and Donini 1997; Selman and Kautz 1996) and *approximate solving* (Ermon et al. 2013; Chakraborty, Meel, and Vardi 2016; Meel, Shrotri, and Vardi 2017). KC tools map WMC instances into a representation where they can solved efficiently, but such representations could be exponentially sized in the worst case. Approximate solvers return a model count estimate with probabilistic guarantees, but they typically have impractical runtime complexity. Hence, both approaches struggle to scale in practice.

In this paper, we focus on weighted #DNF, for which, a relatively efficient approximation algorithm exists (Karp, Luby, and Madras 1989). This algorithm, called KLM, runs in $O(nm)$, where n denotes the number of variables and m the number of clauses of the input DNF formula. Our contribution is to develop a neural-symbolic model, which performs weighted #DNF approximation via a novel approach that is based on graph neural networks (GNNs). By construction, our model produces approximations in $O(m\bar{w})$, where $\bar{w}$ denotes the average clause width. Our model does not provide guarantees, but enables a speed-up of multiple orders of magnitude in the average case relative to other tools. This is especially true since, in practice, $\bar{w} << n$.

Our GNN learns to accurately estimate weighted model counts and generalizes to novel formulas. Indeed, it computes solutions to unseen weighted #DNF instances with 99% accuracy relative to an additive error threshold of 0.1 with respect to tight KLM approximations. It also generalizes to larger problem instances involving up to 15K variables remarkably well, despite only seeing formulas with at most 5K variables during training. These findings suggest that GNNs can effectively and efficiently perform large-scale #DNF through training with dense and reliable data.

2 Weighted Model Counting

Given a (finite) set S of propositional variables, a *literal* is of the form v, or $\neg v$, where $v \in S$. A *conjunctive clause* is

[*]This is a short version of a paper that will appear in the Proceedings of AAAI-20 (Abboud, Ceylan, and Lukasiewicz 2020), and is also available at: arxiv.org/pdf/1904.02688.pdf.

a conjunction of literals, and a *disjunctive clause* is a disjunction of literals. A clause has *width* k if it has exactly k literals. A formula ϕ is in *conjunctive normal form (CNF)* if it is a conjunction of disjunctive clauses, and it is in *disjunctive normal form (DNF)* if it is a disjunction of conjunctive clauses. We say that a DNF (resp., CNF) has width k if it contains clauses of width at most k. An assignment $\nu : S \mapsto \{0, 1\}$ maps every variable to either 0 (false), or 1 (true). An assignment ν *satisfies* a propositional formula ϕ, denoted $\nu \vDash \phi$, in the usual sense, where $\vDash$ is the propositional entailment relation.

Given a propositional formula ϕ, its *model count (MC)* $\#\phi$ is the number of assignments ν satisfying ϕ. The *weighted model count (WMC)* of ϕ is given by $\sum_{\nu \vDash \phi} w(\nu)$, where $w : \mathfrak{A} \mapsto \mathbb{R}$ is a *weight function*, and $\mathfrak{A}$ is the set of all possible assignments. In this work, we set $w : \mathfrak{A} \mapsto [0, 1] \cap \mathbb{Q}$ such that every assignment is mapped to a rational probability and $\sum_{\nu \in \mathfrak{A}} w(\nu) = 1$. As common in the literature, we view every propositional variable as an independent Bernoulli random variable and assign probabilities to literals.

Exactly solving weighted #DNF instances is #P-hard and thus intractable. The KLM algorithm (Karp, Luby, and Madras 1989) is a fully polynomial randomized approximation scheme (FPRAS), and provides probabilistic guarantees for weighted #DNF. More formally, given an error $\epsilon > 0$ and a confidence value $0 < \delta < 1$, KLM computes $\hat{\mu}$, an approximation of the true weighted model count μ, in polynomial time such that $\Pr\Big(\mu(1-\epsilon) \leq \hat{\mu} \leq \mu(1+\epsilon)\Big) \geq 1 - \delta$.

3 Graph Neural Network Model

Graph neural networks (GNNs) (Gori, Monfardini, and Scarselli 2005; Scarselli et al. 2009) are neural networks specifically designed to process structured graph data. In a GNN, every graph node x is given a vector representation v_x, which is updated iteratively. A node x receives information from its *neighborhood* $N(x)$, which is the set of nodes connected by an edge to x. Let $v_{x,t}$ denote the value of v_x at iteration t. We write a node update as:

$$v_{x,t+1} = combine\Big(v_{x,t}, aggregate\big(N(x)\big)\Big),$$

where *combine* and *aggregate* are functions, and *aggregate* is permutation-invariant. Upon termination of all iterations, the final node representations are used to compute the target output. GNNs are highly expressive computational models: GNNs can be as discerning between graphs as the Weisfeiler-Lehman (WL) graph isomorphism heuristic (Xu et al. 2019; Morris et al. 2019), and can autonomously learn relationships between nodes, identify important features and generalize to unseen graphs.

We propose a new method for solving weighted #DNF problems based on GNNs. We model DNF formulas as graphs, and then build a GNN architecture to iterate over these graphs to compute an approximate weighted model count.

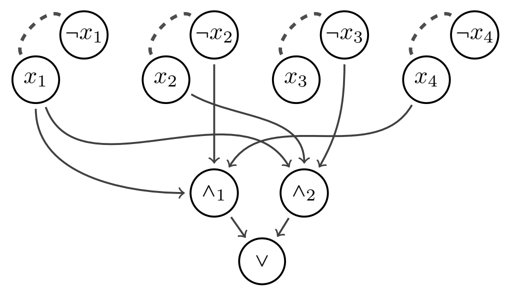

Figure 1: Graph encoding of the DNF formula $\phi = (x_1 \wedge \neg x_2 \wedge x_4) \vee (x_1 \wedge x_2 \wedge \neg x_3)$.

Model Setup

We encode a DNF formula as a graph with 3 layers as shown in Figure 1: a *literal* layer, a *conjunction* layer, and a *disjunction* layer. In the *literal* layer, every DNF variable is represented by 2 nodes corresponding to its positive and negative literals, which are connected by a (dashed) edge to highlight that they are complementary. In the *conjunction* layer, every node represents a conjunction and is connected to *literal* nodes whose literals appear in the conjunction. Finally, the *disjunction* layer contains a single disjunction node, which is connected to all nodes in the *conjunction* layer.

To approximate the model count of a DNF formula, we use a message-passing GNN model that iterates over the corresponding DNF graph and returns a Gaussian distribution. We use layer-norm LSTMs (Li et al. 2016; Ba, Kiros, and Hinton 2016) as our *combine* function, and sum as our *aggregate* function. Initially, the network computes vector representations for all literal nodes, given their probabilities, using a multi-layer perceptron (MLP) f_{enc}. More formally, a k−dimensional representation $v_{x_i,0}$ of a literal x_i with probability p_i is computed as $v_{x_i,0} = f_{enc}(p_i)$. Nodes in the *conjunction* and *disjunction* layers are initialized to two representation vectors v_c and v_d, respectively, and the values for these vectors are learned over the course of training. After initialization, node representations are updated across T message passing iterations.

Message Passing Protocol

A message passing iteration consists of four steps:
(a) *Literal* layer nodes compute messages using an MLP M_l and pass them to their neighboring *conjunction* layer nodes. These *conjunction* nodes then aggregate these messages and update their representation using a layer-norm LSTM L_{c_1}. The updated *conjunction* node representations, denoted $\hat{v}_{x_c,t+1}$, are given formally as

$$\hat{v}_{x_c,t+1} = L_{c_1}\Big(v_{x_c,t}, \sum_{x_l \in N(x_l)} M_l(v_{x_l,t})\Big).$$

(b) *Conjunction* layer nodes send messages to the disjunction node via an MLP M_c. The disjunction node aggregates these and updates using a layer-norm LSTM L_d, i.e.,

$$v_{x_d,t+1} = L_d\Big(v_{x_d,t}, \sum_{x_c \in N(x_d)} M_c(\hat{v}_{x_c,t+1})\Big).$$

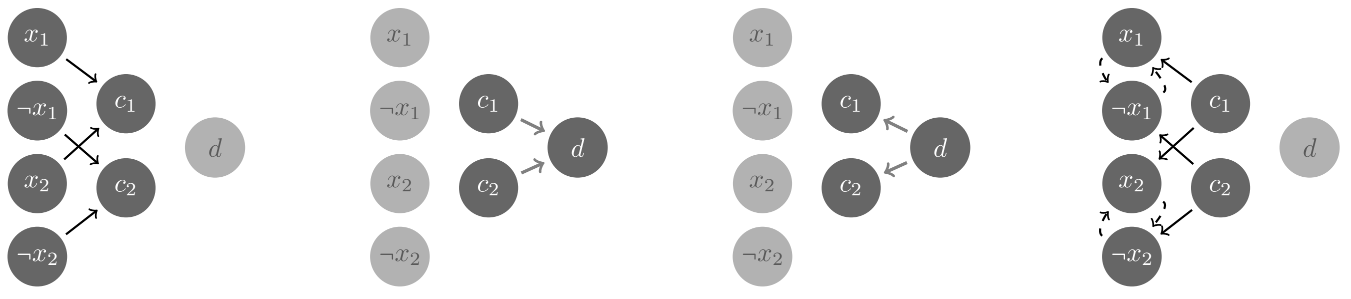

Figure 2: Message passing protocol on the DNF formula $\psi = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$.

(c) The disjunction node computes a message using an MLP M_d and sends it to the *conjunction* nodes, which update their representation using a different LSTM cell L_{c_2}:

$$v_{x_c,t+1} = L_{c_2}\Big(\hat{v}_{x_c,t+1}, M_d(v_{x_d,t+1})\Big).$$

(d) Using their latest representations, *conjunction* nodes send messages to neighboring nodes in the *literal* layer. *Literal* layer nodes aggregate these messages and concatenate them (represented with ||) with messages from their corresponding negated literal. Then, they use this message to update their representations using a layer-norm LSTM L_l:

$$v_{x_l,t+1} = L_l\Big(v_{x_l,t}, \big(\sum_{x_c \in N(x_l)} M_c(v_{x_c,t+1}) || M_l(v_{\neg x_l,t})\big)\Big).$$

A visual representation of the 4 message passing steps for a simple formula is provided in Figure 2. In this protocol, we use 2 distinct LSTM cells L_{c_1} and L_{c_2} to update the representations of *conjunction* nodes at steps (a) and (c), so that the network learns separate update procedures for literal-based and disjunction-based updates. At the end of message passing, the final disjunction node representation $v_{x_d,T}$ is passed through an MLP f_{out}. The final layer of this MLP consists of two neurons n_μ and n_σ, which return the mean and standard deviation, respectively, of a predicted Gaussian distribution.

Loss Function

Given ϵ and δ, KLM returns an estimate $\hat{\mu}$ of the true model count μ within a multiplicative bound with respect to ϵ, and this bound holds with probability $1-\delta$. The multiplicity of the bound interval on $\hat{\mu}$ w.r.t. ϵ makes it hard to fit standard distributions on it. Hence, we apply a natural logarithm to this bound to get the additive bound on $\log \mu$:

$$\log \hat{\mu} - \log(1+\epsilon) \leq \log \mu \leq \log \hat{\mu} + \log(1+\epsilon).$$

We then fit a Gaussian $\mathcal{N}(\mu', \sigma)$ to this bound by setting $\mu' = \hat{\mu}$ and $\sigma = {}^{\log(1+\epsilon)}/_{F^{-1}(1-\frac{\delta}{2})}$, where F^{-1} denotes the inverse cumulative distribution function of the standard Gaussian distribution. The GNN is thus trained to predict $\log \mu$, a negative number. We adapt the exponential linear unit (ELU) (Clevert, Unterthiner, and Hochreiter 2016) activation function and apply it to n_μ and n_σ. More specifically,

Table 1: Distribution of formula sizes in the training set.

Size (n)	50	100	250	500
Count	30000	20000	16000	12000
Size (n)	**750**	**1000**	**2500**	**5000**
Count	10000	8000	6000	3000

we use

$$ELU+1(x) = \begin{cases} e^{-x} & \text{if } x \leq 0 \\ x+1 & \text{otherwise,} \end{cases}$$

such that n_μ uses $-ELU+1(x)$, and n_σ uses $ELU+1(x)$, thereby restricting their outputs to be negative and positive, respectively.

To compare the predicted Gaussian and the KLM result, we use Kullback-Leibler (KL) divergence, which for two Gaussians $\mathcal{N}_1(\mu_1, \sigma_1)$ and $\mathcal{N}_2(\mu_2, \sigma_2)$ is given by:

$$KL(\mathcal{N}_1, \mathcal{N}_2) = \log \frac{\sigma_2}{\sigma_1} - \frac{1}{2} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}.$$

We set $\mathcal{N}_1$ to be the prediction returned by the network and $\mathcal{N}_2$ to be the KLM approximation. This choice is critical in order to avoid the system minimizing the training loss by learning to produce arbitrarily large values of σ_2.

4 Experiments

We train our model on a large set of DNF formulas and measure its generalization relative to new DNF formulas. These formulas are distinct in terms of *structure* (i.e., the underlying clauses and variables in every clause) and *size* (i.e, the number of clauses and variables is larger), so our experiments target generalization in both aspects. To evaluate *structure generalization*, we run our GNN on unseen formulas of comparable size to training formulas and measure its performance. To evaluate *size generalization*, we run tests on novel, larger formulas and assess how well the GNN performs.

Experimental Setup

In our experiments, we compare the predictions of the network $\hat{\mu}$ with those of KLM and check whether their abso-

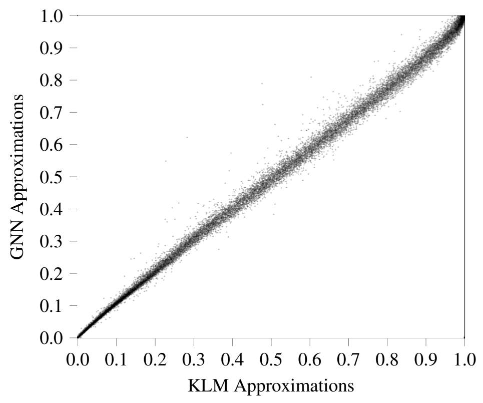

Figure 3: A gray-scale heat map representing the distribution of GNN predictions compared to KLM approximations.

Table 2: GNN accuracy (%) w.r.t. to additive thresholds.

Evaluation Data	**Thresholds**			
	0.02	0.05	0.10	0.15
Training Set	87.14	98.80	99.97	99.99
Test Set	87.37	98.76	99.95	99.98

lute difference falls within pre-defined additive thresholds. We opt for additive error, as opposed to multiplicative error, as the former produces an absolute distance metric, whereas the latter is relative to the model count.

Owing to the lack of standardized benchmarks, we generate synthetic formulas using a novel randomized procedure designed to produce more variable formulas. We generate 100K distinct training formulas, where formula counts per n are shown in Table 1. For every n, formulas are generated with fixed clause width $w \in \{3, 5, 8, 13, 21, 34\}$ and number of clauses m from $\{0.25, 0.375, 0.5, 0.625, 0.75\} \cdot n$, such that every valid setting is represented equally. More details about our data generation can be found in the conference version of this paper (Abboud, Ceylan, and Lukasiewicz 2020). The *structure* evaluation test set is generated analogously, and contains 13080 distinct formulas. The *size* evaluation set contains 348 formulas with $n = 10K$ and 116 formulas with $n = 15K$, with one probability distribution each. For all experiments, we use $k = 128$-dimensional vector representations and run the system for $T = 8$ message passing iterations. Further details on hyperparameter setup can be found in the conference version of this paper.

Results

On the *structure* generalization test, network predictions align very closely with those of KLM, as shown in Figure 3. The model is within 0.02 of the KLM WMC estimate over 87.37% of the test set, and this rises to 99.95% for a threshold of 0.1. The model also performs consistently across different n, with accuracy varying by at most 4.5% between any two different n values for all four test thresholds. Overall test results are given in Table 2.

Table 3: GNN accuracy (%) over test set by threshold versus number of formula variables (n).

n	**Thresholds**			
	0.02	0.05	0.10	0.15
50	85.58	98.58	99.98	100.0
100	87.87	98.87	100.0	100.0
250	87.93	99.24	100.0	100.0
500	87.67	99.40	99.99	100.0
750	87.56	99.15	100.0	100.0
1000	86.79	99.01	99.98	100.0
2500	90.06	98.17	99.85	99.94
5000	88.15	95.86	99.48	99.74

Table 4: Accuracy (%) by threshold with respect to additive thresholds on *size* generalization test formulas.

n	**Thresholds**			
	0.02	0.05	0.10	0.15
10K	79.89	89.94	97.13	99.71
15K	72.41	81.90	94.83	97.41

The proximity between training and testing accuracies at all thresholds shows that the network has not fit or memorized its training formulas, but has instead learned a general WMC procedure. The results parametrized by n are provided in Table 3. These results show that the network maintains a high accuracy (e.g., 95.5% for threshold 0.05) across all n values, and so does not rely on a particular n to achieve its high overall performance.

On the *size* generalization task, the model maintains accuracies of 97.13% and 94.83% with a threshold of 0.1 on 10K and 15K-variable formulas respectively, despite having as many as triple the variables as in training. The full results for *size* generalization are given in Table 4.

These results show that reliable approximate model counting on large-scale formulas can be achieved using neural message passing (NMP), even with training restricted to smaller formulas. Further analysis of *structure* and *size* generalization results relative to clause widths, as well as additional experiments with different datasets and message passing iteration counts T can be found in the conference version of this paper.

Running Time Analysis

In the average case, our GNN runs in $O(m\bar{w})$, where $\bar{w}$ denotes the average formula clause width. By contrast, KLM runs in $O(nm)$, so is much slower for standard cases in practice, where $\bar{w} << n$. In the worst case, our GNN runs in $O(nm)$, which is asymptotically identical to KLM. However, in a best-case scenario where $\bar{w}$ is upper-bounded, the GNN complexity drops to just $O(n + m)$, enabling linear-

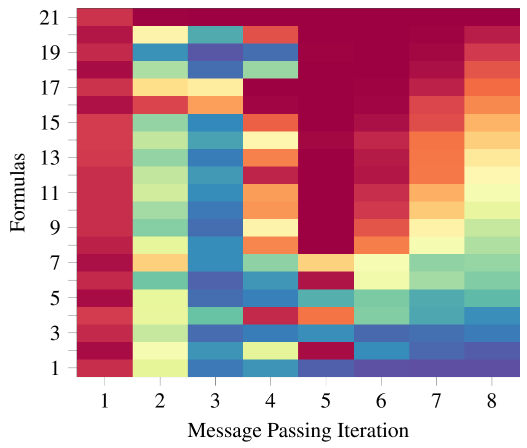

Figure 4: GNN estimates over message passing iterations. Red denotes small probability, blue denotes high probability.

time approximations to be made, whereas KLM remains $O(nm)$, since its complexity does not depend on $\bar{w}$. Hence, our system enables much faster approximations than KLM in practice, where $\bar{w} << n$, and these approximations are in linear time with bounded clause width $\bar{w}$.

Our GNN also does not perform slower at smaller widths w as with KLM, which relies on sampling. Random assignments in KLM are replaced when they satisfy a clause. Hence, more replacements are made for smaller w, as clause satisfaction is more likely, and this causes a heavy computational overhead. To illustrate, KLM needs 2375 seconds (about 40 minutes) to run on a formula with $n = 15K$, and $w = 3$ using $\epsilon = 0.1$ and $\delta = 0.05$, whereas it only requires 306s when $w = 34$. By contrast, the GNN requires only 0.104 and 0.223 seconds respectively. A detailed complexity analysis for our tool, as well as a complete presentation of KLM and GNN running times, can be found in the conference version of this paper.

Discussions: Analyzing the Model

To examine how the network makes predictions, we selected 21 formulas $f_i : i \in [1, 21]$ from the *structure* test set with weighted KLM model counts of roughly $\frac{21-i}{20}$. We then ran the network on these formulas and computed the predicted probability at the end of every message passing iteration. Results are visualized in Figure 4. Initially, the network starts with a low estimate. Then, in the first 3 iterations, it accumulates probabilities and hits a “spike”, which can be mapped to messages from literal nodes reaching the disjunction node. Following this, the network lowers its estimates, before adjusting and refining them in the final iterations.

Unlike (Selsam et al. 2019), where the estimate of satisfiability increases mostly monotonically, our network estimates fluctuate aggressively. A large initial estimate is made, and then reduced and refined. In doing so, the network seems to be initially estimating the naive sum of conjunction probabilities, and subsequently revisiting its estimates as it better captures intersections between conjunctions. This falls in line with our observations, as any understanding of intersections can only occur starting from the third iteration, when the disjunction and conjunction nodes will have passed each other more global information. This also explains the limited performance observed in our ablation study (cf. conference paper): With just 2 iterations, the system cannot capture conjunction intersections, so can only make naive estimates.

5 Related Work

Weighted #DNF belongs to the wider family of WMC problems, which have been extensively studied due to their connection with probabilistic inference. Weighted #DNF is #P-hard (Valiant 1979), so is highly intractable. Surprisingly, even weighted #DNF counting on positive, partitioned DNF formulas with clause width at most 2 (Provan and Ball 1983) remains #P-hard.

Weighted #DNF has applications that depend on fast online reasoning capabilities. For example, query evaluation in probabilistic database systems can be reduced to weighted #DNF through the so-called lineage representation of queries (Suciu et al. 2011). Other models that build on probabilistic databases can also directly benefit from such advances (Ceylan, Darwiche, and Van den Broeck 2016; Borgwardt, Ceylan, and Lukasiewicz 2017).

As a result, many methods have been developed to exactly solve or approximate WMC solutions. One such method is *knowledge compilation (KC)*, where WMC problems are compiled into a new representation in which they are solved efficiently and exactly. KC pushes computational overhead to a preprocessing phase, but compensates for this by subsequently enabling efficient, linear-time probabilistic inference (Darwiche and Marquis 2002). However, compiled representations can be of exponential size in the worst-case.

An alternative is to produce approximate solutions to circumvent the intractability of WMC (Stockmeyer 1983). *Hashing-based methods* (Ermon et al. 2013; Chakraborty, Meel, and Vardi 2013; 2016) produce an approximation with probabilistic guarantees. Importantly, (Chakraborty, Meel, and Vardi 2013) also yields an FPRAS when restricted to *unweighted* #DNF; see e.g. (Meel, Shrotri, and Vardi 2017). However, *none* of these hashing methods can currently be applied to weighted #DNF.

Our work builds on recent applications of GNNs (Scarselli et al. 2009) on a variety of reasoning tasks, such as SAT (Selsam et al. 2019) and the traveling salesman problem (TSP) (Prates et al. 2019). These works achieve encouraging results, but do not generalize beyond very small instances (i.e., 40 variables) of their respective problems. This is expected, since SAT and TSP are NP-complete and are hard to approximate with strong guarantees. By contrast, our work tackles a problem with a known polynomial-time approximation, learns from a dense dataset of approximate solutions, and reliably generalizes to large-scale instances with tolerable loss in performance. To our knowledge, our model is the first proposal that combines reasoning and deep learning, while also scaling to realistic problem instance sizes.

6 Summary and Outlook

We presented a neural-symbolic approach to efficiently produce weighted #DNF approximations. This work shows that neural networks can be effectively applied to large-scale weighted #DNF given dense and reliable training data.

Looking forward, we will analyze the viability of GNNs for other reasoning problems, particularly in light of their expressive power, which could be limiting for problems with less structured graph representations. We hope that this work inspires further research leading to less data-dependent neural-symbolic methods, and a greater understanding of neural method performance over challenging problems.

7 Acknowledgements

This work was supported by the Alan Turing Institute under the UK EPSRC grant EP/N510129/1, the AXA Research Fund, and by the EPSRC grants EP/R013667/1, EP/L012138/1, and EP/M025268/1. Ralph Abboud is funded by the Oxford-DeepMind Graduate Scholarship and the Alun Hughes Graduate Scholarship. Experiments for this work were conducted on servers provided by the Advanced Research Computing (ARC) cluster administered by the University of Oxford.

References

Abboud, R.; Ceylan, İ. İ.; and Lukasiewicz, T. 2020. Learning to Reason: Leveraging neural networks for approximate DNF counting. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Borgwardt, S.; Ceylan, İ. İ.; and Lukasiewicz, T. 2017. Ontology-mediated queries for probabilistic databases. In *Proc. of AAAI*.

Cadoli, M., and Donini, F. 1997. A survey on knowledge compilation. *AI Communications* 10(3-4).

Ceylan, İ. İ.; Darwiche, A.; and Van den Broeck, G. 2016. Open-world probabilistic databases. In *Proc. of KR*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A scalable approximate model counter. In *Proc. of CP*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2016. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*.

Clevert, D.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and accurate deep network learning by exponential linear units (ELUs). In *Proc. of ICLR*.

Darwiche, A., and Marquis, P. 2002. A Knowledge Compilation Map. *JAIR* 17(1).

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic prolog and its application in link discovery. In *Proc. of IJCAI*.

Domshlak, C., and Hoffmann, J. 2007. Probabilistic planning via heuristic forward search and weighted model counting. *JAIR* 30(1).

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2009. Model counting. In *Handbook of Satisfiability*. IOS Press.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *Proc. of IJCNN*.

Karp, R. M.; Luby, M.; and Madras, N. 1989. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms* 10(3).

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2016. Gated graph sequence neural networks. In *Proc. of ICLR*.

Meel, K. S.; Shrotri, A. A.; and Vardi, M. Y. 2017. On hashing-based approaches to approximate DNF-counting. In *Proc. of FSTTCS*.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proc. of AAAI*.

Prates, M. O. R.; Avelar, P. H. C.; Lemos, H.; Lamb, L.; and Vardi, M. 2019. Learning to solve NP-complete problems - A graph neural network for the decision TSP. In *Proc. of AAAI*.

Provan, J. S., and Ball, M. O. 1983. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM* 12(4).

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1).

Selman, B., and Kautz, H. 1996. Knowledge compilation and theory approximation. *JACM* 43(2).

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2019. Learning a SAT solver from single-bit supervision. In *Proc. of ICLR*.

Stockmeyer, L. 1983. The complexity of approximate counting. In *Proc. of STOC*. ACM.

Suciu, D.; Olteanu, D.; Ré, C.; and Koch, C. 2011. *Probabilistic Databases*, volume 3. Morgan & Claypool.

Valiant, L. G. 1979. The complexity of computing the permanent. *TCS* 8(2).

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *Proc. of ICLR*.