Generating Random Instances of Weighted Model Counting
An Empirical Analysis with Varying Primal Treewidth

Paulius Dilkas*

National University of Singapore, Singapore, Singapore
paulius.dilkas@nus.edu.sg

Abstract. Weighted model counting (WMC) is an extension of propositional model counting with applications to probabilistic inference and other areas of artificial intelligence. In recent experiments, WMC algorithms perform similarly overall but with significant differences on specific subsets of benchmarks. A good understanding of the differences in the performance of algorithms requires identifying key characteristics that favour some algorithms over others. In this paper, we introduce a random model for WMC instances with a parameter that influences primal treewidth—the parameter most commonly used to characterise the difficulty of an instance. We then use this model to experimentally compare the performance of WMC algorithms c2d, Cachet, d4, DPMC, and miniC2D. Using these random instances, we show that the easy-hard-easy pattern is different for algorithms based on dynamic programming and algebraic decision diagrams than for all other solvers. We also show how all WMC algorithms scale exponentially with respect to primal treewidth and how this scalability varies across algorithms and densities. Finally, we combine insights from experiments involving both random and competition instances to determine how the best-performing WMC algorithm varies depending on clause density and primal treewidth.

Keywords: Weighted model counting · Random model · Parameterised complexity

1 Introduction

Weighted model counting (WMC)—a weighted generalisation of propositional model counting (#SAT) [19]—has emerged as a powerful computational framework for problems in a variety of domains. In particular, WMC has been used to perform probabilistic inference for graphical models [8, 16, 17, 29, 71], probabilistic programs [55], and probabilistic logic programs [45]. More recently, WMC was used in the context of neural-symbolic artificial intelligence as well [75]. Extensions of WMC add support for continuous variables [11], infinite domains [10], and

* The work was done while the author was a PhD student at the University of Edinburgh.

first-order logic [51, 73] and generalise the definition to support arbitrary pseudo-Boolean functions instead of clauses [35]. Exact WMC algorithms can be broadly classified as based on search [69, 72], knowledge compilation [30, 58, 61], and dynamic programming [38, 39]. Other alternatives include approximate [15, 65] and parallel algorithms [24, 44], hybrid approaches [53], quantum computing [66], and reduction to model counting [14].

Recent papers that include experimental comparisons of WMC algorithms show many of them performing very similarly overall [38, 39] but with overwhelming differences when run on specific subsets of data [34, 35, 58]. Examples of such segregating data sets include bipartite Bayesian networks by Sang et al. [71] and relational Bayesian networks by Chavira et al. [20] that encode reachability in graphs under node deletion. So far, such performance differences remain unexplained. However, knowledge about the nature of these differences can inform our choices and aid in further algorithmic developments. Moreover, identifying performance predictors of algorithms is often an important step in developing a portfolio approach to the problem [76]. Lastly, if new algorithms are always tested on the same set of benchmarks, eventually they may become somewhat fitted to the particular characteristics of those instances, leading to algorithms that may perform worse when run on new types of data [56].

Both theoretical and experimental analysis of SAT algorithms on random instances is a rich area of research spanning almost forty years. Variations of some of the first random models ever proposed [46, 63] continue to be instrumental up to this day for, e.g., establishing the location of the threshold between satisfiable and unsatisfiable instances [2] and approximating #SAT [47]. Other random models consider non-uniform variable frequencies [3], fixing the number of times each variable occurs both positively and negatively [22], and adding other constraints such as cardinality and 'exclusive or' [62]. Experimental work investigating how SAT algorithms behave on random instances typically focuses on parameters that describe each instance independently of its size. The most common parameter is the ratio of clauses to variables, i.e., *(clause) density*. Early work in the area showed random 3-SAT instances to be at their hardest when density is around 4.25 [59]. Later work revealed that the interaction between density and empirical hardness is much more solver-dependent [21]. Many other parameters such as heterogeneity, locality, and modularity have emerged from attempts to generate random instances similar to industry benchmarks [3, 13, 48, 49].

In contrast, the analysis of WMC algorithms on random instances is only beginning to be developed. Early on, Sang et al. [69, 70] ran one of the WMC algorithms on random 3-CNF formulas and observed an easy-hard-easy pattern with respect to (w.r.t.) clause density. Recently, Gupta et al. [52] included some WMC algorithms in their study on phase transitions in knowledge compilation. Two phase transitions were identified: one w.r.t. clause density and another w.r.t. a new parameter called *solution density*. There is also a recent attempt [33] to compare WMC algorithms on random instances of a particular application of WMC, i.e., probabilistic logic programs. However, it finds no meaningful differences among the algorithms in that context. Our work complements previous results by including WMC algorithms of various kinds (i.e., not just those based on knowledge compilation) and introducing another parameter of interest.

What parameters are most appropriate to study WMC? Like SAT [4], WMC is known to be fixed-parameter tractable w.r.t. primal treewidth (or a closely related notion) [5, 26, 30, 69]. However—as we show in Section 4—instances generated by a standard random model for k-CNF formulas fail to exhibit enough variance in primal treewidth for us to infer its effect on the behaviour of the algorithms. Therefore, we present an extension of this model with a parameter that influences primal treewidth. The performance of WMC algorithms that use data structures called *algebraic decision diagrams* (ADDs) [6] is also known to depend on the numerical values of weights [38, 39]. Thus, our random model also includes two parameters that control redundancies in these values.

In addition to introducing a new random model for WMC instances, the contributions of this paper include several findings about the behaviour of WMC algorithms on instances generated by our model. First, we show that the easy-hard-easy pattern w.r.t. density is different for dynamic programming algorithms than for all other algorithms. Second, we present statistical evidence that all the algorithms scale exponentially w.r.t. primal treewidth and estimate how the base of that exponential changes w.r.t. density. Third, we show how the performance of ADD-based algorithms gradually improves w.r.t. the proportion of weights that have repeating values. Fourth, we complement our findings on random instances with an experimental study on WMC competition benchmarks, showing how the best-performing algorithm changes depending on density and primal treewidth.

2 Preliminaries

Notation. For any graph G, we write $\mathcal{V}(G)$ for its set of nodes and $\mathcal{E}(G)$ for its set of edges. Let S be a finite set. We write 2^S to denote the powerset of S and $\mathcal{U}S$ for the discrete uniform probability distribution on S. We represent any other probability distribution as a pair (S, p) where $p\colon S \to [0, 1]$ is a probability mass function. For any probability distribution $\mathcal{P}$, we write $x \leftsquigarrow \mathcal{P}$ to denote the act of sampling x from $\mathcal{P}$. For instance, $x \leftsquigarrow (\{1, 2\}, \{1 \mapsto 0.1, 2 \mapsto 0.9\})$ means that x becomes equal to 1 with probability 0.1 or to 2 with probability 0.9.

By *variable*, we always mean a Boolean variable. A *literal* is either a variable (say, v) or its negation (denoted $\neg v$), respectively called *positive* and *negative* literal. A *clause* is a disjunction of literals. A *formula* is any well-formed expression consisting of variables, negation, conjunction, and disjunction. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, and it is in k-CNF if every clause has exactly k literals. While we use the set-theoretic notation for CNF formulas (e.g., writing $c \in \phi$ to mean that clause c is one of the clauses in formula ϕ), duplicate clauses are still allowed. The *primal graph* of a CNF formula is a graph that has a node for every variable, and there is an edge between two variables if they coappear in some clause. The *primal treewidth* of a formula is the treewidth of its primal graph, where treewidth is as in Definition 1.

Definition 1 ([67]). *A* tree decomposition *of a graph G is a pair (T, χ), where T is a tree and $\chi\colon \mathcal{V}(T) \to 2^{\mathcal{V}(G)}$ is a labelling function, with the following properties:*

- $\bigcup_{t \in \mathcal{V}(T)} \chi(t) = \mathcal{V}(G)$*;*
- *for every $e \in \mathcal{E}(G)$, there is $t \in \mathcal{V}(T)$ such that e has both endpoints in $\chi(t)$;*
- *for all $t, t', t'' \in \mathcal{V}(T)$, if t' is on the path between t and t'', then $\chi(t) \cap \chi(t'') \subseteq \chi(t')$.*

The width *of tree decomposition (T, χ) is $\max_{t \in \mathcal{V}(T)} |\chi(t)| - 1$. The* treewidth *of graph G is the smallest w such that G has a tree decomposition of width w.*

Given a CNF formula ϕ, SAT is a decision problem that asks whether there exists a way to assign values to all variables in ϕ such that ϕ evaluates to true. Such a formula is said to be *satisfiable*; otherwise, it is *unsatisfiable*. #SAT is a problem that asks to count the number of such assignments. WMC extends #SAT with a weight function on literals and asks to compute the sum of the weights of the models of the given formula, where the weight of a model is the product of the weights of the literals in it [19]. For example, the WMC of the formula $x \lor y$ with a weight function $w\colon \{x, y, \neg x, \neg y\} \to \mathbb{R}_{\geq 0}$ defined as $w(x) = 0.3$, $w(y) = 0.2$, $w(\neg x) = 0.7$, $w(\neg y) = 0.8$ is $w(x)w(y) + w(x)w(\neg y) + w(\neg x)w(y) = 0.3 \times 0.2 + 0.3 \times 0.8 + 0.7 \times 0.2 = 0.44$.

3 Background on WMC Algorithms

In this section, we briefly review the three major approaches to WMC—search, knowledge compilation, and dynamic programming—and their corresponding algorithms. The main search-based WMC algorithm CACHET[1] [69] is based on a conflict-driven clause learning SAT solver [60], which is then extended with a component caching scheme and adapted to counting.

Knowledge compilation refers to transformations of propositional formulas into more restrictive formats that make various operations (such as model counting) tractable in the size of the representation [32]. C2D[2] [30], D4[3] [58], and MINIC2D[4] [61] are all algorithms of this type. C2D compiles to deterministic decomposable negation normal form (d-DNNF) [27]. Similarly, D4 compiles to decision-DNNF (also known as decomposable decision graphs) [42]. The only difference between d-DNNF and decision-DNNF is that decision-DNNF has if-then-else constructions instead of disjunctions [58]. Finally, MINIC2D compiles to decision-SDDs—a subset of sentential decision diagrams (SDDs) that form a subset of d-DNNF [31].

All of the algorithms mentioned above run the same way regardless of whether computing WMC or #SAT. Two recent WMC algorithms instead use data structures whose size (and thus the runtime of the algorithm) depends on the numerical

[1] https://henrykautz.com/Cachet/index.htm
[2] http://reasoning.cs.ucla.edu/c2d/
[3] https://www.cril.univ-artois.fr/KC/d4.html
[4] http://reasoning.cs.ucla.edu/minic2d/

values of weights. These data structures represent *pseudo-Boolean functions*, i.e., functions of the form $f: 2^X \to \mathbb{R}_{\geq 0}$, where X is a set. ADDMC is the first such algorithm [38]. It uses ADDs to represent pseudo-Boolean functions, combining and simplifying them in a bottom-up dynamic programming fashion. Since the size of an ADD for f depends on the cardinality of the range of f [6], the performance of the algorithm is sensitive to the numerical values of weights, e.g., to how frequently they repeat. DPMC[5] extends ADDMC in two ways [39]. First, DPMC allows for the order and nesting of operations on ADDs to be determined from an approximately-minimal-width tree decomposition rather than by heuristics.[6] Second, tensors are offered as an alternative to ADDs.

In all known parameterised complexities of WMC algorithms, the exponential factor is a function of primal treewidth or a closely related parameter. Interestingly, c2d is specifically designed to handle high primal treewidth (which the author refers to as *connectivity* [25]) and improves upon an earlier algorithm that has $\mathcal{O}(mw2^w)$ time complexity, where m is the number of clauses, and w is the width of the decomposition tree which is known to be at most primal treewidth [26, 30]. While the complexity of Cachet was not analysed directly, the algorithm is based on component caching which is known to have a $2^{\mathcal{O}(w)}n^{\mathcal{O}(1)}$ time complexity, where n is the number of variables, and w is the branchwidth of the underlying hypergraph [5, 69], which is known to be within a constant factor of primal treewidth [68]. Similarly, the complexity of DPMC is not described in the paper, although the authors define a notion of width w that is at most primal treewidth plus one and estimate the runtime of the (execution part of the) algorithm to be proportional to 2^w [39].

4 Random *k*-CNF Formulas with Varying Primal Treewidth

Our random model is based on the following parameters: (a) the number of variables $\nu \in \mathbb{N}^+$, (b) density $\mu \in \mathbb{R}_{>0}$, (c) clause width $\kappa \in \mathbb{N}^+$ (for k-CNF formulas, $\kappa = k$), (d) a parameter $\rho \in [0, 1]$ that influences the primal treewidth of the formula, (e) the proportion $\delta \in [0, 1]$ of variables x such that $w(x) = 1$ and $w(\neg x) = 0$ or $w(x) = 0$ and $w(\neg x) = 1$, (f) and the proportion $\epsilon \in [0, 1 - \delta]$ of variables x such that $w(x) = w(\neg x) = 0.5$. The first three parameters are the standard parameters used to generate random κ-CNF formulas with $\nu\mu$ clauses (up to rounding). Parameters δ and ϵ control the numerical values of weights similarly to determinism and parameter equality—facets of local structure considered in the literature on probabilistic models [74]. While all other WMC algorithms disregard the weights, DPMC [39] can exploit both determinism and equal weights to solve the problem faster. Indeed, higher values of both δ and ϵ result in ADDs having fewer real-numbered values they need to represent. Thus, the ADDs are smaller and can be handled more efficiently.

[5] https://github.com/vardigroup/dpmc

[6] There is also a recent line of work in using tree decompositions to guide the heuristics of search-based model counters [57].

Algorithm 1: Generating a random formula

```
   Input: $\nu, \kappa \in \mathbb{N}^+$ (such that $\kappa < \nu$), $\mu \in \mathbb{R}_{>0}$, $\rho \in [0, 1]$.
   Output: A $k$-CNF formula $\phi$.
1  $\phi \leftarrow$ empty CNF formula;
2  $G \leftarrow$ empty graph;
3  for $i \leftarrow 1$ to $\lfloor \nu\mu \rfloor$ do
4      $X \leftarrow \emptyset$;
5      for $j \leftarrow 1$ to $\kappa$ do
6          $x \leftarrow$ NewVariable($X$, $G$);
7          $\mathcal{V}(G) \leftarrow \mathcal{V}(G) \cup \{ x \}$;
8          $\mathcal{E}(G) \leftarrow \mathcal{E}(G) \cup \{ \{ x, y \} \mid y \in X \}$;
9          $X \leftarrow X \cup \{ x \}$;
10     $\phi \leftarrow \phi \cup \{ l \leftsquigarrow \mathcal{U}\{ x, \neg x \} \mid x \in X \}$;
11 return $\phi$;
12 Function NewVariable($X$, $G$):
13     $N \leftarrow \{ e \in \mathcal{E}(G) \mid |e \cap X| = 1 \}$;
14     if $N = \emptyset$ then
15         return $x \leftsquigarrow \mathcal{U}(\{ x_1, x_2, \dots, x_\nu \} \setminus X)$;
16     return $x \leftsquigarrow \Big( \{ x_1, x_2, \dots, x_\nu \} \setminus X,$
17         $y \mapsto \frac{1-\rho}{\nu - |X|} + \rho \frac{|\{ z \in X \mid \{ y, z \} \in \mathcal{E}(G) \}|}{|N|} \Big)$;
```

The process for generating random k-CNF formulas is summarized as Algorithm 1. The idea behind the algorithm is to reduce the density of the primal graph (via having some overlapping edges) while: (a) avoiding having many variables that do not occur in any clause and (b) promoting tree-like subgraphs that are likely to have low treewidth. For the rest of this section, let $\{x_i\}_{i=1}^{\nu}$ be the variables of the formula under construction. We simultaneously construct both formula ϕ and its primal graph G.[7] Each iteration of the first for-loop adds a clause to ϕ. This is done by constructing a set X of variables to be included in the clause, and then randomly adding either x or $\neg x$ to the clause for each $x \in X$ on line 10. Function `NewVariable` randomly selects each new variable x, and lines 7–9 add x to the graph and the formula while also adding edges between x and all the other variables in the clause. To select each variable, line 13 defines set N to contain all edges with exactly one endpoint in X. The edges added to G by line 8 form a subset of N. If $N = \emptyset$, we select the variable uniformly at random (u.a.r.) from all viable candidates. Otherwise, ρ determines how much we bias the uniform distribution towards variables that would introduce fewer new edges to G.

When $\rho = 0$, Algorithm 1 reduces to what has become the standard random model for k-CNF formulas. Equivalently to Franco and Paull [46], we indepen-

[7] The idea to directly take the primal graph into consideration while generating the formula is new—cf. random SAT instance generators based on, e.g., adversarial evolution [56] and community structure [48].

dently sample a fixed number of clauses, each clause has no duplicate variables, and each variable becomes either a positive or a negative literal with equal probabilities. At the other extreme, when $\rho = 1$, the first variable of a clause is still chosen u.a.r., but all other variables are chosen from those that already coappear in a clause (if possible). The probability that a variable is selected to be included in a clause scales linearly w.r.t. the proportion of edges in N that would be repeatedly added to G if the variable y was added to the clause. This is an arbitrary choice (which appears to work well, see Section 4.1) although alternatives (e.g., exponential scaling) could be considered. As long as $\rho < 1$, every k-CNF formula retains a positive probability of being generated by the algorithm.

To transform the generated formula into a WMC instance, we need to define weights on literals.[8] We want to partition all variables into three groups: those with weights equal to zero and one, those with weights equal to 0.5, and those with arbitrary weights, where the size of each group is determined by δ and ϵ. To do this, we sample a permutation $\pi \leftsquigarrow \mathcal{U}S_\nu$ (where S_ν is the permutation group on $\{1, 2, \dots, \nu\}$), and assign to each *variable* x_n a weight drawn u.a.r. from (a) $\mathcal{U}\{\,0, 1\,\}$ if $\pi(n) \le \nu\delta$, (b) $\mathcal{U}\{\,0.5\,\}$ if $\nu\delta < \pi(n) \le \nu\delta + \nu\epsilon$, and (c) $\mathcal{U}\{\,0.01, 0.02, \dots, 0.99\,\}$[9] if $\pi(n) > \nu\delta + \nu\epsilon$. We extend these weights to weights on *literals* by choosing the weight of each positive literal to be equal to the weight of its variable, and the weight of each negative literal to be such that $w(x) + w(\neg x) = 1$ for all variables x. This restriction is to ensure consistent answers among the algorithms.

Example 1. Let $\nu = 5$, $\mu = 0.6$, $\kappa = 3$, $\rho = 0.3$, $\delta = 0.4$, and $\epsilon = 0.2$ and consider how Algorithm 1 generates a random instance. Since $\kappa = 3$, and $\lfloor \nu\mu \rfloor = 3$, the algorithm will generate a 3-CNF formula with three clauses.

For the first variable of the first clause, we are choosing u.a.r. from $\{\,x_1, x_2, \dots, x_5\,\}$. Suppose the algorithm chooses x_5. Graph G then gets its first node but no edges. The second variable is chosen u.a.r. from $\{\,x_1, x_2, x_3, x_4\,\}$. Suppose the second variable is x_2. Then G gets another node and its first edge between x_2 and x_5. The third variable in the first clause is similarly chosen u.a.r. from $\{\,x_1, x_3, x_4\,\}$ because the only edge in G has both endpoints in $X = \{\,x_2, x_5\,\}$, and so $N = \varnothing$. Suppose the third variable is x_1. Graph G becomes a triangle connecting x_1, x_2, and x_5. Each of the three variables is then added to the clause as either a positive or a negative literal (with equal probabilities). Thus, the first clause becomes, e.g., $\neg x_5 \lor x_2 \lor x_1$.

The first variable of the second clause is chosen u.a.r. from $\{\,x_1, x_2, \dots, x_5\,\}$. Suppose it is x_5 again. When the function `NewVariable` tries to choose the second variable, $X = \{\,x_5\,\}$, and so $N = \{\,\{\,x_1, x_5\,\}, \{\,x_2, x_5\,\}\,\}$. The second variable is chosen from the discrete probability distribution $\Pr(x_1) = \Pr(x_2) = \frac{1-0.3}{5-1} + 0.3 \times \frac{1}{2} = 0.325$ and $\Pr(x_3) = \Pr(x_4) = \frac{1-0.3}{5-1} = 0.175$.

[8] Algorithms such as DPMC and ADDMC [38, 39] support a more flexible way of assigning weights that can lead to significant performance improvements [34, 35].

[9] For convenience, we represent $(0, 1)$ as 99 discrete values.

We skip the details of how all remaining variables and clauses are selected and consider the weight assignment. First, we shuffle the list of variables and get, e.g., $L = (x_4, x_3, x_2, x_1, x_5)$. This means that the first $\nu\delta = 5 \times 0.4 = 2$ variables of L get weights u.a.r. from $\{ 0, 1 \}$, the next $\nu\epsilon = 5 \times 0.2 = 1$ variable gets a weight of 0.5, and the remaining two variables get weights u.a.r. from $\{ 0.01, 0.02, \dots, 0.99 \}$. The weight function $w\colon \{ x_1, x_2, \dots, x_5, \neg x_1, \neg x_2, \dots, \neg x_5 \} \to [0, 1]$ can then be defined as, e.g., $w(x_4) = w(\neg x_3) = 0$, $w(x_3) = w(\neg x_4) = 1$, $w(x_2) = w(\neg x_2) = 0.5$, $w(x_1) = 0.23$, $w(\neg x_1) = 0.77$, $w(x_5) = 0.18$, and $w(\neg x_5) = 0.82$.

4.1 Validating the Model

The idea behind our model is that manipulating the value of ρ should allow us to generate instances of varying primal treewidth. Is this effect observable in practice? In addition, as WMC instances are mostly used for probabilistic inference, they tend to be satisfiable. Therefore, we want to filter out unsatisfiable instances from those generated by the model and need to ensure that the proportion of satisfiable instances remains sufficiently high. Given that higher values of ρ can result in constraints on variables being more localised and concentrated, we ask: are instances generated with higher values of ρ less likely to be satisfiable? To answer both questions, we run the following experiment.

Experiment 1. We fix $\nu = 100, \delta = \epsilon = 0$, and consider random instances with $\mu = 2.5 \times \sqrt{2}^{-5}, 2.5 \times \sqrt{2}^{-4}, \dots, 2.5 \times \sqrt{2}^{5}$, $\kappa = 2, 3, 4, 5$, and ρ going from 0 to 1 in steps of 0.01. For each combination of parameters, we generate ten instances.[10] We check if each instance is satisfiable using MiniSat[11] 2.2.0 [41] and calculate its (approximate) primal treewidth using HTD[12] [1].

Remark 1. Here and henceforth, we use HTD to provide heuristic upper bounds on true treewidth as exact computation would make the experiments significantly more time-consuming. However, we compared the accuracy of HTD with exact treewidth algorithm Jdrasil[13] [7] on 3 % of our random instances. The difference between the upper bound produced by HTD and the exact value was never higher than four and up to two in 85 % of all cases. Since the difference is small enough to not have a qualitative effect, hereafter we write '(primal) treewidth' to mean 'the heuristic upper bound on treewidth found by HTD'.

Figure 1 shows the relationship between ρ and primal treewidth. Except for when both μ and κ are very low (i.e., the formulas are small in both clause width and the number of clauses), primal treewidth decreases as ρ increases. This downward trend becomes sharper as μ increases, however, not uniformly: it splits into a roughly linear segment that approaches a horizontal line (for most

[10] Since one expects similar values of ρ to produce instances with similar properties, and ρ's are enumerate quite densely, generating only ten instances is sufficient.
[11] http://minisat.se/MiniSat.html
[12] https://github.com/mabseher/htd
[13] https://maxbannach.github.io/Jdrasil/

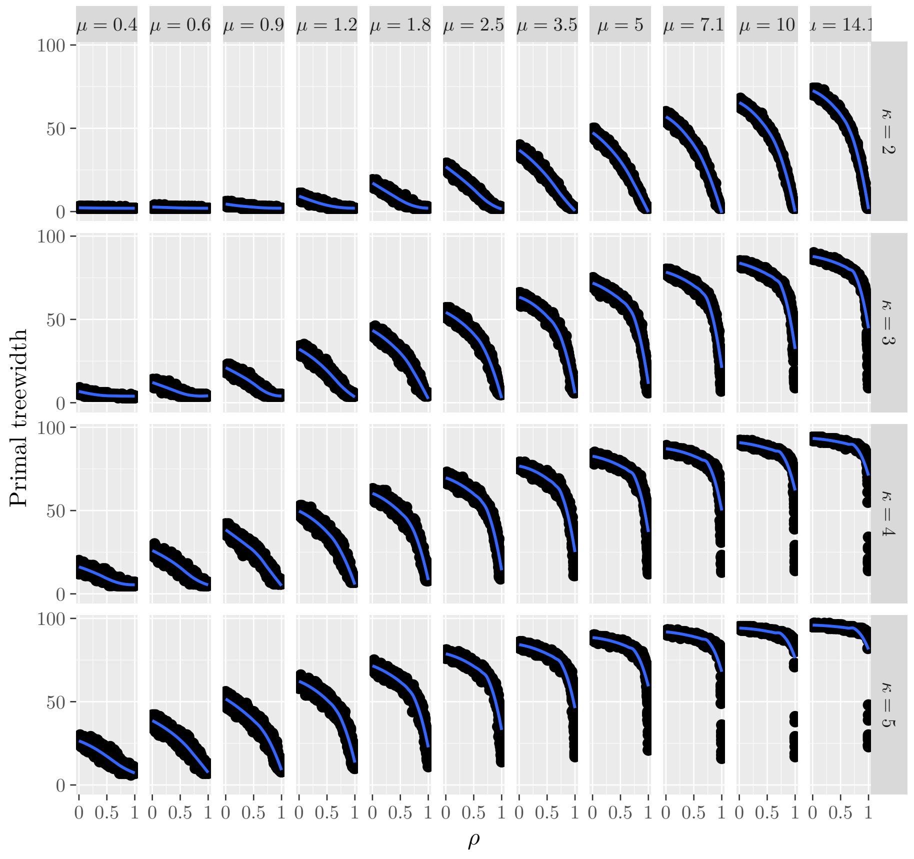


Fig. 1. The relationship between ρ and primal treewidth for various values of μ and κ for k-CNF formulas from Experiment 1. Black points represent individual instances, and blue lines are smoothed means computed using locally weighted smoothing. The values of μ are rounded to one decimal place.

values of ρ) and a sharply-decreasing segment that approaches a vertical line (when ρ is close to one). Higher values of κ seem to expedite this transition, i.e., with a higher value of κ, a lower value of μ is sufficient for a smooth downward curve between ρ and primal treewidth to turn into a combination of a horizontal and a vertical line. While this behaviour may be troublesome when generating formulas with higher values of μ (almost all of which would be unsatisfiable), the relationship between ρ and primal treewidth is excellent for generating 3-CNF formulas close to and below the satisfiability threshold of 4.25 [23]. Regarding satisfiability, the proportion of satisfiable 3-CNF formulas drops from 63.6 % when $\rho = 0$ to 50.9 % when $\rho = 1$, so—while ρ does affect satisfiability—the effect is not significant enough to influence our experimental setup in the next section.

5 Experimental Results

In Section 5.1, we examine how the runtimes of WMC algorithms change w.r.t. the parameters of our random model. Then, in Section 5.2, we run an experiment with WMC competition benchmarks to check whether the conclusions drawn from random instances apply to real data. Full experimental results as well as an implementation of Algorithm 1 are available at https://github.com/dilkas/cpaior23-d.

For all of these experiments, we use Scientific Linux 7, GCC 10.2.0, Python 3.8.1, R 4.1.0, C2D 2.20 [30], CACHET 1.22 [69], HTD 1.2.0 [1], and perform no preprocessing. With both C2D and D4 [58], we use QUERY-DNNF[14] to compute the numerical answer from the compiled circuit. We omit ADDMC [38] from our experiments as it exceeds time and memory limits on too many instances; however, observations about the behaviour of DPMC [39] apply to ADDMC as well, with the addendum that the tree decomposition implicitly used by ADDMC may have a significantly higher width. DPMC is run with tree decomposition-based planning (using one iteration of HTD) and ADD-based execution—the combination that was found to be most effective by Dudek et al. [39].

5.1 Experiments on Random Instances

We restrict our attention to 3-CNF formulas, generate 100 satisfiable instances for each *combination* of parameters, and run each of the five algorithms with a 500 s time limit and an 8 GiB memory limit on Intel Xeon E5–2630. While both limits are somewhat low, we prioritise large numbers of instances to increase the accuracy and reliability of our results. Unless stated otherwise, in each plot of this section, lines denote median values, and shaded areas show interquartile ranges. We run the following three experiments, setting $\nu = 70$ in all of them as we found that this produces instances of suitable difficulty.

Experiment 2 (Density and Primal Treewidth). Let $\nu = 70$, μ go from 1 to 4.3 in steps of 0.3, ρ go from 0 to 0.5 in steps of 0.01, and $\delta = \epsilon = 0$.

Experiment 3 (δ). Let $\nu = 70$, $\mu = 2.2$[15], $\rho = 0$, δ go from 0 to 1 in steps of 0.01, and $\epsilon = 0$.

Experiment 4 (ϵ). Same as Experiment 3 but with $\delta = 0$ and ϵ going from 0 to 1 in steps of 0.01.

In each experiment, the proportion of algorithm runs that timed out never exceeded 3.8 %. While in Experiment 2 only 1 % of experimental runs ran out of memory, the same percentage was higher in Experiments 3 and 4—10 and 12 %, respectively. D4 [58] and C2D are the algorithms that experienced the most issues fitting within the memory limit, accounting for 66–72 % and 28–33 % of such instances, respectively. We exclude the runs that terminated early due to running out of memory from the rest of our analysis.

[14] http://www.cril.univ-artois.fr/kc/d-DNNF-reasoner.html

[15] Experiment 2 shows this density to be the most challenging for DPMC.

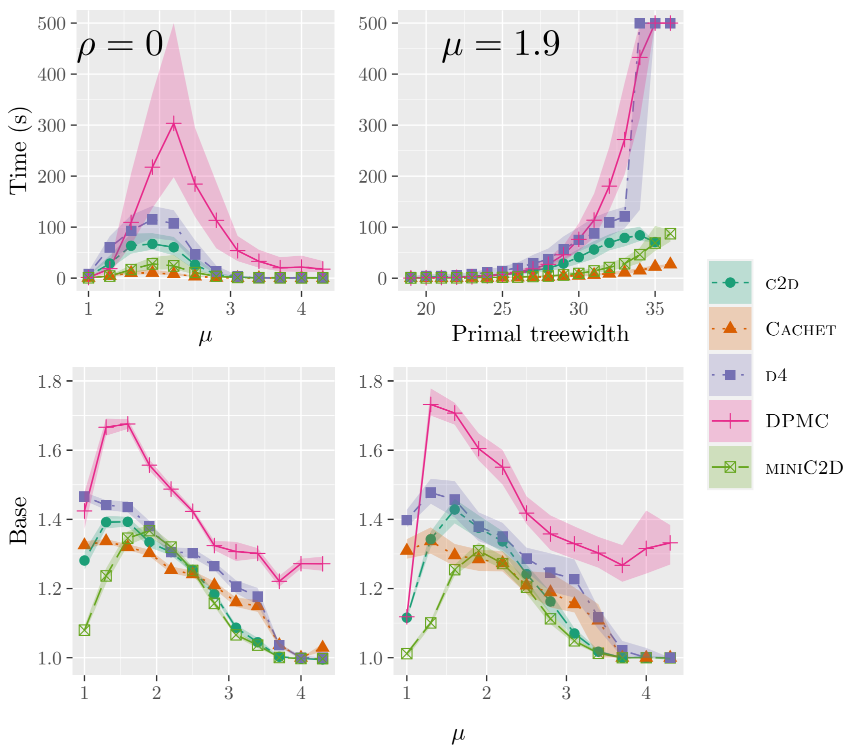
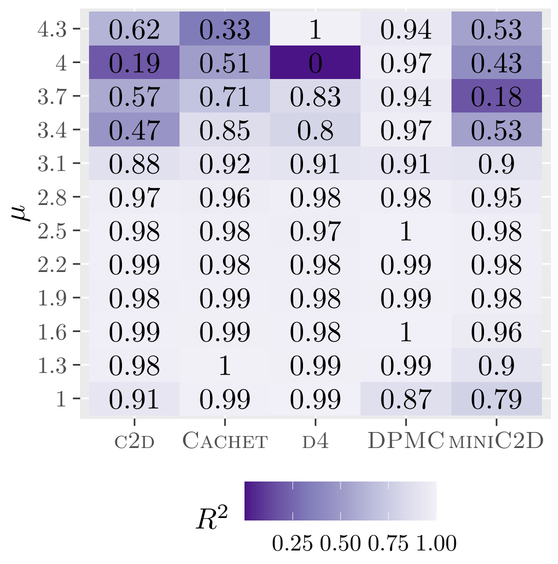


Fig. 2. Visualisations of the data from Experiment 2. The top-left plot shows how the runtime of each algorithm changes w.r.t. density when $\rho = 0$. The top-right plot shows changes in the runtime of each algorithm w.r.t. primal treewidth with μ fixed at 1.9. The plots at the bottom show how the estimated base of the exponential relationship between primal treewidth and the runtime of each algorithm depends on μ. The bottom-left plot is for the simple linear model (with shaded areas showing standard error), and the bottom-right plot uses the estimates provided by ESA [64] (with shaded areas showing 95 % confidence intervals).

In Experiment 2, we investigate how the runtime of each algorithm depends on the density and primal treewidth by varying both μ and ρ. The results are in Figure 2. The first thing to note is that the peak hardness w.r.t. density occurs at around 1.9 for all algorithms except for DPMC, which peaks at 2.2 instead.[16] This finding is consistent with previous works that show Cachet, miniC2D, and a d-DNNF compilation algorithm to peak at 1.8 [28, 52, 69].[17]

The other question we want to investigate is how each algorithm scales w.r.t. primal treewidth. The top-right plot in Figure 2 shows this relationship for a fixed value of μ, and one can see some evidence that the runtime of DPMC

[16] The exact values—while illegible from the plot—can be confirmed by numerical data.
[17] For comparison, #SAT algorithms are known to peak at densities 1.2 and 1.5 [9, 12].



μ	c2d	Cachet	d4	DPMC	miniC2D
4.3	0.62	0.33	1	0.94	0.53
4	0.19	0.51	0	0.97	0.43
3.7	0.57	0.71	0.83	0.94	0.18
3.4	0.47	0.85	0.8	0.97	0.53
3.1	0.88	0.92	0.91	0.91	0.9
2.8	0.97	0.96	0.98	0.98	0.95
2.5	0.98	0.98	0.97	1	0.98
2.2	0.99	0.98	0.98	0.99	0.98
1.9	0.98	0.99	0.98	0.99	0.98
1.6	0.99	0.99	0.98	1	0.96
1.3	0.98	1	0.99	0.99	0.9
1	0.91	0.99	0.99	0.87	0.79

Fig. 3. The coefficients of determination (rounded to one decimal place) of all the linear models fitted for the top-right subplot of Figure 2

grows faster w.r.t. primal treewidth compared to the other algorithms. We use two statistical techniques to quantify this growth: a simple linear regression model and the empirical scaling analyzer (ESA) v2[18] [64]. In both cases, for each algorithm and value of μ in Experiment 2, we select the median runtime for all available primal treewidth values. In the former case, we fit the model $\ln t \sim \alpha w + \beta$, where t is the median runtime of the algorithm, w is the primal treewidth, and α and β are parameters.[19] In other words, we express median runtime as $e^{\beta}(e^{\alpha})^{w}$. In the latter case, we run ESA with 1001 bootstrap samples, a window of 101, and use the first 30 % of the data for training.

The results of both models are qualitatively the same (except for DPMC run on instances with $\mu = 1$) and are displayed at the bottom of Figure 2. We find that DPMC scales worse w.r.t. primal treewidth than any other algorithm across all values of μ and is the only algorithm that does not become indifferent to primal treewidth when faced with high-density formulas. A second look at the top-left subplot of Figure 2 suggests an explanation for the latter observation. The runtimes of all algorithms except for DPMC approach zero when $\mu > 3$ while the median runtime of DPMC approaches a small non-zero constant instead. This observation also explains why Figure 3 shows that the fitted models fail to explain the data for non-ADD algorithms running on high-density instances—

[18] https://github.com/YashaPushak/ESA

[19] Similar analyses have been used to investigate polynomial-to-exponential phase transitions in SAT [21] and the behaviour of SAT solvers on CNF-XOR formulas [37].

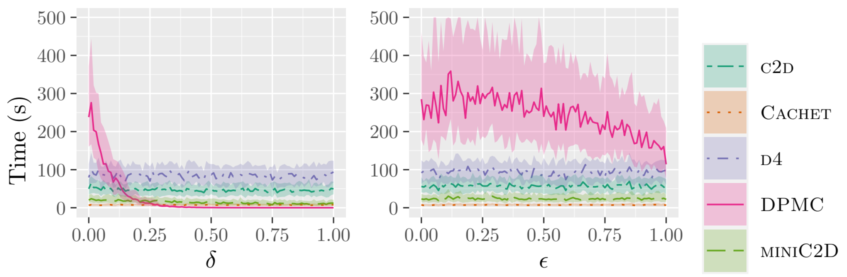


Fig. 4. Changes in the runtime of each algorithm as a result of changing δ (on the left-hand side) and ϵ (on the right-hand side) as in Experiments 3 and 4

the runtimes are too small to be meaningful. In all other cases, an exponential relationship between primal treewidth and runtime fits the experimental data remarkably well.

Another thing to note is that miniC2D [61] is the only algorithm that exhibits a clear low-high-low pattern in the bottom subplots of Figure 2. To a smaller extent, the same may apply to c2d and DPMC, although the evidence for this is limited due to relatively large gaps between different values of μ. In contrast, the runtimes of Cachet and d4 remain dependent on primal treewidth even when the density of the WMC instance is very low, suggesting that miniC2D should have an advantage on low-density high-primal-treewidth instances.

Finally, Experiments 3 and 4 investigate how changing the numerical values of weights can simplify a WMC instance. The results are in Figure 4. As expected, the runtimes of all algorithms other than DPMC stay the same regardless of the value of δ or ϵ. The runtime of DPMC, however, experiences a sharp (exponential?) decline with increasing δ. The decline w.r.t. ϵ is also present, although significantly less pronounced and with high variance.

To sum, we found that c2d and d4 are the most memory-intensive algorithms, Cachet is great on random instances in general, miniC2D excels on low-density high-primal-treewidth instances, and DPMC is at its best on low-density low-primal-treewidth instances. Furthermore, a median instance with all weights equal to each other is about three times easier for DPMC than a median instance with random weights. Another important observation is about how peak hardness w.r.t. density depends on the algorithm: DPMC peaks at a higher density than all other WMC algorithms, which peak at a higher density than (some) #SAT algorithms.

5.2 Experiments on Competition Benchmarks

To check whether our observations on random instances are accurate on real data, we use the 100 public instances from track 2 of the 2022 model counting competition[20]—an annual competition that has been running since 2020 [43].

[20] https://mccompetition.org/2022/mc_description

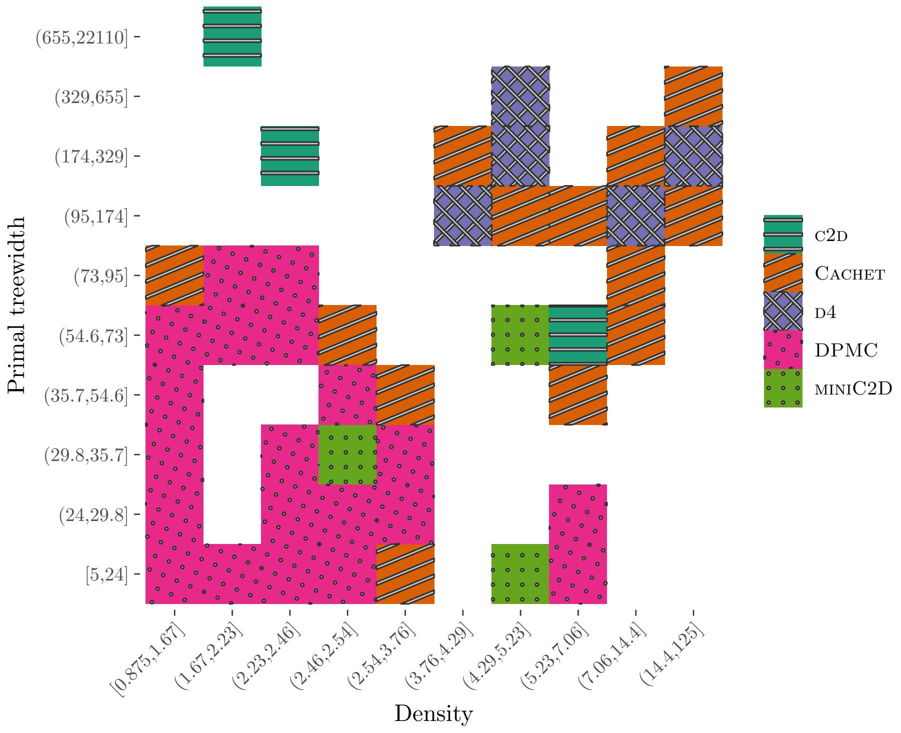


Fig. 5. The best-performing algorithm for each combination of density and primal treewidth according to the experiments on competition benchmarks. Both ranges of values are divided into ten bins so that there are ten instances in each bin. The best-performing algorithm for each combination of bins is the algorithm that solved the largest number of instances, with ties broken by minimising total runtime. An empty cell means that either no benchmark had this combination of density and primal treewidth or all algorithms failed on all such instances.

This time, we run the algorithms on Intel Xeon Gold 6138 with 32 GiB of memory and a one hour time limit. As in Section 5.1, we compute the density and the primal treewidth of each instance.

Figure 5 shows the best-performing algorithm for various combinations of the parameters. We observe that: (a) DPMC [39] is best on most instances with low primal treewidth, (b) c2d [30] can handle some low-density high-primal-treewidth instances that all the other algorithms fail on, (c) Cachet [69] (as well as d4 [58] to some extent) excel when both density and primal treewidth are quite high, (d) and miniC2D [61] does not have a clear niche. Hence, the observation in Section 5.1 that DPMC is good on low-density low-primal-treewidth instances is confirmed by the experiments on real data. Moreover, higher density instances can also favour DPMC as long as primal treewidth is sufficiently low. On the other hand, while the experiments on random instances suggested that miniC2D might excel at low-density high-primal-treewidth instances, our experiments on competition benchmarks suggest otherwise. Instead, c2d, Cachet or DPMC could be the right choice depending on the exact values.

6 Conclusions and Future Work

In this paper, we studied the behaviour of and differences among WMC algorithms on random instances generated by a standard model for k-CNF formulas extended with parameters that control primal treewidth and literal weights. Among other things, we established statistical evidence for the existence of an exponential relationship between primal treewidth and the runtimes of all WMC algorithms on instances generated by our model. The runtime of the ADD-based algorithm was observed to peak at a higher density, scale worse w.r.t. primal treewidth, and depend negatively on repeating weight values compared to algorithms based on search or knowledge compilation. These observations can, to some degree, be extended to a closely related weighted projected model counting algorithm [40] as well as to other applications of ADDs more generally, e.g., probabilistic inference [18, 50] and stochastic planning [54].

One limitation of our work is that variability in primal treewidth was achieved via a parameter, and this could bias randomness in some unexpected way (although it is encouraging that there is only a slight decrease in the proportion of satisfiable instances between $\rho = 0$ and $\rho = 1$). Perhaps a theoretical investigation of the proposed model is warranted, including a characterisation of how ρ influences primal treewidth and the structure of the primal graph more generally. Since treewidth is widely used in parameterised complexity [36], formally establishing a connection with ρ could make our random model useful for a variety of other hard computational problems.

To keep the number of experiments feasible, we restricted our attention to 3-CNF formulas, although, of course, this is not very representative of real-world WMC instances. The model could be adapted to generate non-k-CNF formulas, and perhaps a more representative structure could be achieved by introducing new variables that clauses define to be equivalent to select conjunctions of literals as is done in one of the WMC encodings for Bayesian networks [29].

Acknowledgements. The author would like to thank Vaishak Belle and the anonymous reviewers for their feedback on earlier versions of this work. The author was supported by the EPSRC Centre for Doctoral Training in Robotics and Autonomous Systems, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016834/1). This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF) (http://www.ecdf.ed.ac.uk/). For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission.

References

1. Abseher, M., Musliu, N., Woltran, S.: htd - A free, open-source framework for (customized) tree decompositions and beyond. In: Salvagnin, D., Lombardi, M. (eds.) Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10335, pp. 376–386. Springer (2017). https://doi.org/10.1007/978-3-319-59776-8_30
2. Achlioptas, D., Moore, C.: The asymptotic order of the random k-SAT threshold. In: 43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings. pp. 779–788. IEEE Computer Society (2002). https://doi.org/10.1109/SFCS.2002.1182003
3. Ansótegui, C., Bonet, M.L., Levy, J.: Towards industrial-like random SAT instances. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 387–392 (2009), http://ijcai.org/Proceedings/09/Papers/072.pdf
4. Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. J. Artif. Intell. Res. **40**, 353–373 (2011). https://doi.org/10.1613/jair.3152
5. Bacchus, F., Dalmao, S., Pitassi, T.: Solving #SAT and Bayesian inference with backtracking search. J. Artif. Intell. Res. **34**, 391–442 (2009). https://doi.org/10.1613/jair.2648
6. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. Formal Methods Syst. Des. **10**(2/3), 171–206 (1997). https://doi.org/10.1023/A:1008699807402
7. Bannach, M., Berndt, S., Ehlers, T.: Jdrasil: A modular library for computing tree decompositions. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (eds.) 16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK. LIPIcs, vol. 75, pp. 28:1–28:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPIcs.SEA.2017.28
8. Bart, A., Koriche, F., Lagniez, J., Marquis, P.: An improved CNF encoding scheme for probabilistic inference. In: Kaminka, G.A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F., van Harmelen, F. (eds.) ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 613–621. IOS Press (2016). https://doi.org/10.3233/978-1-61499-672-9-613
9. Bayardo Jr., R.J., Pehoushek, J.D.: Counting models using connected components. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA. pp. 157–162. AAAI Press / The MIT Press (2000), http://www.aaai.org/Library/AAAI/2000/aaai00-024.php
10. Belle, V.: Open-universe weighted model counting. In: Singh, S., Markovitch, S. (eds.) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 3701–3708. AAAI Press (2017), http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/15008
11. Belle, V., Passerini, A., Van den Broeck, G.: Probabilistic inference in hybrid domains by weighted model integration. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial

Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 2770–2776. AAAI Press (2015), http://ijcai.org/Abstract/15/392

12. Birnbaum, E., Lozinskii, E.L.: The good old Davis-Putnam procedure helps counting models. J. Artif. Intell. Res. **10**, 457–477 (1999). https://doi.org/10.1613/jair.601
13. Bläsius, T., Friedrich, T., Sutton, A.M.: On the empirical time complexity of scale-free 3-SAT at the phase transition. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 117–134. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_7
14. Chakraborty, S., Fried, D., Meel, K.S., Vardi, M.Y.: From weighted to unweighted model counting. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 689–695. AAAI Press (2015), http://ijcai.org/Abstract/15/103
15. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Approximate model counting. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 1015–1045. IOS Press (2021). https://doi.org/10.3233/FAIA201010
16. Chavira, M., Darwiche, A.: Compiling Bayesian networks with local structure. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005. pp. 1306–1312. Professional Book Center (2005), http://ijcai.org/Proceedings/05/Papers/0931.pdf
17. Chavira, M., Darwiche, A.: Encoding CNFs to empower component analysis. In: Biere, A., Gomes, C.P. (eds.) Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4121, pp. 61–74. Springer (2006). https://doi.org/10.1007/11814948_9
18. Chavira, M., Darwiche, A.: Compiling Bayesian networks using variable elimination. In: Veloso, M.M. (ed.) IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. pp. 2443–2449 (2007), http://ijcai.org/Proceedings/07/Papers/393.pdf
19. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artif. Intell. **172**(6-7), 772–799 (2008). https://doi.org/10.1016/j.artint.2007.11.002
20. Chavira, M., Darwiche, A., Jaeger, M.: Compiling relational Bayesian networks for exact inference. Int. J. Approx. Reason. **42**(1-2), 4–20 (2006). https://doi.org/10.1016/j.ijar.2005.10.001
21. Coarfa, C., Demopoulos, D.D., Aguirre, A.S.M., Subramanian, D., Vardi, M.Y.: Random 3-SAT: The plot thickens. Constraints An Int. J. **8**(3), 243–261 (2003). https://doi.org/10.1023/A:1025671026963
22. Coja-Oghlan, A., Wormald, N.: The number of satisfying assignments of random regular k-SAT formulas. Comb. Probab. Comput. **27**(4), 496–530 (2018). https://doi.org/10.1017/S0963548318000263
23. Crawford, J.M., Auton, L.D.: Experimental results on the crossover point in random 3-SAT. Artif. Intell. **81**(1-2), 31–57 (1996). https://doi.org/10.1016/0004-3702(95)00046-1
24. Dal, G.H., Laarman, A.W., Lucas, P.J.F.: Parallel probabilistic inference by weighted model counting. In: Studený, M., Kratochvíl, V. (eds.) International Conference on

Probabilistic Graphical Models, PGM 2018, 11-14 September 2018, Prague, Czech Republic. Proceedings of Machine Learning Research, vol. 72, pp. 97–108. PMLR (2018), http://proceedings.mlr.press/v72/dal18a.html

25. Darwiche, A.: Compiling knowledge into decomposable negation normal form. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages. pp. 284–289. Morgan Kaufmann (1999), http://ijcai.org/Proceedings/99-1/Papers/042.pdf
26. Darwiche, A.: Decomposable negation normal form. J. ACM **48**(4), 608–647 (2001). https://doi.org/10.1145/502090.502091
27. Darwiche, A.: On the tractable counting of theory models and its application to truth maintenance and belief revision. J. Appl. Non Class. Logics **11**(1-2), 11–34 (2001). https://doi.org/10.3166/jancl.11.11-34
28. Darwiche, A.: A compiler for deterministic, decomposable negation normal form. In: Dechter, R., Kearns, M.J., Sutton, R.S. (eds.) Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada. pp. 627–634. AAAI Press / The MIT Press (2002), http://www.aaai.org/Library/AAAI/2002/aaai02-094.php
29. Darwiche, A.: A logical approach to factoring belief networks. In: Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M. (eds.) Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002. pp. 409–420. Morgan Kaufmann (2002)
30. Darwiche, A.: New advances in compiling CNF into decomposable negation normal form. In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004. pp. 328–332. IOS Press (2004)
31. Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. In: Walsh, T. (ed.) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 819–826. IJCAI/AAAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-143
32. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002). https://doi.org/10.1613/jair.989
33. Dilkas, P., Belle, V.: Generating random logic programs using constraint programming. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12333, pp. 828–845. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_48
34. Dilkas, P., Belle, V.: Weighted model counting with conditional weights for Bayesian networks. In: de Campos, C.P., Maathuis, M.H., Quaeghebeur, E. (eds.) Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2021, Virtual Event, 27-30 July 2021. Proceedings of Machine Learning Research, vol. 161, pp. 386–396. AUAI Press (2021), https://proceedings.mlr.press/v161/dilkas21a.html
35. Dilkas, P., Belle, V.: Weighted model counting without parameter variables. In: Li, C., Manyà, F. (eds.) Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12831, pp. 134–151. Springer (2021). https://doi.org/10.1007/978-3-030-80223-3_10

36. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer (2013). https://doi.org/10.1007/978-1-4471-5559-1

37. Dudek, J.M., Meel, K.S., Vardi, M.Y.: The hard problems are almost everywhere for random CNF-XOR formulas. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 600–606. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/84

38. Dudek, J.M., Phan, V., Vardi, M.Y.: ADDMC: Weighted model counting with algebraic decision diagrams. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. pp. 1468–1476. AAAI Press (2020), https://ojs.aaai.org/index.php/AAAI/article/view/5505

39. Dudek, J.M., Phan, V.H.N., Vardi, M.Y.: DPMC: Weighted model counting by dynamic programming on project-join trees. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12333, pp. 211–230. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_13

40. Dudek, J.M., Phan, V.H.N., Vardi, M.Y.: ProCount: Weighted projected model counting with graded project-join trees. In: Li, C., Manyà, F. (eds.) Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12831, pp. 152–170. Springer (2021). https://doi.org/10.1007/978-3-030-80223-3_11

41. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003). https://doi.org/10.1007/978-3-540-24605-3_37

42. Fargier, H., Marquis, P.: On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA. pp. 42–47. AAAI Press (2006), http://www.aaai.org/Library/AAAI/2006/aaai06-007.php

43. Fichte, J.K., Hecher, M., Hamiti, F.: The model counting competition 2020. ACM J. Exp. Algorithmics **26**, 13:1–13:26 (2021). https://doi.org/10.1145/3459080

44. Fichte, J.K., Hecher, M., Woltran, S., Zisser, M.: Weighted model counting on the GPU by exploiting small treewidth. In: Azar, Y., Bast, H., Herman, G. (eds.) 26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland. LIPIcs, vol. 112, pp. 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.ESA.2018.28

45. Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D.S., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory Pract. Log. Program. **15**(3), 358–401 (2015). https://doi.org/10.1017/S1471068414000076

46. Franco, J., Paull, M.C.: Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. Discret. Appl. Math. **5**(1), 77–87 (1983). https://doi.org/10.1016/0166-218X(83)90017-3
47. Galanis, A., Goldberg, L.A., Guo, H., Yang, K.: Counting solutions to random CNF formulas. SIAM J. Comput. **50**(6), 1701–1738 (2021). https://doi.org/10.1137/20M1351527
48. Giráldez-Cru, J., Levy, J.: Generating SAT instances with community structure. Artif. Intell. **238**, 119–134 (2016). https://doi.org/10.1016/j.artint.2016.06.001
49. Giráldez-Cru, J., Levy, J.: Locality in random SAT instances. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 638–644. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/89
50. Gogate, V., Domingos, P.M.: Approximation by quantization. In: Cozman, F.G., Pfeffer, A. (eds.) UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011. pp. 247–255. AUAI Press (2011)
51. Gogate, V., Domingos, P.M.: Probabilistic theorem proving. Commun. ACM **59**(7), 107–115 (2016). https://doi.org/10.1145/2936726
52. Gupta, R., Roy, S., Meel, K.S.: Phase transition behavior in knowledge compilation. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12333, pp. 358–374. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_21
53. Hecher, M., Thier, P., Woltran, S.: Taming high treewidth with abstraction, nested dynamic programming, and database technology. In: Pulina, L., Seidl, M. (eds.) Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12178, pp. 343–360. Springer (2020). https://doi.org/10.1007/978-3-030-51825-7_25
54. Hoey, J., St-Aubin, R., Hu, A.J., Boutilier, C.: SPUDD: Stochastic planning using decision diagrams. In: Laskey, K.B., Prade, H. (eds.) UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999. pp. 279–288. Morgan Kaufmann (1999)
55. Holtzen, S., Van den Broeck, G., Millstein, T.D.: Scaling exact inference for discrete probabilistic programs. Proc. ACM Program. Lang. **4**(OOPSLA), 140:1–140:31 (2020). https://doi.org/10.1145/3428208
56. Hossain, M.M., Abbass, H.A., Lokan, C., Alam, S.: Adversarial evolution: Phase transition in non-uniform hard satisfiability problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010. pp. 1–8. IEEE (2010). https://doi.org/10.1109/CEC.2010.5586506
57. Korhonen, T., Järvisalo, M.: Integrating tree decompositions into decision heuristics of propositional model counters (short paper). In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021. LIPIcs, vol. 210, pp. 8:1–8:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CP.2021.8
58. Lagniez, J., Marquis, P.: An improved decision-DNNF compiler. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 667–673. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/93

59. Mitchell, D.G., Selman, B., Levesque, H.J.: Hard and easy distributions of SAT problems. In: Swartout, W.R. (ed.) Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992. pp. 459–465. AAAI Press / The MIT Press (1992), http://www.aaai.org/Library/AAAI/1992/aaai92-071.php
60. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001. pp. 530–535. ACM (2001). https://doi.org/10.1145/378239.379017
61. Oztok, U., Darwiche, A.: A top-down compiler for sentential decision diagrams. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 3141–3148. AAAI Press (2015), http://ijcai.org/Abstract/15/443
62. Pote, Y., Joshi, S., Meel, K.S.: Phase transition behavior of cardinality and XOR constraints. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 1162–1168. ijcai.org (2019). https://doi.org/10.24963/ijcai.2019/162
63. Purdom Jr., P.W., Brown, C.A.: An analysis of backtracking with search rearrangement. SIAM J. Comput. **12**(4), 717–733 (1983). https://doi.org/10.1137/0212049
64. Pushak, Y., Hoos, H.H.: Advanced statistical analysis of empirical performance scaling. In: Coello, C.A.C. (ed.) GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020. pp. 236–244. ACM (2020). https://doi.org/10.1145/3377930.3390210
65. Renkens, J., Kimmig, A., Van den Broeck, G., De Raedt, L.: Explanation-based approximate weighted model counting for probabilistic logics. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. pp. 2490–2496. AAAI Press (2014), http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8484
66. Riguzzi, F.: Quantum weighted model counting. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 2640–2647. IOS Press (2020). https://doi.org/10.3233/FAIA200401
67. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. J. Comb. Theory, Ser. B **36**(1), 49–64 (1984). https://doi.org/10.1016/0095-8956(84)90013-3
68. Robertson, N., Seymour, P.D.: Graph minors. X. Obstructions to tree-decomposition. J. Comb. Theory, Ser. B **52**(2), 153–190 (1991). https://doi.org/10.1016/0095-8956(91)90061-N
69. Sang, T., Bacchus, F., Beame, P., Kautz, H.A., Pitassi, T.: Combining component caching and clause learning for effective model counting. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings (2004), http://www.satisfiability.org/SAT04/programme/21.pdf
70. Sang, T., Beame, P., Kautz, H.A.: Heuristics for fast exact model counting. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3569, pp. 226–240. Springer (2005). https://doi.org/10.1007/11499107_17

71. Sang, T., Beame, P., Kautz, H.A.: Performing Bayesian inference by weighted model counting. In: Veloso, M.M., Kambhampati, S. (eds.) Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA. pp. 475–482. AAAI Press / The MIT Press (2005), http://www.aaai.org/Library/AAAI/2005/aaai05-075.php
72. Sharma, S., Roy, S., Soos, M., Meel, K.S.: GANAK: A scalable probabilistic exact model counter. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 1169–1176. ijcai.org (2019). https://doi.org/10.24963/ijcai.2019/163
73. Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., De Raedt, L.: Lifted probabilistic inference by first-order knowledge compilation. In: Walsh, T. (ed.) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 2178–2185. IJCAI/AAAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-363
74. Vlasselaer, J., Meert, W., Van den Broeck, G., De Raedt, L.: Exploiting local and repeated structure in dynamic Bayesian networks. Artif. Intell. **232**, 43–53 (2016). https://doi.org/10.1016/j.artint.2015.12.001
75. Xu, J., Zhang, Z., Friedman, T., Liang, Y., Van den Broeck, G.: A semantic loss function for deep learning with symbolic knowledge. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 5498–5507. PMLR (2018), http://proceedings.mlr.press/v80/xu18h.html
76. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008). https://doi.org/10.1613/jair.2490