Parameterized Complexity of Weighted Model Counting in Theory and Practice

July 3, 2021

1 Introduction

'An accepted practice has often been to test new techniques on problem instances that have been used in previous studies. A disadvantage with such an approach is that the techniques developed may [become] somewhat fitted to those problem instances, in the same way that classification techniques can be over fitted to data.'

TODO

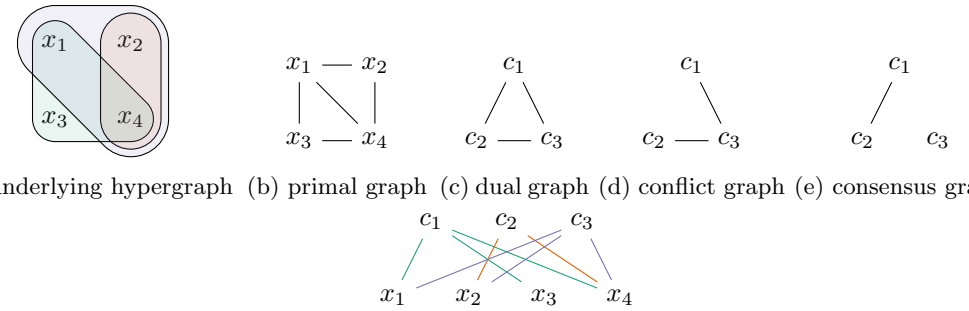
- Replace the citation for my paper with the real one (and consider citing the other paper). I could also cite my CP paper as context [Dilkas and Belle, 2020].
- Fix references.
- List lots of applications.
- Decrease the size of some of the figures.
- When citing a theorem, use the 'author (year)' format (elsewhere where appropriate as well).

2 Preliminaries

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. The *length* of a formula is the total number of literals. While we use 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. Given a CNF formula ϕ, the *propositional satisfiability problem* (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,[1] and *(propositional) model counting* (#SAT) 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 [Chavira and Darwiche, 2008]. For example, the WMC of the formula $x \vee y$ with a weight function $w\colon \{x, y, \neg x, \neg y\} \to \mathbb{R}$ defined as $w(x) = 0.3$, $w(y) = 0.2$, $w(\neg x) = 0.7$, $w(\neg y) = 0.8$ is $w(x)w(y) + w(x)w(\neg y) + w(\neg x)w(y) = 0.3 \times 0.2 + 0.3 \times 0.8 + 0.7 \times 0.2 = 0.44$.

Parameters for parameterized complexity results are often based on structural properties of graphs [Downey and Fellows, 2013], and the parameterized complexities of #SAT and WMC algorithms are no exception. Here we describe several graphs derivable from a CNF formula that are used in relevant complexity

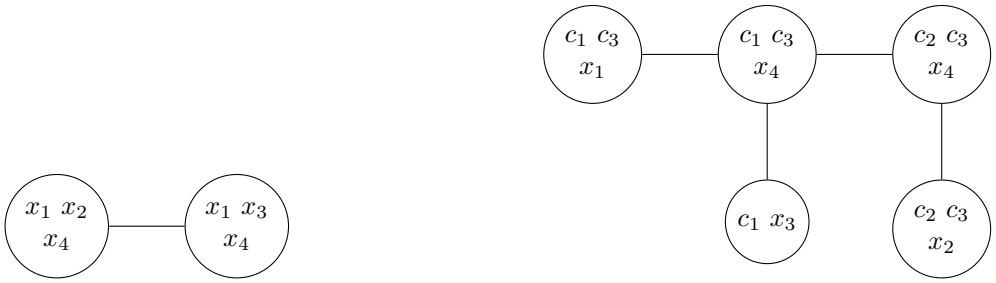
[1] Such a formula is said to be *satisfiable*. Otherwise, it is *unsatisfiable*.



(a) underlying hypergraph (b) primal graph (c) dual graph (d) conflict graph (e) consensus graph

(f) incidence graph

Figure 1: Graphs associated with a CNF formula



(a) of Fig. 1b (b) of Fig. 1f

Figure 2: Minimum-width tree decompositions (both of width two)

results. As an example, consider the CNF formula $(x_4 \vee \neg x_3 \vee x_1) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$. This is a satisfiable formula with model count equal to nine. Let c_1, c_2, and c_3 refer to the three clauses in the order listed. The graphs for this formula are pictured in Fig. 1. In both the *underlying hypergraph* and the *primal graph*[2], there is a node for every variable. In the former, each clause c is represented by a hyperedge that envelops the variables in c whereas in the latter there is an edge between a pair of variables if they coappear in some clause. Effectively, this means that each hyperedge of the underlying hypergraph becomes a clique in the primal graph. Next, the *dual*, *consensus*, and *conflict* graphs have clauses as variables. In the dual graph, two clauses are adjacent if they have variables in common. In the conflict (respectively, consensus) graph, two clauses are adjacent if they do (respectively, do not) contain complementary literals. Finally, the *incidence* graph is a bipartite graph with nodes for both variables and clauses. An edge connects a variable v to a clause c if v appears in c. Treewidth [Robertson and Seymour, 1984] and branchwidth [Robertson and Seymour, 1991] are the two parameters used in parameterized complexity results for #SAT and WMC—both were introduced by Robertson and Seymour in their seminal series of papers on graph minors. We write *primal treewidth* to refer to the treewidth of the primal graph, and similarly for other graphs. As treewidth and tree decompositions will be used in Section 4, we provide the formal definitions below and examples of minimum-width tree decompositions of the primal and incidence graphs of our example formula in Fig. 2.

Definition 1 ([Robertson and Seymour, 1984])**.** A *tree decomposition* of a graph G is a pair (T, χ), where T is a tree and $\chi : \mathcal{V}(T) \to 2^{\mathcal{V}(G)}$ is a labelling function, with the following properties:

- $\bigcup_{t \in \mathcal{V}(T)} \chi(t) = \mathcal{V}(G)$;
- for every edge $e \in \mathcal{E}(G)$, there exists $t \in \mathcal{V}(T)$ such that e has both endpoints in $\chi(t)$;

[2]Primal graphs are known by many other names such as Gaifman, (variable) interaction, and variable incidence graphs.

- 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.

3 WMC Algorithms and Their Complexities

Most WMC algorithms can be broadly classified as based on SAT solvers, knowledge compilation, or pseudo-Boolean function manipulation. Other approaches to WMC not covered here include approximate [Renkens et al., 2014] and parallel algorithms [Dal et al., 2018, Fichte et al., 2018], quantum computing [Riguzzi, 2020] and reduction to model counting [Chakraborty et al., 2015]. CACHET[3] is a WMC algorithm based on the Davis-Putnam-Logemann-Loveland (DPLL) [Davis and Putnam, 1960, Davis et al., 1962] search procedure that underlies most SAT algorithms, combined with component caching and clause learning [Sang et al., 2004]. C2D[4] [Darwiche, 2004], D4[5] [Lagniez and Marquis, 2017], and MINIC2D[6] [Oztok and Darwiche, 2015] are all compilation algorithms that support WMC. C2D compiles to deterministic decomposable negation normal form (d-DNNF) [Darwiche, 2001b]. Similarly, D4 compiles to decision-DNNF (also known as decomposable decision graphs) [Fargier and Marquis, 2006]. The only difference between d-DNNF and decision-DNNF is that decision-DNNF has if-then-else constructions instead of disjunctions [Lagniez and Marquis, 2017]. With both C2D and D4, we use QUERY-DNNF[7] to compute the numerical answer from the compiled circuit. Finally, MINIC2D [Oztok and Darwiche, 2015] compiles to decision-SDDs—a subset of sentential decision diagrams (SDDs) that form a subset of d-DNNF [Darwiche, 2011].

All of the algorithms mentioned above are #SAT algorithms that were later adapted to support weights. Two recently proposed WMC algorithms instead use data structures that natively support weights and can thus take advantage of redundancies in the numerical values of weights or other numbers. These data structures are representations of *pseudo-Boolean functions*, i.e., functions of the form $2^X \to \mathbb{R}$, where X is a set, and 2^X denotes its powerset. ADDMC is the first such algorithm [Dudek et al., 2020a]. It uses *algebraic decision diagrams* (ADDs) [Bahar et al., 1997] as the representation for pseudo-Boolean functions (ADDs are described in more detail in Section 4.1). DPMC[8] extends ADDMC in two ways [Dudek et al., 2020b]. 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. Second, tensors are offered as an alternative to ADDs. In this paper, we focus on tree decomposition-based planning and ADD-based execution—the best-performing combination in the original set of experiments [Dudek et al., 2020b]. Note that the complexity results in Section 4 apply equally to both ADDMC and DPMC (with the observation that the implicit tree decomposition used by ADDMC may have significantly higher width), but we omit ADDMC from our experiments in Section 6 because it would exceed time and memory limits on too many instances.

Some parameterized complexity results already exist for CACHET [Sang et al., 2004] and C2D [Darwiche, 2004] but not for D4 [Lagniez and Marquis, 2017] or MINIC2D [Oztok and Darwiche, 2015] (and the complexity of ADDMC [Dudek et al., 2020a] and DPMC [Dudek et al., 2020b] is the topic of Section 4). While the complexity of CACHET has not been analysed directly, CACHET 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) [Bacchus et al., 2009, Sang et al., 2004]. Interestingly, C2D is specifically designed to handle high primal treewidth (which the author calls *connectivity* [Darwiche, 1999]) 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) [Darwiche, 2001a, Darwiche, 2004].

Parameterized complexity of #SAT is a much more studied subject. #SAT is known to be *fixed-parameter tractable* (FPT; i.e., the problem can be solved in $f(k)\|\phi\|^{\mathcal{O}(1)}$ time, where f is some computable function, and

[3] https://cs.rochester.edu/u/kautz/Cachet/
[4] http://reasoning.cs.ucla.edu/c2d/
[5] https://www.cril.univ-artois.fr/KC/d4.html
[6] http://reasoning.cs.ucla.edu/minic2d/
[7] http://www.cril.univ-artois.fr/kc/d-DNNF-reasoner.html
[8] https://github.com/vardigroup/dpmc

$\|\phi\|$ is the length of the formula) with respect to (w.r.t.) incidence treewidth as a consequence of Courcelle's Theorem (i.e., model count is definable in monadic second-order logic) [Courcelle et al., 2001]. Parameters that are known to make #SAT FPT include primal, dual, and incidence treewidth [Samer and Szeider, 2010]. Ganian and Szeider [Ganian and Szeider, 2021] also show #SAT to be FPT w.r.t. consensus treewidth and W[1]-hard (i.e., not FPT under standard assumptions) w.r.t. conflict treewidth. They also introduce another parameter that makes #SAT FPT, which is inspired by modularity of graphs.

TODO: reference and describe modularity.

4 Parameterized Complexity of DPMC

For any graph G, we write $\mathcal{V}(G)$ for its set of vertices, $\mathcal{E}(G)$ for its set of edges. If G is a tree (respectively, a directed graph), then we denote its set of leaves (respectively, sinks) as $\mathcal{L}(G)$, and $\mathcal{C}(v)$ denotes the children (respectively, direct successors) of a vertex $v \in \mathcal{V}(G)$. For any function $f\colon X \to Y$, we denote its domain as $\mathrm{dom}(f)$ (i.e., $\mathrm{dom}(f) = X$) and its codomain as $\mathrm{cod}(f)$ (i.e., $\mathrm{cod}(f) = Y$). We write $\mathbb{N}_0$ and $\mathbb{N}^+$ for natural numbers with and without zero, respectively.

We generalise some notation for pseudo-Boolean functions from previous work [Dilkas and Belle, 2021b]. Let X be a set of variables, ϕ a propositional formula over X, and $f, g\colon 2^X \to \mathbb{R}$ pseudo-Boolean functions. We write $[\phi]_g^f$ to denote a function $2^X \to \mathbb{R}$ defined as

$$[\phi]_g^f(Y) \coloneqq \begin{cases} f(Y) & \text{if } Y \vDash \phi \\ g(Y) & \text{if } Y \nvDash \phi \end{cases}$$

for any $Y \subseteq X$. Two special cases will be particularly useful. First, if f and g are real numbers instead, they are to be interpreted as constant pseudo-Boolean functions $2^X \to \mathbb{R}$. Second, if ϕ is just a variable name, then $Y \vDash \phi$ is equivalent to $\phi \in Y$.

4.1 ADDs, Operations on ADDs, and Their Complexities

Our definition of an ADD is partially based on the original definition [Bahar et al., 1997] as well as more recent work [Dudek et al., 2020b] but states some details more explicitly. The definition can also be stated more generally to use any set instead of $\mathbb{R}$ and include the possibility of a single ADD representing multiple functions.

Definition 2. Given a set of variables X and a variable ordering represented as an injection $\sigma\colon X \to \mathbb{N}^+$, an *ADD* is a tuple $(G, r, \rho, \chi, \epsilon)$ where:

- G is a rooted directed acyclic graph with root $r \in \mathcal{V}(G)$ (i.e., there is a directed path from r to any other node),
- $\rho\colon \mathcal{L}(G) \to \mathbb{R}$ labels sinks with real numbers,
- $\chi\colon \mathcal{V}(G) \setminus \mathcal{L}(G) \to X$ labels other nodes with variable names,
- and $\epsilon\colon \mathcal{E}(G) \to \{0, 1\}$ labels edges.

Moreover, the following properties must be satisfied.

- Every node has outdegree either zero or two. In the latter case, the two outgoing edges $e, f \in \mathcal{E}(G)$ are such that $\epsilon(e) = 1$, and $\epsilon(f) = 0$. If $e = (v, u)$, and $f = (v, w)$ for some $u, v, w \in \mathcal{V}(G)$, then u is the *positive successor* of v, and w is the *negative successor*.
- For every directed path with node sequence $v_1, v_2, \ldots, v_n$ such that $v_i \notin \mathcal{L}(G)$ for all i, we have that $\sigma(\chi(v_i)) < \sigma(\chi(v_{i+1}))$ for all $i = 1, 2, \ldots, n-1$.

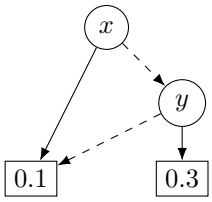


Figure 3: A graphical representation of the ADD from Example 1. Sinks are represented as rectangles and other nodes as circles. The labels of nodes are written directly on them. An edge e is dashed if $\epsilon(e) = 0$ and solid otherwise.

We say that an ADD *has* variable $x \in X$ if there is a node $v \in \mathcal{V}(G) \setminus \mathcal{L}(G)$ such that $\chi(v) = x$.

It remains to define how an ADD can be interpreted as a pseudo-Boolean function. Let $[\![\cdot]\!] : \mathcal{V}(G) \to \mathbb{R}^{2^X}$ be defined as

$$[\![v]\!] \coloneqq \begin{cases} \rho(v) & \text{if } v \in \mathcal{L}(G) \\ [\chi(v)]_{[\![w]\!]}^{[\![u]\!]} & \text{if } v \notin \mathcal{L}(G) \end{cases}$$

for all $v \in \mathcal{V}(G)$, where $u \in \mathcal{V}(G)$ and $w \in \mathcal{V}(G)$ are the positive and negative successors of v, respectively. Here, $[\![\cdot]\!]$ returns a pseudo-Boolean function $2^X \to \mathbb{R}$ for each node of G, and the interpretation of the ADD itself is defined to be $[\![r]\!]$. As before, $\rho(v)$ should be interpreted as a constant function.

Fact 1 ([Bahar et al., 1997])**.** *For any fixed set of variables X and ordering function $\sigma : X \to \mathbb{N}^+$, there is a unique (up to isomorphism)* canonical *ADD for every pseudo-Boolean function $2^Y \to S$ for all $Y \subseteq X$. Any ADD can be* reduced *to its canonical form in time linear in the number of nodes.*

Example 1. Let $f : 2^{\{x,y\}} \to \mathbb{R}$ be a pseudo-Boolean function defined as $f(\varnothing) = f(\{x\}) = f(\{x,y\}) = 0.1$, and $f(\{y\}) = 0.3$ and $\sigma : \{x,y\} \to \mathbb{N}^+$ be the variable ordering function defined as $\sigma(x) = 1$, and $\sigma(y) = 2$. Then the canonical ADD for f under σ is pictured in Section 4.1 and can be formally defined as $(G, a, \rho, \chi, \epsilon)$, where:

- $\mathcal{V}(G) = \{a, b, c, d\}$,
- $\mathcal{E}(G) = \{(a,b), (a,c), (b,c), (b,d)\}$,
- $\rho(c) = 0.1$, $\rho(d) = 0.3$,
- $\chi(a) = x$, $\chi(b) = y$,
- $\epsilon((a,c)) = \epsilon((b,d)) = 1$, $\epsilon((a,b)) = \epsilon((b,c)) = 0$.

Various operations can be defined on ADDs. We list the ones pertinent to our needs in Table 1: reduction of an ADD to its canonical form, addition/multiplication as well as scalar addition/multiplication, two types of *restrictions*, and *projection*. For a more detailed description, we refer the reader to previous work [Bahar et al., 1997, Dilkas and Belle, 2021a, Dudek et al., 2020a]. Throughout the paper, we assume that projection has the lowest precedence and extend the definition to allow for sets of variables. For any $W = \{w_1, w_2, \ldots, w_k\} \subseteq X$, let $\exists_W f : 2^{X \setminus W} \to \mathbb{R}$ be defined as $\exists_W f(Z) \coloneqq \exists_{w_1} \exists_{w_2} \cdots \exists_{w_k} f(Z)$ for all $Z \subseteq X \setminus W$, where the order of w_i's is immaterial. We assume that after every operation on ADDs, the resulting ADD is reduced to its canonical form and write 'the ADD for some function' to mean 'the *canonical* ADD'. In line with DPMC [Dudek et al., 2020b], we consider X and σ to be fixed throughout the execution of the algorithm, with X containing all relevant variables.

Fact 2. *The ADD for a pseudo-Boolean function $2^X \to \mathbb{R}$ has at most $2^{|X|+1}$ nodes. This upper bound is achieved when the function is injective.*

Table 1: Operations on ADDs, their definitions, the time complexity of the best-known algorithm for performing each operation, and references to the papers that introduced these algorithms. Let $f\colon 2^X \to \mathbb{R}$ and $g\colon 2^Y \to \mathbb{R}$ be pseudo-Boolean functions represented by ADDs with n and m nodes respectively, and let $r \in \mathbb{R}$, and $x \in X$.

Operation	Definition	Complexity	Source
reduce f		$\mathcal{O}(n)$	[Somenzi, 2015]
$r+f$, rf	$r+f\colon 2^X \to \mathbb{R}$, $(r+f)(Z) \coloneqq r + f(Z)$	$\mathcal{O}(n)$	[Bryant, 1986]
$f+g$, fg	$f+g\colon 2^{X\cup Y} \to \mathbb{R}$, $(f+g)(Z) \coloneqq f(Z) + g(Z)$	$\mathcal{O}(mn)$	[Bryant, 1986]
$f\vert_{x=0}$	$f\vert_{x=0}\colon 2^{X\setminus\{x\}} \to \mathbb{R}$, $f\vert_{x=0}(Z) \coloneqq f(Z)$	$\mathcal{O}(n)$	[Bryant, 1986]
$f\vert_{x=1}$	$f\vert_{x=1}\colon 2^X \to \mathbb{R}$, $f\vert_{x=1}(Z) \coloneqq f(Z \cup \{x\})$	$\mathcal{O}(n)$	[Bryant, 1986]
$\exists_x f$	$\exists_x f\colon 2^{X\setminus\{x\}} \to \mathbb{R}$, $\exists_x f(Z) \coloneqq f\vert_{x=0}(Z) + f\vert_{x=1}(Z)$	$\mathcal{O}(n^2)$	a corollary of other results

Lemma 1. *Let ϕ be a conjunction/disjunction of n literals. Then the ADD for $[\phi]_q^p$ (for any $p, q \in \mathbb{R}$ such that $p \neq q$) can be constructed in $\mathcal{O}(2^n)$ time.*

Proof. The ADD for $[\phi]_0^1$ can be constructed with a sequence of $n-1$ binary operations, with one of the two operands always the ADD representation of a literal. The number of variables in the other operand then follows the sequence $1, 2, 3, \ldots, n-1$. By Fact 2, the numbers of nodes in the ADDs for these operands is then $2^2, 2^3, \ldots, 2^n$. Since one of the operands is of constant size, the overall time complexity of all binary operations is then $\sum_{i=2}^{n} \mathcal{O}(2^i) = \mathcal{O}(2^n)$. Finally, note that $[\phi]_q^p = (p-q)[\phi]_0^1 + q$. As scalar operations can be performed in linear time, the overall complexity remains $\mathcal{O}(2^n)$. □

4.2 Main Result

We begin by compiling relevant results from previous work with some slight modifications.

Definition 3 ([Dudek et al., 2020b])**.** Let X be a set of variables and ϕ be a CNF formula over X. A *project-join tree* (PJT) of ϕ is a tuple (T, r, γ, π) where:

- T is a tree with root $r \in \mathcal{V}(T)$,
- $\gamma\colon \mathcal{L}(T) \to \phi$ is a bijection between the leaves of T and the clauses of ϕ, and
- $\pi\colon \mathcal{V}(T) \setminus \mathcal{L}(T) \to 2^X$ is a labelling function on internal nodes.

Moreover, the following properties must be satisfied.

- $\{\, \pi(t) : t \in \mathcal{V}(T) \setminus \mathcal{L}(T) \,\}$ is a partition of X, and
- for each internal node $t \in \mathcal{V}(T) \setminus \mathcal{L}(T)$, variable $x \in \pi(t)$, and clause $c \in \phi$ such that x appears in c, the leaf node $\gamma^{-1}(c)$ must be a descendant of t in T.

Let (T, r, γ, π) be a PJT. We can encapsulate the operations on ADDs performed during DPMC [Dudek et al., 2020b] execution by $\delta(r)$, where $\delta\colon \mathcal{V}(T) \to \mathbb{R}^{\mathrm{cod}(\pi)}$ is a recursive function defined as

$$\delta(t) \coloneqq \begin{cases} \gamma(t) & \text{if } t \in \mathcal{L}(T) \\ \exists_{\pi(t)} \prod_{u \in \mathcal{C}(t)} \delta(u) \prod_{x \in \pi(t)} [x]_{w(\neg x)}^{w(x)} & \text{if } t \notin \mathcal{L}(T).^{9} \end{cases} \tag{1}$$

The range of $\delta(r)$ then contains a single real number, i.e., the answer.

[9] This is a variation of Eq. (2) in the DPMC paper [Dudek et al., 2020b].

Definition 4 (The Width of a PJT [Dudek et al., 2020b])**.** Let (T, r, γ, π) be a PJT and $\texttt{Vars}\colon \mathcal{V}(T) \to \mathrm{cod}(\pi)$ and $\texttt{size}\colon \mathcal{V}(T) \to \mathbb{N}_0$ be two functions defined on the nodes of the PJT defined respectively as

$$\texttt{Vars}(t) \coloneqq \begin{cases} X & \text{if } t \in \mathcal{L}(T) \text{ and } \mathrm{dom}(\gamma(t)) = 2^X \\ \left(\bigcup_{u \in \mathcal{C}(t)} \texttt{Vars}(u)\right) \setminus \pi(t) & \text{if } t \notin \mathcal{L}(T) \end{cases}$$

and

$$\texttt{size}(t) \coloneqq \begin{cases} |\texttt{Vars}(t)| & \text{if } t \in \mathcal{L}(T) \\ |\texttt{Vars}(t) \cup \pi(t)| & \text{if } t \notin \mathcal{L}(T) \end{cases}$$

for any $t \in \mathcal{V}(T)$. Intuitively, $\texttt{size}(t)$ is the maximum number of variables that can appear during the computation of $\delta(t)$ (excluding recursive calls). The *width* of a PJT (T, r, γ, π) is then $\max_{t \in \mathcal{V}(T)} \texttt{size}(t)$.

We are now ready to state the parameterized complexity of DPMC [Dudek et al., 2020b].

Theorem 1. *Let ϕ be a CNF formula over a set of variables X and (T, r, γ, π) be its PJT of width w. Then* DPMC *[Dudek et al., 2020b] runs in time $\mathcal{O}(4^w mn)$, where $m = |\mathcal{L}(T)| = |\phi|$ is the number of clauses, and $n = |X|$ is the number of variables.*

Proof. We consider the complexity of constructing ADD representations of the clauses in ϕ and the complexity of multiplication and projection operations throughout the recursive calls of δ from Eq. (1). Let $t \in \mathcal{L}(T)$ be a leaf. Note that $\texttt{size}(t) \leq w$ by Definition 4. Assuming that clauses have no repeating variables, it takes $\mathcal{O}(2^w)$ time to construct $\delta(t)$ by Lemma 1 and $\mathcal{O}(2^w m)$ time to construct all leaves of the PJT.

Now let $t \in \mathcal{V}(T) \setminus \mathcal{L}(T)$ be an internal node. Here we multiply $|\mathcal{C}(t)|$ ADDs from recursive calls with $|\pi(t)|$ constant-sized ADDs that hold the relevant weights and project the variables in $\pi(t)$. We assume that $\prod_{u \in \mathcal{C}(t)} \delta(u)$ is computed first, and then multiplied by each $[x]^{w(x)}_{w(\neg x)}$ for all $x \in \pi(t)$. Note that $|\mathcal{C}(t)| \leq |\mathcal{L}(T)| = m$, $|\pi(t)| \leq w$, and the ADDs involved can have up to w variables. This means that each ADD has $\mathcal{O}(2^w)$ nodes regardless of whether it comes from $\mathcal{C}(t)$ or from multiplications.

Each multiplication of ADDs from recursive calls takes $\mathcal{O}(4^w)$ time by Table 1, and so all such multiplications will take $\mathcal{O}(4^w m)$ time for t and $\mathcal{O}(4^w mn)$ time across all $t \in \mathcal{V}(T) \setminus \mathcal{L}(T)$ since the number of internal nodes in the PJT is bounded by the number of variables.

Each multiplication of a constant-sized ADD and an ADD with $\mathcal{O}(2^w)$ nodes will take $\mathcal{O}(2^w)$ time, and so $|\pi(t)|$ of them will take $\mathcal{O}(2^w w)$ time. Summing this across all internal nodes of the PJT gives $\mathcal{O}(2^w wn)$.

Each variable is projected exactly once and is always projected from an ADD with at most $\mathcal{O}(2^w)$ nodes. Thus, the time complexity of projecting all variables is $\mathcal{O}(4^w n)$. Overall, we get $\mathcal{O}(2^w m)$ time for constructing initial ADDs, $\mathcal{O}(4^w mn)$ and $\mathcal{O}(2^w wn)$ for the two types of multiplications, and $\mathcal{O}(4^w n)$ for projections, resulting in $\mathcal{O}(4^w mn)$ time in total. □

The PJT of a formula is (efficiently) constructed from an approximately-minimal-width tree decomposition of its primal graph. Moreover, Dudek et al. [Dudek et al., 2020b] show that if the tree decomposition has width w, then the PJT will have width at most $w + 1$. This allows us to restate Theorem 1 without referring to PJTs.

Corollary 1. *Given a* WMC *instance with a CNF formula ϕ with a tree decomposition of its primal graph of width w,* DPMC *[Dudek et al., 2020b] runs in time $\mathcal{O}(4^w mn)$, where m is the number of clauses, and n is the number of variables.*

Since the 4^w part is tight (because the multiplication of two ADDs with m and n nodes is $\Theta(mn)$ when the corresponding pseudo-Boolean functions are injective, and all possible products of their values are different), we can conjecture that DPMC [Dudek et al., 2020b] performs worse on high-primal-treewidth instances compared to other algorithms such as C2D [Darwiche, 2004] which scales at most as 2^w w.r.t. primal treewidth. Section 6 tests this conjecture on randomly generated WMC instances.

5 Random k-CNF Formulas with Varying Primal Treewidth

Let S be a finite set. We write $\mathcal{U}S$ for the discrete uniform probability distribution on S. We represent any other probability distribution as a pair (S, p) where $p{:}\, S \to [0,1]$ 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 $\mathcal{P}$.

Our random k-CNF model is based on the following parameters:

- the number of variables $\nu \in \mathbb{N}^+$,
- (clause) *density* $\mu \in \mathbb{R}_{>0}$ (i.e., the ratio of the number of clauses and the number of variables),
- clause width $\kappa \in \mathbb{N}^+$ (for k-CNF formulas, $\kappa = k$),
- a parameter $\rho \in [0,1]$ that influences the primal treewidth of the formula,
- the proportion $\delta \in [0,1]$ of variables x such that $w(x) = 1$ and $w(\neg x) = 0$ or $w(x) = 0$ and $w(\neg x) = 1$,
- and the proportion $\epsilon \in [0, 1-\delta]$ of variables x such that $w(x) = w(\neg x) = 0.5$.

Remark. Note that not having a parameter that in some way influences primal treewidth is unrealistic for generating instances with a wide range of primal treewidth values. Without such a parameter (i.e., if $\rho = 0$), the variance of primal treewidth is relatively small compared to the range of values one could generate by varying ρ between zero and one (evidence for this is presented in Section 5.1).

Remark. The parameters δ and ϵ that control the numerical values of weights are part of the model because the running time of DPMC [Dudek et al., 2020b] (and other algorithms based on ADDs) depends on these numerical values. Indeed, the running time depends on the numbers of nodes in various ADDs, and this depends on the range of each pseudo-Boolean function. If all weights are different real numbers in $(0,1)$, then performing addition and multiplication on them is unlikely to result in any duplicates, and the numbers of nodes in ADDs is maximised. But weights such as zero and one are particularly 'simplifying' because they are respectively the additive and multiplicative identities. Including many copies of the same weight (such as 0.5) can be somewhat simplifying as well.

Algorithm 1: Generating a random k-CNF formula

Data: $\nu, \kappa \in \mathbb{N}^+$ such that $\kappa < \nu$, $\mu \in \mathbb{R}_{>0}$, $\rho \in [0,1]$.
Result: A k-CNF formula ϕ.

$\phi \leftarrow$ empty CNF formula;
$G \leftarrow$ empty graph;
for $i \leftarrow 1$ **to** $\lfloor \nu\mu \rfloor$ **do**
  $X \leftarrow \varnothing$;
  for $j \leftarrow 1$ **to** κ **do**
    $x \leftarrow$ `NewVariable`(X, G);
    $\mathcal{V}(G) \leftarrow \mathcal{V}(G) \cup \{ x \}$;
    $\mathcal{E}(G) \leftarrow \mathcal{E}(G) \cup \{ \{ x, y \} \mid y \in X \}$;
    $X \leftarrow X \cup \{ x \}$;
  $\phi \leftarrow \phi \cup \{ l \leftsquigarrow \mathcal{U}\{ x, \neg x \} \mid x \in X \}$;
return ϕ;
Function `NewVariable`(X, G)**:**
  $N \leftarrow \{ e \in \mathcal{E}(G) \mid |e \cap X| = 1 \}$;
  if $N = \varnothing$ **then return** $x \leftsquigarrow \mathcal{U}(\{ x_1, x_2, \ldots, x_\nu \} \setminus X)$;
  return $x \leftsquigarrow \left(\{ x_1, x_2, \ldots, x_\nu \} \setminus X, y \mapsto \frac{1-\rho}{\nu - |X|} + \rho \frac{|\{ z \in X \mid \{ y, z \} \in \mathcal{E}(G) \}|}{|N|} \right)$;

The process behind generating random k-CNF formulas is summarised as Algorithm 1. For the rest of this section, let $x_1, x_2, \ldots, x_\nu$ be the variables of the formula that is being generated. We simultaneously constructs both formula ϕ and its primal graph G. Each execution 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 to 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 that will be added to G by Line 8 will form a subset of N. If N is empty, 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 the smallest number of new edges to G. Note that when $\rho = 0$, Algorithm 1 reduces to the standard random model for k-CNF formulas where each clause has no duplicate variables. Also note that on Line 15 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 Experiment 1 and Fig. 4) although alternatives (e.g., exponential scaling) could be considered.

To transform the generated k-CNF formula into a WMC instance, we need to define weights on literals.[10] 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 $\{1, 2, \ldots, \nu\}$), and assign to each *variable* x_n a weight drawn u.a.r. from

- $\mathcal{U}\{0, 1\}$ if $\pi(n) \leq \nu\delta$,
- $\mathcal{U}\{0.5\}$ if $\nu\delta < \pi(n) \leq \nu\delta + \nu\epsilon$,
- and $\mathcal{U}\{0.01, 0.02, \ldots, 0.99\}$[11] 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 2. 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 $\lfloor \nu\mu \rfloor = 3$, Algorithm 1 will generate a 3-CNF formula with three clauses.

For the first variable of the first clause, we are choosing u.a.r. from $\{x_1, x_2, \ldots, x_5\}$. Suppose the algorithm chooses x_5. The 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. The 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., $\neg x_5 \vee x_2 \vee x_1$.

The first variable of the second clause is chosen u.a.r. from $\{x_1, x_2, \ldots, x_5\}$.[12] 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

$$\Pr(x_3) = \Pr(x_4) = \frac{1 - 0.3}{5 - 1} = 0.175.$$

[10] Note that algorithms such as DPMC [Dudek et al., 2020b] and ADDMC [Dudek et al., 2020a] support a more flexible way of assigning weights that can lead to significant performance improvements [Dilkas and Belle, 2021a, Dilkas and Belle, 2021b].

[11] The interval $(0, 1)$ is represented as 99 discrete values purely for convenience.

[12] Note that the first variable of any clause is always chosen this way.

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., $L = (x_4, x_3, x_2, x_1, x_5)$, i.e., the permutation π is

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 2 & 1 & 5 \end{pmatrix}$$

which 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 0.5 weight, and the remaining two variables get weights u.a.r. from $\{0.01, 0.02, \ldots, 0.99\}$. The weight function $w{:}\{x_1, x_2, \ldots, x_5, \neg x_1, \neg x_2, \ldots, \neg x_5\} \to [0, 1]$ 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 $w(\neg x_5) = 0.82$.

5.1 Validating the Model

In this section, we briefly discuss the following two questions.

- Are instances generated with higher values of ρ less likely to be satisfiable?
- Does increasing the value of ρ reduce the primal treewidth of the generated instances?

To answer these questions we run the following experiment.

Experiment 1. Fix $\nu = 100, \delta = \epsilon = 0$, and consider random instances with $\mu = 2.5{\times}\sqrt{2}^{-5}, 2.5{\times}\sqrt{2}^{-4}, \ldots, 2.5{\times}\sqrt{2}^{5}$, $\kappa = 2, 3, 45$, and ρ from 0 to 1 in steps of 0.01. For each combination of parameters, we generate ten instances. We check if each instance is satisfiable using MiniSat[13] [Eén and Sörensson, 2003] and calculate its (approximate) primal treewidth using htd[14] [Abseher et al., 2017].

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 random model. To that end, we need to select parameter values such that the proportion of satisfiable instances is sufficiently high. It is well-known that there is a sharp transition from satisfiable to unsatisfiable 3-CNF formulas at $\mu = 4.26$ regardless of the value of ν (this is known as the *satisfiability threshold*) [Crawford and Auton, 1996]. Also, the values of δ and ϵ clearly have no effect on the satisfiability of generated instances, so we only need to check if higher values of ρ make more instances unsatisfiable. The data from Experiment 1 shows that 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.

Figure 4 shows the relationship between ρ and primal treewidth. Except for when both μ and κ are set to very low values (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 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 perfect for generating 3-CNF formulas close to and below the satisfiability threshold.

6 Experimental Results

All experiments were run on Intel Xeon E5-2630 with Scientific Linux 7, GCC 10.2.0, Python 3.8.1, and R 4.1.0. 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. While

[13] http://minisat.se/MiniSat.html
[14] https://github.com/mabseher/htd

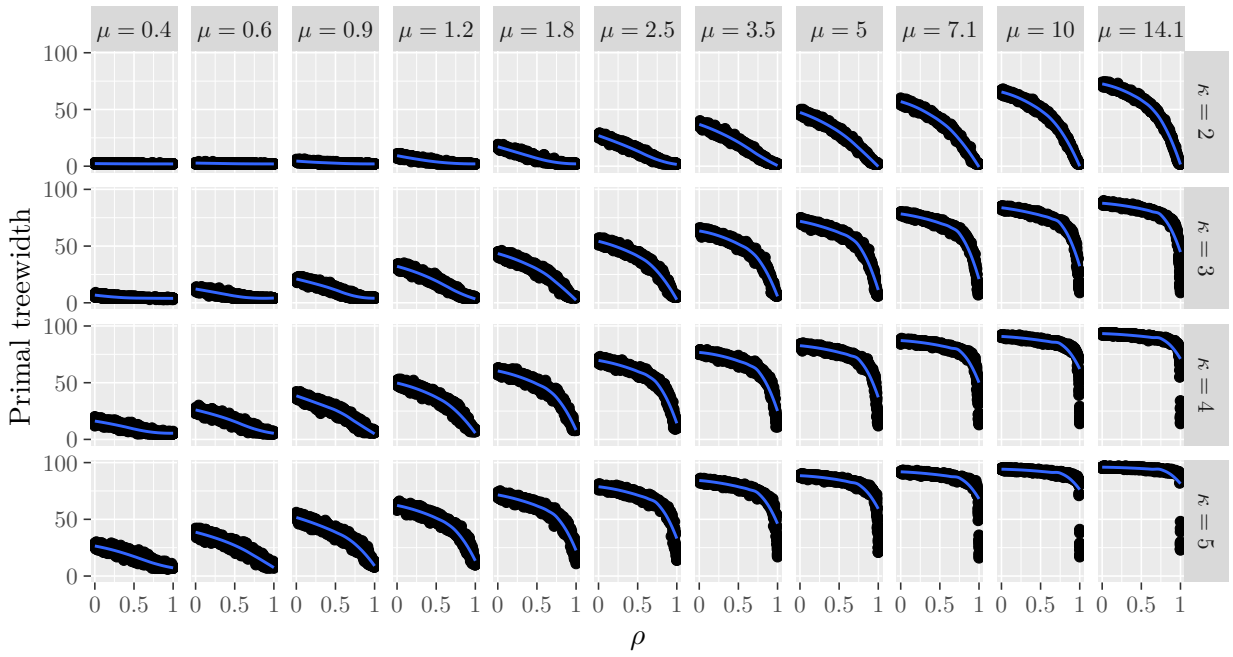


Figure 4: The relationship between ρ and primal treewidth for various values of μ and κ for k-CNF formulas from Experiment 1. Black points represent individual instances, and blue lines are smoothed means computed using locally weighted smoothing. The values of μ are rounded to one decimal place.

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 regions 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 $\nu = 70$, $\mu = 2.2$ (the peak for DPMC [Dudek et al., 2020b] according to Experiment 2), $\rho = 0$, 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, this percentage was higher in Experiments 3 and 4—10 and 12 %, respectively. D4 [Lagniez and Marquis, 2017] and C2D [Darwiche, 2004] 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.

In Experiment 2, we investigate how the running time of each algorithm depends on density and primal treewidth by varying both μ and ρ. The results are plotted in Fig. 5. The first thing to note is that peak hardness w.r.t. density occurs at around 1.9 for all algorithms except for DPMC [Dudek et al., 2020b] which peaks at 2.2 instead. This is consistent with previous work which shows CACHET to peak at 1.8 [Sang et al., 2004].[15]

The other major question we want to investigate using this experiment is how each algorithm scales w.r.t. primal treewidth. The two plots at the bottom of Fig. 5 show this relationship for fixed values

[15]For comparison, #SAT algorithms have been observed to peak at densities 1.2 and 1.5 [Jr. and Pehoushek, 2000].

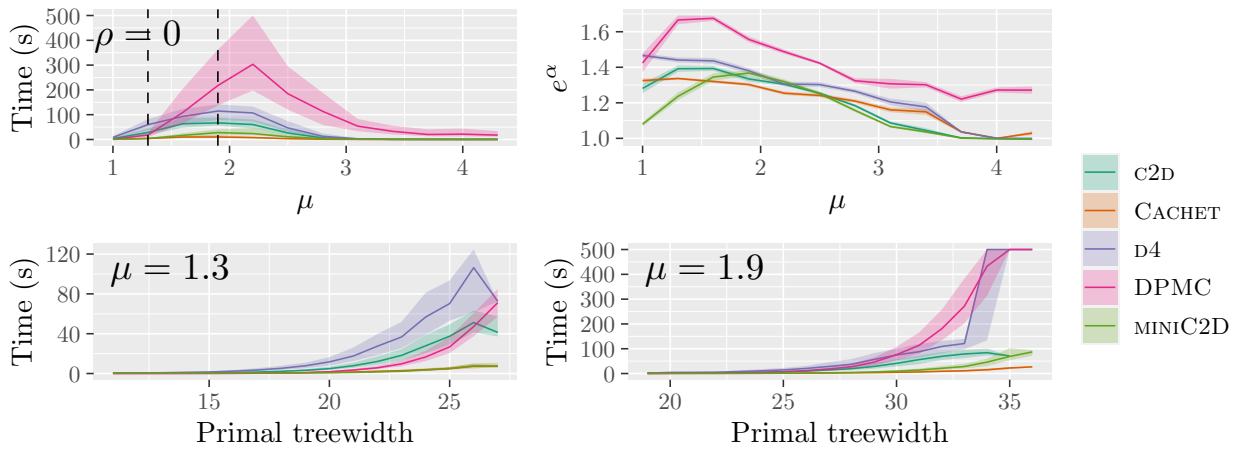


Figure 5: Visualisations of the data from Experiment 2. The top-left plot shows how the running time of each algorithm changes w.r.t. density when $\rho = 0$. For each algorithm and value of μ, each line in the top-right plot shows the estimated base of the exponential for a linear model where runtime is assumed to scale exponentially w.r.t. primal treewidth, and shaded regions show standard error. The two plots at the bottom show changes in the running time of each algorithm w.r.t. primal treewidth for selected fixed values of μ. These selected values are also marked by dashed vertical lines in the top-left plot.

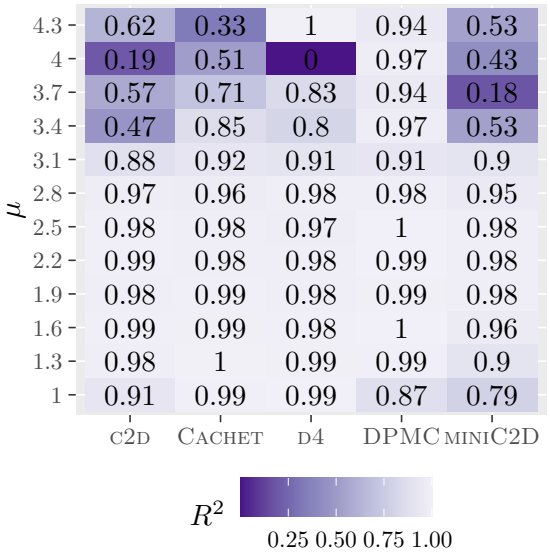


Figure 6: The coefficients of determination (rounded to one decimal place) of all the linear models fitted for the top-right subplot of Fig. 5

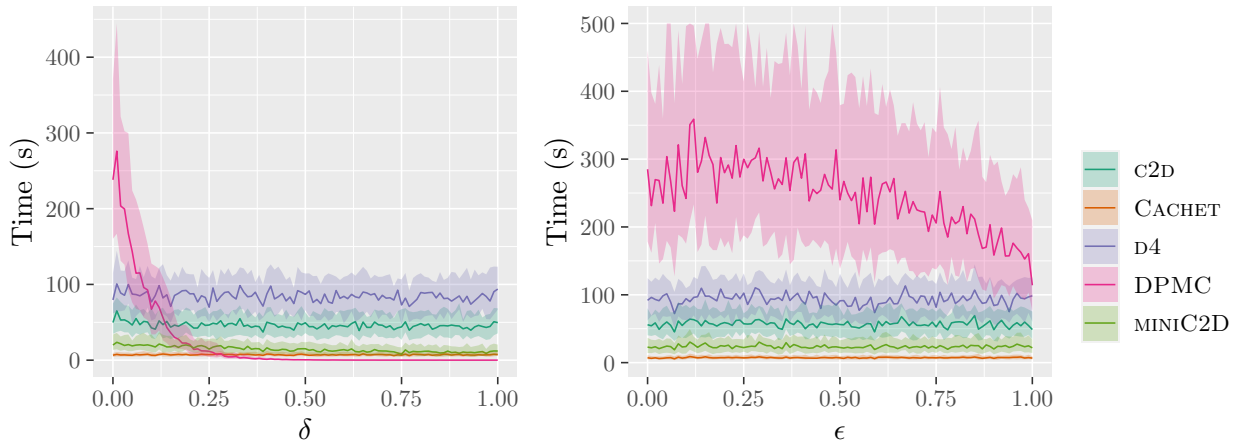


Figure 7: Changes in the running time of each algorithm as a result of changing δ (on the left-hand-side) and ϵ (on the right-hand-side) according to the data from Experiments 3 and 4

of μ, and one can see some evidence that the running time of DPMC [Dudek et al., 2020b] grows faster w.r.t. primal treewidth than the running time of the other algorithms, although the difference is not very significant. The top-right subplot of Fig. 5 shows how this growth depends on μ. For each algorithm and value of μ in Experiment 2, we select the median runtime for all available values of primal treewidth and fit the model $\ln t \sim \alpha w + \beta$, where t is the running time of the algorithm, w is the primal treewidth, and α and β are parameters.[16] Indeed, 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 Fig. 5 suggests an explanation. The running times of all algorithms except for DPMC approach zero when $\mu > 3$ while the median running time of DPMC approaches a small non-zero constant instead. This also explains why Fig. 6 shows that the fitted models fail to explain the data for non-ADD algorithms running on high-density instances—the running times 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 [Oztok and Darwiche, 2015] is the only algorithm that exhibits a clear low-high-low pattern in the top right subplot of Fig. 5. To a smaller extent, the same may apply to c2d [Darwiche, 2004] and DPMC [Dudek et al., 2020b] as well, although the evidence for this is limited due to relatively large gaps between different values of μ in Experiment 2. In contrast, the running times of Cachet [Sang et al., 2004] and d4 [Lagniez and Marquis, 2017] remain dependent on primal treewidth even when the density of the WMC instance is very low. This suggests that miniC2D should have an advantage on low-density high-primal-treewidth instances.

To sum, while DPMC [Dudek et al., 2020b] performed much worse than the other algorithms in Experiment 2, it can surpass c2d [Darwiche, 2004] and d4 [Lagniez and Marquis, 2017] on lower-density instances, particularly if some structure or redundancy in the instance results in lower primal treewidth. On the other hand, miniC2D [Oztok and Darwiche, 2015] and, especially, Cachet [Sang et al., 2004] appear to be exceptionally good at handling random instances. Finally, note that the exponential relationship between runtime and primal treewidth is worse in theory than in practice across all algorithms. In particular, the highest base of the exponential across all values of μ (i.e., the peak value in the top-right subplot of Fig. 5) is 1.47 for c2d and 1.75 for DPMC—both numbers are lower than their respective upper bounds of two and four.

Experiments 3 and 4 investigate how changing the numerical values of weights can simplify a WMC instance. The results are plotted in Fig. 7. As expected, the running time of all algorithms other than

[16]Similar statistical analyses have been used to investigate polynomial-to-exponential phase transitions in SAT [Coarfa et al., 2003].

DPMC [Dudek et al., 2020b] stay the same regardless of the value of δ or ϵ. The running time 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.

How are these random instances different from real data? As a small but representative sample, we take the WMC encodings of Bayesian networks created using the method by Sang et al. [Sang et al., 2005] as found in the experimental setup[17] of the DPMC paper [Dudek et al., 2020b]. A typical real WMC instance has $\nu = 200$ variables, half of which have equal weights (i.e., $\epsilon = 0.5$), an average clause width of $\kappa = 2.6$, a density of $\mu = 2.5$, and a primal treewidth of 28. Our random instances have fewer variables and (for the most part) lower density. Another important difference is that our instances are in k-CNF whereas a typical encoding of a Bayesian network has many two-literal clauses mixed with clauses of various longer widths. Despite real instances having more variables, their primal treewidth is rather low. Perhaps this partially explains why the performance of DPMC is in line with the performance of all other algorithms on traditionally-used benchmarks [Dudek et al., 2020b] despite struggling with most of our random data.

7 Related Work on Random CNF Formulas

TODO: 'flip a coin' is too informal.

- The main random model [Franco and Paull, 1983, Jr. and Brown, 1983].
- [Gent and Walsh, 1994] suggest two models. First, clause widths follow a probability distribution with finite support. Second, for each clause and for each variable, we flip a biased coin for whether to include the variable in the clause or not (empty and unit clauses can be filtered out) (this is called the constant probability model).
- One of the earliest papers that use the constant probability model [Mitchell et al., 1992, Goldberg et al., 1982]. There are two variations: when contradictory literals are allowed (i.e., both have probability p of being included) and when they're not (i.e., we have probability p of the positive literal being included, probability p of the negative literal being included, and probability $1 - 2p$ of neither).
- Random instances are typically seen as too easy for SAT [Cha and Iwama, 1995] (this is also the reference for planted SAT) (and this paper claims they're too hard [Achlioptas and Moore, 2002]), but (at least for some algorithms), according to my experiments, they are too hard for WMC (most likely because of all the redundancy that WMC instances often have).
- Random CNF generators that generate instances with exactly one solution or ensure (un)satisfiability over a wide range of densities [Asahiro et al., 1993].
- [Dudek et al., 2017] consider random k-CNF-XOR formulas. They fix the number of variables, the densities of regular and XOR clauses, the number of (distinct-variable) literals per regular clause, and the probability that a variable is included in a XOR clause. XOR clauses are constructed by flipping a biased coin for each variable and flipping a fair coin for whether to add one or zero to the XOR clause.
- [Hossain et al., 2010] use adversarial evolution to generate hard 3-SAT instances such that different variables have different probabilities of appearing in a clause.
- [Coja-Oghlan and Wormald, 2018] consider random *regular* k-CNF instances where each variables appears d times as a positive literal and d times as a negative literal for some positive integer d.
- The number of (3-CNF) clauses varies, i.e., a biased coined is flipped for each possible clause. A possible extension: clauses that falsify a fixed truth assignment are forbidden [Atserias, 2005] (i.e., have a planted solution). In contrast to planted instances, *filtered* means that satisfiability is checked at the end (this is what we do; maybe I should mention this in the algorithm).

[17] https://github.com/vardigroup/DPMC/releases

- Another variation is to forbid multiple clauses with exactly the same variables and to require each clause to have at least c negative literals for some positive constant c (this is for k-CNF) [Gao, 2009].
- A combination of 2-clauses and 3-clauses [Achlioptas and Menchaca-Mendez, 2012].
- Two random models for k-CNF that strive to achieve a fixed *impurity* value (i.e., another alternative variable for phase transition) [Lozinskii, 2006].
- A random model for non-k-CNF instances where both variable frequencies and clause widths follow power law distributions [Giráldez-Cru and Levy, 2017]. They also observe that the satisfiability threshold is less sharp, and the instances (by the simpler version of the model) are trivial to SAT solvers even on the threshold.
- Random models for both 3-CNF and non-k-CNF formulas that attempt to be industrial-like by manipulating the probability distribution from which variables are sampled and the probability distribution used to assign each literal to a clause [Ansótegui et al., 2009].
- Random 3-CNF with community structure (as measured by modularity) [Giráldez-Cru and Levy, 2016]: we flip a biased coin for whether all variables in a clause come from the same community or all from different communities.
- Modularity is not the same as treewidth because communities can be interconnected in arbitrary ways (e.g., can be a clique) whereas a tree decomposition must be a tree.
- Skewed random k-SAT: positive and negative literals appear with different probabilities [Sinopalnikov, 2004].
- Random 3-CNF that's constructed in two steps: create a set number of smaller 3-CNF formulas, and then add clauses that connect them [Slater, 2002].

Phase transitions.

- Easy-hard-easy-type phase transitions vary from (SAT) solver to solver and are not necessarily centred around the satisfiability threshold [Coarfa et al., 2003].
- Phase transition for evolutionary algorithms [Doerr et al., 2017].
- For a fixed density, the proportion of clauses that have at most one negative literal produces another phase transition [van Maaren and van Norden, 2005].
- [Bläsius et al., 2019] also show medians and how peak hardness shifts according to the scale-free 3-SAT parameter.
- Structural entropy as an alternative parameter for phase transition [Lin et al., 2021].

8 Conclusion

- Should (unweighted) model count be a parameter? Do we expect some instances to be better with a bigger model count?
- Future work: can we formally establish how treewidth depends on ρ? A theoretical explanation for Figure 3 would be very interesting. Can I provide guarantees about the probability distribution of primal treewidth, e.g., how it scales w.r.t. ρ?
- Could (pseudo-)Boolean function sensitivity feature in the complexity of ADD representation ('On the Average Sensitivity and Density of k-CNF Formulas')?

- Additional experiment: would CACHET be worse if I added some redundant structure, e.g., new variables that are defined to be equivalent to some conjunctions?
- k-CNF is not very representative of typical WMC instances.
- Consider changing the probability dependence from linear to some kind of exponential or power law.
- The observation that the running time of DPMC [Dudek et al., 2020b] drops sharply with increasing δ suggests that an approach that uses DPMC to compute the WMC of instances simplified by a small number of CACHET iterations could be successful.
- Maybe mention that the observations about DPMC (in particular, the experiments on changing weight values) might be somewhat applicable to other applications of ADDs.
- Summarise the implications of the experimental results.
- Acknowledgment: ECDF was used.
- Acknowledge Florent Capelli for his awesome answer on StackExchange.
- Another thing one could do: compare my exponential curves with all kinds of polynomial curves to establish that the relationship is indeed exponential.
- A possible explanation for the difference between 1.75 and 4 is that ρ affects more than just primal treewidth.
- An interesting avenue for future work: a WMC algorithm that's FPT w.r.t. consensus treewidth.
- FPT of WFOMC? Maybe too much logic stuff.
- Kernelization for WMC.

References

[Abseher et al., 2017] Abseher, M., Musliu, N., and Woltran, S. (2017). htd - A free, open-source framework for (customized) tree decompositions and beyond. In Salvagnin, D. and Lombardi, M., editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 376–386. Springer.

[Achlioptas and Menchaca-Mendez, 2012] Achlioptas, D. and Menchaca-Mendez, R. (2012). Exponential lower bounds for DPLL algorithms on satisfiable random 3-cnf formulas. In Cimatti, A. and Sebastiani, R., editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 327–340. Springer.

[Achlioptas and Moore, 2002] Achlioptas, D. and Moore, C. (2002). 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*, pages 779–788. IEEE Computer Society.

[Ansótegui et al., 2009] Ansótegui, C., Bonet, M. L., and Levy, J. (2009). Towards industrial-like random SAT instances. In Boutilier, C., editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 387–392.

[Asahiro et al., 1993] Asahiro, Y., Iwama, K., and Miyano, E. (1993). Random generation of test instances with controlled attributes. In Johnson, D. S. and Trick, M. A., editors, *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 377–393. DIMACS/AMS.

[Atserias, 2005] Atserias, A. (2005). Definability on a random 3-cnf formula. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 458–466. IEEE Computer Society.

[Bacchus et al., 2009] Bacchus, F., Dalmao, S., and Pitassi, T. (2009). Solving #sat and bayesian inference with backtracking search. *J. Artif. Intell. Res.*, 34:391–442.

[Bahar et al., 1997] Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., and Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Formal Methods Syst. Des.*, 10(2/3):171–206.

[Bläsius et al., 2019] Bläsius, T., Friedrich, T., and Sutton, A. M. (2019). On the empirical time complexity of scale-free 3-sat at the phase transition. In Vojnar, T. and Zhang, L., editors, *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*, volume 11427 of *Lecture Notes in Computer Science*, pages 117–134. Springer.

[Bryant, 1986] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691.

[Cha and Iwama, 1995] Cha, B. and Iwama, K. (1995). Performance test of local search algorithms using new types of random CNF formulas. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 304–311. Morgan Kaufmann.

[Chakraborty et al., 2015] Chakraborty, S., Fried, D., Meel, K. S., and Vardi, M. Y. (2015). From weighted to unweighted model counting. In Yang, Q. and Wooldridge, M. J., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 689–695. AAAI Press.

[Chavira and Darwiche, 2008] Chavira, M. and Darwiche, A. (2008). On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799.

[Coarfa et al., 2003] Coarfa, C., Demopoulos, D. D., Aguirre, A. S. M., Subramanian, D., and Vardi, M. Y. (2003). Random 3-sat: The plot thickens. *Constraints An Int. J.*, 8(3):243–261.

[Coja-Oghlan and Wormald, 2018] Coja-Oghlan, A. and Wormald, N. (2018). The number of satisfying assignments of random regular k-sat formulas. *Comb. Probab. Comput.*, 27(4):496–530.

[Courcelle et al., 2001] Courcelle, B., Makowsky, J. A., and Rotics, U. (2001). On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discret. Appl. Math.*, 108(1-2):23–52.

[Crawford and Auton, 1996] Crawford, J. M. and Auton, L. D. (1996). Experimental results on the crossover point in random 3-sat. *Artif. Intell.*, 81(1-2):31–57.

[Dal et al., 2018] Dal, G. H., Laarman, A. W., and Lucas, P. J. F. (2018). Parallel probabilistic inference by weighted model counting. In Studený, M. and Kratochvíl, V., editors, *International Conference on Probabilistic Graphical Models, PGM 2018, 11-14 September 2018, Prague, Czech Republic*, volume 72 of *Proceedings of Machine Learning Research*, pages 97–108. PMLR.

[Darwiche, 1999] Darwiche, A. (1999). Compiling knowledge into decomposable negation normal form. In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 284–289. Morgan Kaufmann.

[Darwiche, 2001a] Darwiche, A. (2001a). Decomposable negation normal form. *J. ACM*, 48(4):608–647.

[Darwiche, 2001b] Darwiche, A. (2001b). 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.

[Darwiche, 2004] Darwiche, A. (2004). New advances in compiling CNF into decomposable negation normal form. In de Mántaras, R. L. and Saitta, L., editors, *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*, pages 328–332. IOS Press.

[Darwiche, 2011] Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In Walsh, T., editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 819–826. IJCAI/AAAI.

[Davis et al., 1962] Davis, M., Logemann, G., and Loveland, D. W. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

[Davis and Putnam, 1960] Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *J. ACM*, 7(3):201–215.

[Dilkas and Belle, 2020] Dilkas, P. and Belle, V. (2020). Generating random logic programs using constraint programming. In Simonis, H., editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 828–845. Springer.

[Dilkas and Belle, 2021a] Dilkas, P. and Belle, V. (2021a). Weighted model counting with conditional weights for Bayesian networks. In de Campos, C. and Maathuis, M., editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2021, virtual online, July 27-30, 2021*, Proceedings of Machine Learning Research. AUAI Press.

[Dilkas and Belle, 2021b] Dilkas, P. and Belle, V. (2021b). Weighted model counting without parameter variables. In Li, C. M. and Manyà, F., editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, Lecture Notes in Computer Science. Springer.

[Doerr et al., 2017] Doerr, B., Neumann, F., and Sutton, A. M. (2017). Time complexity analysis of evolutionary algorithms on random satisfiable k-cnf formulas. *Algorithmica*, 78(2):561–586.

[Downey and Fellows, 2013] Downey, R. G. and Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

[Dudek et al., 2017] Dudek, J. M., Meel, K. S., and Vardi, M. Y. (2017). The hard problems are almost everywhere for random CNF-XOR formulas. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 600–606. ijcai.org.

[Dudek et al., 2020a] Dudek, J. M., Phan, V., and Vardi, M. Y. (2020a). 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*, pages 1468–1476. AAAI Press.

[Dudek et al., 2020b] Dudek, J. M., Phan, V. H. N., and Vardi, M. Y. (2020b). DPMC: weighted model counting by dynamic programming on project-join trees. In Simonis, H., editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 211–230. Springer.

[Eén and Sörensson, 2003] Eén, N. and Sörensson, N. (2003). An extensible SAT-solver. In Giunchiglia, E. and Tacchella, A., editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer.

[Fargier and Marquis, 2006] Fargier, H. and Marquis, P. (2006). 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*, pages 42–47. AAAI Press.

[Fichte et al., 2018] Fichte, J. K., Hecher, M., Woltran, S., and Zisser, M. (2018). Weighted model counting on the GPU by exploiting small treewidth. In Azar, Y., Bast, H., and Herman, G., editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Franco and Paull, 1983] Franco, J. and Paull, M. C. (1983). Probabilistic analysis of the davis putnam procedure for solving the satisfiability problem. *Discret. Appl. Math.*, 5(1):77–87.

[Ganian and Szeider, 2021] Ganian, R. and Szeider, S. (2021). New width parameters for SAT and #sat. *Artif. Intell.*, 295:103460.

[Gao, 2009] Gao, Y. (2009). Data reductions, fixed parameter tractability, and random weighted d-cnf satisfiability. *Artif. Intell.*, 173(14):1343–1366.

[Gent and Walsh, 1994] Gent, I. P. and Walsh, T. (1994). The SAT phase transition. In Cohn, A. G., editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8-12, 1994*, pages 105–109. John Wiley and Sons, Chichester.

[Giráldez-Cru and Levy, 2016] Giráldez-Cru, J. and Levy, J. (2016). Generating SAT instances with community structure. *Artif. Intell.*, 238:119–134.

[Giráldez-Cru and Levy, 2017] Giráldez-Cru, J. and Levy, J. (2017). Locality in random SAT instances. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 638–644. ijcai.org.

[Goldberg et al., 1982] Goldberg, A., Jr., P. W. P., and Brown, C. A. (1982). Average time analyses of simplified davis-putnam procedures. *Inf. Process. Lett.*, 15(2):72–75.

[Hossain et al., 2010] Hossain, M. M., Abbass, H. A., Lokan, C., and Alam, S. (2010). 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*, pages 1–8. IEEE.

[Jr. and Brown, 1983] Jr., P. W. P. and Brown, C. A. (1983). An analysis of backtracking with search rearrangement. *SIAM J. Comput.*, 12(4):717–733.

[Jr. and Pehoushek, 2000] Jr., R. J. B. and Pehoushek, J. D. (2000). Counting models using connected components. In Kautz, H. A. and Porter, B. W., editors, *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*, pages 157–162. AAAI Press / The MIT Press.

[Lagniez and Marquis, 2017] Lagniez, J. and Marquis, P. (2017). An improved decision-DNNF compiler. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673. ijcai.org.

[Lin et al., 2021] Lin, Q., Wang, X., and Niu, J. (2021). A new method for 3-satisfiability problem phase transition on structural entropy. *IEEE Access*, 9:2093–2099.

[Lozinskii, 2006] Lozinskii, E. L. (2006). Impurity: Another phase transition of SAT. *J. Satisf. Boolean Model. Comput.*, 1(2):123–141.

[Mitchell et al., 1992] Mitchell, D. G., Selman, B., and Levesque, H. J. (1992). Hard and easy distributions of SAT problems. In Swartout, W. R., editor, *Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992*, pages 459–465. AAAI Press / The MIT Press.

[Oztok and Darwiche, 2015] Oztok, U. and Darwiche, A. (2015). A top-down compiler for sentential decision diagrams. In Yang, Q. and Wooldridge, M. J., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3141–3148. AAAI Press.

[Renkens et al., 2014] Renkens, J., Kimmig, A., den Broeck, G. V., and Raedt, L. D. (2014). Explanation-based approximate weighted model counting for probabilistic logics. In Brodley, C. E. and Stone, P., editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 2490–2496. AAAI Press.

[Riguzzi, 2020] Riguzzi, F. (2020). Quantum weighted model counting. In Giacomo, G. D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., and Lang, J., editors, *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)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2640–2647. IOS Press.

[Robertson and Seymour, 1984] Robertson, N. and Seymour, P. D. (1984). Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64.

[Robertson and Seymour, 1991] Robertson, N. and Seymour, P. D. (1991). Graph minors. x. obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190.

[Samer and Szeider, 2010] Samer, M. and Szeider, S. (2010). Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64.

[Sang et al., 2004] Sang, T., Bacchus, F., Beame, P., Kautz, H. A., and Pitassi, T. (2004). 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*.

[Sang et al., 2005] Sang, T., Beame, P., and Kautz, H. A. (2005). Performing Bayesian inference by weighted model counting. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 475–482. AAAI Press / The MIT Press.

[Sinopalnikov, 2004] Sinopalnikov, D. A. (2004). Satisfiability threshold of the skewed random k-sat. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*.

[Slater, 2002] Slater, A. (2002). Modelling more realistic SAT problems. In McKay, B. and Slaney, J. K., editors, *AI 2002: Advances in Artificial Intelligence, 15th Australian Joint Conference on Artificial Intelligence, Canberra, Australia, December 2-6, 2002, Proceedings*, volume 2557 of *Lecture Notes in Computer Science*, pages 591–602. Springer.

[Somenzi, 2015] Somenzi, F. (2015). CUDD: CU decision diagram package release 3.0.0. *University of Colorado at Boulder*.

[van Maaren and van Norden, 2005] van Maaren, H. and van Norden, L. (2005). Correlations between horn fractions, satisfiability and solver performance for fixed density random 3-cnf instances. *Ann. Math. Artif. Intell.*, 44(1-2):157–177.