Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

New Algorithms for #2-SAT and #3-SAT

Junqiang Peng[1] , **Zimo Sheng**[2] and **Mingyu Xiao**[1*]
[1]University of Electronic Science and Technology of China, Chengdu, China
[2]Anhui University
jqpeng0@foxmail.com, shengzimo2016@gmail.com, myxiao@uestc.edu.cn

Abstract

The #2-SAT and #3-SAT problems involve counting the number of satisfying assignments (also called models) for instances of 2-SAT and 3-SAT, respectively. In 2010, Zhou et al. proposed an $\mathcal{O}^*(1.1892^m)$-time algorithm for #2-SAT and an efficient approach for #3-SAT, where m denotes the number of clauses. In this paper, we show that the weighted versions of #2-SAT and #3-SAT can be solved in $\mathcal{O}^*(1.1082^m)$ and $\mathcal{O}^*(1.4423^m)$ time, respectively. These results directly apply to the unweighted cases and achieve substantial improvements over the previous results. These advancements are enabled by the introduction of novel reduction rules, a refined analysis of branching operations, and the application of path decompositions on the primal and dual graphs of the formula.

1 Introduction

The BOOLEAN SATISFIABILITY problem (SAT), the first problem to be proven NP-complete [Cook, 1971], is a cornerstone of computational complexity theory. Its counting variant, the PROPOSITIONAL MODEL COUNTING problem (#SAT), introduced and shown to be #P-complete by Valiant (1979), holds comparable significance.

Given a CNF formula, SAT seeks to determine whether the formula is satisfiable, while #SAT aims to count the number of satisfying assignments (also known as models). Both SAT and #SAT, along with their variants, are among the most influential problems in computational theory due to their broad applications in computer science, artificial intelligence, and numerous other domains, both theoretical and practical. As such, these problems have garnered substantial attention and have been extensively studied across various fields, including computational complexity and algorithm design. For comprehensive surveys, For comprehensive surveys, we refer to [Biere *et al.*, 2021; Fichte *et al.*, 2023a].

This paper focuses on #SAT and its weighted extension, WEIGHTED MODEL COUNTING (WMC, or weighted #SAT). In WMC, each literal (a variable or its negation) in the formula is assigned a weight. The task is to compute the sum of the weights of all satisfying assignments, where the weight of an assignment is the product of the weights of its literals. Notably, WMC reduces to #SAT when all literals have identical weights. Efficient algorithms for (weighted) #SAT have a profound impact on various application areas [Gomes *et al.*, 2021], such as probabilistic inference [Roth, 1996; Bacchus *et al.*, 2003; Sang *et al.*, 2005; Chavira and Darwiche, 2008], network reliability estimation [Dueñas-Osorio *et al.*, 2017], and explainable AI [Narodytska *et al.*, 2019]. This significance is highlighted by the annual Model Counting Competition[1], which bridges theoretical advancements and practical implementations in model counting.

A fundamental question is: how fast (weighted) #SAT can be solved in the worst case? The naïve algorithm, which enumerates all assignments, runs in $\mathcal{O}^*(2^n)$ time[2], where n is the number of variables in the formula. Under the Strong Exponential Time Hypothesis (SETH) [Impagliazzo and Paturi, 2001], SAT (and thus #SAT) cannot be solved in $\mathcal{O}^*((2-\epsilon)^n)$ time for any constant $\epsilon > 0$. Another key parameter, the number of clauses m in the formula, has also been extensively studied. It is well-known that #SAT can be solved in $\mathcal{O}^*(2^m)$ time using the Inclusion-Exclusion principle [Iwama, 1989; Lozinskii, 1992], which also applies to the weighted variant. However, no algorithm with a runtime of $\mathcal{O}^*(c^m)$ for $c < 2$ was discovered. In fact, Cygan *et al.* (2016) proved that such an algorithm does not exist unless SETH fails.

The barriers 2^n and 2^m can be broken for restricted versions of (weighted) #SAT. One notable example is the (weighted) #k-SAT problem, where each clause contains at most k literals. There is a rich history of faster algorithms for #2-SAT and #3-SAT [Dubois, 1991; Zhang, 1996; Dahllöf *et al.*, 2005; Kutzkov, 2007; Wahlström, 2008; Zhou *et al.*, 2010]. The current fastest algorithms for weighted #2-SAT and #3-SAT achieve runtimes of $\mathcal{O}^*(1.2377^n)$ [Wahlström, 2008] and $\mathcal{O}^*(1.6423^n)$ [Kutzkov, 2007], respectively. For the parameter m, Zhou *et al.* introduced an $\mathcal{O}^*(1.1740^m)$-time algorithm for #2-SAT and suggested a simple approach for #3-SAT. However, the analysis of the #3-SAT algorithm in [Zhou *et al.*, 2010] does not yield a valid runtime bound.

Our Contribution. We propose two novel algorithms, `Alg2CNF` and `Alg3CNF`, for weighted #2-SAT and

*Corresponding Author

[1]https://mccompetition.org/

[2]The $\mathcal{O}^*$ notation suppresses polynomial factors in the input size.

weighted #3-SAT, achieving runtime bounds of $\mathcal{O}^*(1.1082^m)$ and $\mathcal{O}^*(1.4423^m)$, respectively. The algorithms and complexity bounds directly apply to the unweighted case, significantly improving upon previous results. To this end, we bring new techniques for (weighted) #2-SAT and #3-SAT.

Our Approach. Most existing algorithms, including ours, are classical branch-and-search algorithms (also called DPLL-style algorithms) that first apply reduction (preprocessing) rules and then recursively solve the problem via branching (e.g., selecting a variable and assigning it values). Typically, variables are processed in descending order of their degree, where the *degree* of a variable is the number of its occurrences in the formula. However, such algorithms often perform poorly when encountering low-degree variables. To address this, previous work has employed tailored branching strategies with intricate analyses to mitigate this bottleneck.

Our approach departs from these complexities. Instead of elaborate branching, we apply path decompositions on the primal and dual graphs of the formula to efficiently handle low-degree cases. The primal and dual graphs represent structural relationships between variables and clauses, and path decompositions allow us to transform these structures into a path-like form. Although our algorithms rely on simple branching, we demonstrate through sophisticated analyses that this approach, combined with our reduction rules, achieves substantial efficiency. These ideas may hold potential for broader applications in the future.

Other Related Works. There is extensive research on fast algorithms for SAT and its related problems parameterized by m. We list the current best results for some of these problems, following multiple improvements. Chu *et al.* (2021) showed that SAT can be solved in $\mathcal{O}^*(1.2226^m)$ time. Beigel and Eppstein (2005) introduced an $\mathcal{O}^*(1.3645^m)$-time algorithm for 3-SAT. We also mention the MAXIMUM SATISFIABILITY problem (MaxSAT), an optimization version of SAT, where the objective is to satisfy the maximum number of clauses in a given formula. Currently, MaxSAT and Max-2-SAT can be solved in time $\mathcal{O}^*(1.2886^m)$ [Xiao, 2022] and $\mathcal{O}^*(1.1159^m)$ [Gaspers and Sorkin, 2009], respectively.

Proofs of lemmas marked with ♣ are deferred to the full version of the paper due to space limitations.

2 Preliminaries

2.1 Notations

A *Boolean variable* (or simply *variable*) can be assigned value 1 (TRUE) or 0 (FALSE). A variable x has two corresponding *literals*: the positive literal x and the negative literal $\bar{x}$. We use $\bar{x}$ to denote the negation of literal x, and thus $\bar{\bar{x}} = x$. Let V be a set of variables. A *clause* on V is a set of literals on V. Note that a clause might be empty. A *CNF formula* (or simply *formula*) over V is a set of clauses on V. We denote by $\mathrm{var}(F)$ the variable set of F. For a literal ℓ, $\mathrm{var}(\ell)$ denotes its corresponding variable. For a clause C, $\mathrm{var}(C)$ denotes the set of variables such that either $x \in C$ or $\bar{x} \in C$. We denote by $n(F)$ and $m(F)$ the number of variables and clauses in formula F, respectively.

An *assignment* for variable set V is a mapping $\sigma : V \to \{0, 1\}$. Given an assignment σ, a clause is *satisfied* by σ if at least one literal in it gets value 1 under σ. An assignment for $\mathrm{var}(F)$ is called a *model* of F if σ satisfies all clauses in F. We write $\sigma \models F$ to indicate that σ is a model of F.

Definition 1 (Weighted Model Count)**.** *Let F be a formula, $\mathrm{ltr}(F) := \bigcup_{x\in \mathrm{var}(F)}\{x, \bar{x}\}$ be the set of literals of variables in F, $w : \mathrm{ltr}(F) \to \mathbb{Z}^+$ be a* weight function *that assigns a (positive integer) weight value to each literal, and $\mathcal{A}(F)$ be the set of all possible assignments to $\mathrm{var}(F)$. The* weighted model count $\mathrm{WMC}(F, w)$ *of formula F is defined as*

$$\mathrm{WMC}(F, w) := \sum_{\substack{\sigma\in\mathcal{A}(F)\\ \sigma\models F}} \Big(\prod_{\substack{x\in\mathrm{var}(F)\\ \sigma(x)=1}} w(x) \cdot \prod_{\substack{y\in\mathrm{var}(F)\\ \sigma(y)=0}} w(\bar{y}) \Big).$$

In the Weighted Model Counting problem (WMC), given a formula F and a weight function w, the goal is to compute the weighted model count $\mathrm{WMC}(F, w)$ of formula F. We use weighted #2-SAT and weighted #3-SAT to denote the restricted versions of WMC where the inputs are 2-CNF and 3-CNF formulas, respectively.

A clause containing a single literal ℓ may be simply written as (ℓ). We use C_1C_2 to denote the clause obtained by concatenating clauses C_1 and C_2. For a formula F, we denote $F[\ell = 1]$ as the resulting formula obtained from F by removing all clauses containing literal ℓ and removing all literals $\bar{\ell}$ from all clauses in F.

The *degree* of a variable x in formula F, denoted by $\deg(x)$, is the total number of occurrences of literals x and $\bar{x}$ in F. A *d-variable* (resp., *d^+-variable*) is a variable with degree exactly d (resp., at least d). The degree of a formula F, denoted by $\deg(F)$, is the maximum degree of all variables in F. The *length* of a clause C is the number of literals in C. A clause is a *k-clause* (resp., *k^--clause*) if its length is exactly k (resp., at most k). A formula F is called *k-CNF formula* if each clause in F has length at most k.

We say a clause C *contains* a variable x if $x \in \mathrm{var}(C)$. Two variables x and y are *adjacent* (and *neighbors* of each other) if they appear together in some clause. We denote by $N(x, F)$ (resp., $N_i(x, F)$) the set of neighbors (resp., the set of i-degree neighbors) of variable x in formula F. When F is clear from the context, we may simply write $N(x)$ and $N_i(x)$.

2.2 Graph-related Concepts

The following two prominent graph representations of a CNF formula, namely the *primal graph* and the *dual graph*, will be used in our algorithms.

Definition 2 (Primal graphs)**.** *The* primal graph $G(F)$ *of a formula F is a graph where each vertex corresponds to a variable in the formula. Two vertices x and y are adjacent if and only if $x, y \in \mathrm{var}(C)$ for some clause $C \in F$.*

Definition 3 (Dual graphs)**.** *The* dual graph $G^d(F)$ *of a formula F is the graph where each vertex corresponds to a clause in the formula. Two vertices C_1 and C_2 are adjacent if and only if $\mathrm{var}(C_1) \cap \mathrm{var}(C_2) \neq \emptyset$ for $C_1, C_2 \in F$.*

We also use the concepts of *path decomposition*s, which offer a way to decompose a graph into a path structure.

Definition 4 (Path decompositions)**.** *A* path decomposition *of a graph G is a sequence $P = (X_1, \ldots, X_r)$ of vertex subsets*

$X_i \subseteq V(G)$ ($i \in \{1, \ldots, r\}$) such that: (1) $\bigcup_{i=1}^{r} X_i = V(G)$; (2) For every $uv \in E(G)$, there exists $l \in \{1, \ldots, r\}$ such that X_l contains both u and v; (3) For every $u \in V(G)$, if $u \in X_i \cap X_k$ for some $i \leq k$, then $u \in X_j$ for all $i \leq j \leq k$.

The *width* of a path decomposition $(X_1, \ldots, X_r)$ is defined as $\max_{1 \leq i \leq r}\{|X_i|\} - 1$. The *pathwidth* of a graph G, denoted by $\mathrm{pw}(G)$, is the minimum possible width of any path decomposition of G. The *primal pathwidth* and *dual pathwidth* of a formula are the pathwidths of its primal graph and dual graph, respectively.

The following is a known bound in terms of the pathwidth.

Theorem 1 ([Fomin *et al.*, 2009]). *For any $\epsilon > 0$, there exists an integer n_ϵ such that for every graph G with $n > n_\epsilon$ vertices,*

$$\mathrm{pw}(G) \leq n_3/6 + n_4/3 + n_{\geq 5} + \epsilon n,$$

where $n_i (i \in \{3, 4\})$ is the number of vertices of degree i in G and $n_{\geq 5}$ is the number of vertices of degree at least 5. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

Samer and Szeider (2010) introduced fast algorithms for #SAT parameterized by primal pathwidth and dual pathwidth. With minor modifications, these algorithms can be adapted to solve the weighted version, specifically WMC, without increasing the time complexity.

Theorem 2 ([Samer and Szeider, 2010]). *Given an instance (F, w) of WMC and a path decomposition P of $G(F)$, there is an algorithm (denoted by `AlgPrimalPw`(F, w, P)) that solves WMC in time $\mathcal{O}^*(2^p)$, where p is the width of P.*

Theorem 3 ([Samer and Szeider, 2010]). *Given an instance (F, w) of WMC and a path decomposition P of $G^d(F)$, there is an algorithm (denoted by `AlgDualPw`(F, w, P)) that solves WMC in time $\mathcal{O}^*(2^p)$, where p is the width of P.*

2.3 Branch-and-Search Algorithms

A *branch-and-search algorithm* first applies reduction rules to reduce the instance and then searches for a solution by branching. We need to use a measure to evaluate the size of the search tree generated in the algorithm. Let μ be the measure and $T(\mu)$ be an upper bound on the size of the search tree generated by the algorithm on any instance with the measure of at most μ. A branching operation, which branches on the instance into l branches with the measure decreasing by at least $a_i > 0$ in the i-th branch, is usually represented by a recurrence relation

$$T(\mu) \leq T(\mu - a_1) + \cdots + T(\mu - a_l),$$

or simply by a *branching vector* $(a_1, \ldots, a_l)$. The *branching factor* of the recurrence, denoted by $\tau(a_1, \ldots, a_l)$, is the largest root of the function $f(x) = 1 - \sum_{1 \leq i \leq l} x^{-a_i}$. If the maximum branching factor for all branching operations in the algorithm is at most γ, then $T(\mu) = O(\gamma^\mu)$. More details about analyzing branching algorithms can be found in [Fomin and Kratsch, 2010]. We say that one branching vector is *not worse* than the other if its corresponding branching factor is not greater than that of the latter. The following useful property about branching vectors can be obtained from Lemma 2.2 and Lemma 2.3 in [Fomin and Kratsch, 2010].

Lemma 1. *A branching vector (a_1, a_2) is not worse than $(p-q, q)$ (or $(q, p-q)$) if $a_1 + a_2 \geq p$ and $a_1, a_2 \geq q > 0$.*

3 Framework of Algorithms

An instance of WMC is denoted as $\mathcal{I} = (F, w)$. For the sake of describing recursive algorithms, we use $\mathcal{I} = (F, w, W)$ to denote an instance, where W is a positive integer and the solution to this instance is $W \cdot \mathrm{WMC}(F, w)$. Initially, it holds that $W = 1$, which corresponds to the original WMC.

Our algorithms for weighted #2-SAT and weighted #3-SAT adopt the same framework, which contains three major phases. The *first phase* is to apply some reduction rules to simplify the instance. The reduction rules we use in the algorithm will be introduced in the next subsection.

The *second phase* is to branch on some variable by assigning either 1 or 0 to it. This phase will create branching vectors and exponentially increase the running time of the algorithm. Specifically, branching on variable x in an algorithm `Alg` means doing the following:

- $W_t \leftarrow$ `Alg`$(F[x = 1], w, W)$;
- $W_f \leftarrow$ `Alg`$(F[x = 0], w, W)$;
- Return $w(x) \cdot W_t + w(\bar{x}) \cdot W_f$.

In our algorithms, we may only branch on variables of (relatively) high degree. When all variables have a low degree, the corresponding primal or dual graphs usually have a small pathwidth. In this case, we will apply Theorem 1 to obtain a path decomposition with small width, and then invoke the algorithms in Theorem 2 and Theorem 3 to solve the problem. This is the *third phase* of our algorithms. Before introducing our algorithms for weighted #2-SAT and #3-SAT, we first introduce our reduction rules for general WMC. Since our reduction rules are applicable for general WMC, they can also be applied to both weighted #2-SAT and weighted #3-SAT.

3.1 Reduction Rules

A reduction rule takes $\mathcal{I} = (F, w, W)$ as input and outputs a new instance $\mathcal{I}' = (F', w', W')$. A reduction rule is *correct* if $\mathrm{WMC}(F, w) \cdot W = \mathrm{WMC}(F', w') \cdot W'$ holds.

In total, we have nine reduction rules. Due to the space limitation, the proofs of the correctness of the rules are deferred to the full version. When we consider a rule, we may assume that all previous rules can not be applied now.

The first four reduction rules are simple and well-known.

R-Rule 1 (Elimination of duplicated literals). *If a clause C contains duplicated literals ℓ, remove all but one ℓ in C.*

R-Rule 2 (Elimination of tautology). *If a clause C contains two complementary literals ℓ and $\bar{\ell}$, remove clause C.*

R-Rule 3 (Elimination of subsumptions). *If there are two clauses C and D such that $C \subseteq D$, remove clause D.*

R-Rule 4 (Elimination of 1-clauses). *If there is a 1-clause (ℓ), then $W \leftarrow W \cdot w(\ell)$, and $F \leftarrow F_{\ell=1}$.*

In the algorithms, we may also generate 0-variables that are unassigned yet. The following rule can eliminate them.

R-Rule 5 (Elimination of 0-variables). *If there is a unassigned variable x with $\deg(x) = 0$, let $W \leftarrow W \cdot (w(x) + w(\bar{x}))$ and remove variable x.*

Next two rules are going to deal with some 2-clauses.

R-Rule 6. *If there are two clauses $\ell_a \ell_b$ and $\ell_a \bar{\ell}_b C$ in F, then remove literal $\bar{\ell}_b$ from clause $\ell_a \bar{\ell}_b C$.*

R-Rule 7. *If there are two clauses $\ell_a \ell_b$ and $\bar{\ell}_a \bar{\ell}_b$ in F, do the following:*

1. $w(\bar{\ell}_b) \leftarrow w(\bar{\ell}_b) \cdot w(\ell_a)$, $w(\ell_b) \leftarrow w(\ell_b) \cdot w(\bar{\ell}_a)$;
2. *replace ℓ_a (resp., $\bar{\ell}_a$) with $\bar{\ell}_b$ (resp., ℓ_b) in F;*
3. *remove $\mathrm{var}(\ell_a)$ and apply R-Rule 2 as often as possible.*

We use `Brute`(F, w) to denote the brute-force algorithm that solves WMC by enumerating all possible assignments. Clearly, `Brute`(F, w) runs in $\mathcal{O}^*(2^{n(F)})$ time and it uses constant time if the formula has only a constant number of veriables. The following two rules are based on a divide-and-conquer idea. However, we only apply them for the cases where one part is of constant size.

R-Rule 8. *If formula F can be partitioned into two non-empty sub-formulas F_1 and F_2 with $\mathrm{var}(F_1) \cap \mathrm{var}(F_2) = \emptyset$ and $n(F_1) \le 10$, do the following:*

1. $W' \leftarrow$ `Brute`(F_1, w);
2. $W \leftarrow W \cdot W'$, *and* $F \leftarrow F_2$.

R-Rule 9. *If there is a variable x such that formula F can be partitioned into two non-empty sub-formulas F_1 and F_2, with $\mathrm{var}(F_1) \cap \mathrm{var}(F_2) = \{x\}$ and $n(F_1) \le 10$, do the following:*

1. $W_t \leftarrow$ `Brute`$(F_1[x = 1], w)$;
2. $W_f \leftarrow$ `Brute`$(F_1[x = 0], w)$;
3. $w(x) \leftarrow w(x) \cdot W_t$, *and* $w(\bar{x}) \leftarrow w(\bar{x}) \cdot W_f$;
4. $F \leftarrow F_2$.

Lemma 2 (♣). *All of the nine reduction rules are correct.*

Definition 5 (Reduced formulas)**.** *A formula F is called reduced if none of the above reduction rules is applicable. We use $R(F)$ to denote the reduced formula obtained by iteratively applying the above reduction rules on F.*

Lemma 3 (♣). *For any formula F, applying any reduction rule on F will not increase the number of clauses or increase the length of any clause. Moreover, it takes polynomial time to transfer F into $R(F)$.*

Lemma 4 (♣). *In a reduced formula F, it holds that (1) all clauses are 2^+-clauses; (2) all 2-clauses only contains 2^+-variables.*

Lemma 5 (♣). *In a reduced formula F, if there is a 2-clause $\ell_a \ell_b$, there is no other clause containing $\ell_a \ell_b$, $\bar{\ell}_a \ell_b$, or $\ell_a \bar{\ell}_b$, and there is no 2-clause $\bar{\ell}_a \bar{\ell}_b$.*

4 The Algorithm for Weighted #2-SAT

In this section, we introduce our algorithm, called `Alg2CNF`, for WMC on 2-CNF formulas. The algorithm is presented in Algorithm 1. As we mentioned before, the algorithm comprises three main phases. Phase one (Line 1) is to apply reduction rules to get a reduced instance. Phase two (Lines 4–8) is going to branch on 5^+-variables and some special 4-variables. After phase two, the primal graph of the formula admits a small pathwidth. Phase three (Steps 10-11) is, based on a path decomposition, to use the algorithm `AlgPrimalPw` in Theorem 2 to solve the problem directly.

Algorithm 1 `Alg2CNF`(F, w, W)

Input: 2-CNF formula F, weight function w, and integer W.
Output: The weighted model count $W \cdot \mathrm{WMC}(F, w)$.

1: Apply reduction rules exhaustively to reduce F and update w and W accordingly.
2: **if** F is empty **then return** W.
3: **if** F contains empty clause **then return** 0.
4: **if** $\deg(F) \ge 5$ **then**
5:     Select a variable x with $\deg(x) = \deg(F)$;
6:     Branch on x.
7: **else if** $\exists\, x$ such that $\deg(x) = 4$ and $|N_4(x)| \ge 3$ **then**
8:     Branch on x.
9: **else**
10:     $P \leftarrow$ path decomposition of $G(F)$ via Theorem 1.
11:     $W_{pw} \leftarrow$ `AlgPrimalPw`(F, w, P);
12:     **return** $W \cdot W_{pw}$.

4.1 The Analysis

Although the algorithm itself is simple, its running time analysis is technically involved. We first prove some properties of a reduced 2-CNF, which will be used in our analysis.

Lemma 6 (♣). *In a reduced 2-CNF formula F, it holds that (1) all clauses are 2-clauses; (2) all variables are 2^+-variables; (3) $n(F) \le m(F)$.*

Lemma 7 (♣). *In a reduced 2-CNF formula F, for a variable x, any clause contains at most one variable in $N_2(x)$.*

Lemma 8. *Let F be a reduced 2-CNF formula and x be a variable in F. All clauses containing x would not appear in $R(F[x = 0])$ and $R(F[x = 1])$.*

Proof. By Lemma 6, all clauses in F are 2-clauses. Consider the case that we assign $x = 1$, and the case for $x = 0$ is analogous. All clauses that contain literal x are satisfied and removed. Furthermore, all clauses that contain literal $\bar{x}$ become 1-clauses, and thus R-Rule 4 would be applied to remove them. Thus, all clauses containing x would not appear in $R(F[x = 0])$ and $R(F[x = 1])$. □

To analyze the running time bound, we focus on phase two and phase three since phase one will not exponentially increase the running time. For phase two, we mainly use Lemma 1 to get the worst branching vector. Thus, we need to analyze lower bounds for the decrease of $m(F)$ in a branching operation, which is formally presented in Lemma 9 below. Due to space limitations, we provide a proof sketch of the lemma that includes several claims, with their proofs deferred to the full version.

Lemma 9. *Let F be a reduced 2-CNF formula of degree d and x be a d-variable in F. Let $\Delta_t = m(F) - m(R(F[x = 1]))$ and $\Delta_f = m(F) - m(R(F[x = 0]))$. It holds that*

1. $\Delta_t, \Delta_f \ge d + |N_2(x)|$;
2. $\Delta_t + \Delta_f \ge 2d + |N_2(x)| + \left\lceil \frac{1}{2} \sum_{2 \le i \le d} (i-1)|N_i(x)| \right\rceil + 1$ *if* $d \le 7$.

Proof Sketch. For clarity, we define the following notations:

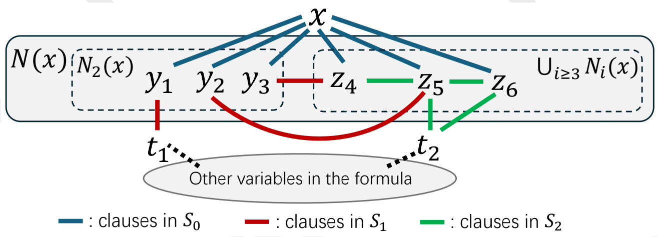

Figure 1: An illustrative example of the primal graph of a formula, highlighting the relationships among variables x, variables in $N_2(x) = \{y_1, y_2, y_3\}$ and $\bigcup_{i\geq 3} N_i(x) = \{z_4, z_5, z_6\}$, and variables t_1, t_2 (that co-occur in clauses with $N(x)$). By Lemma 6, each edge in the graph corresponds to a unique clause in F. By Lemma 7, there is no edge between variables in $N_2(x)$.

- S_0: the set of clauses that contain variable x.
- S_1: the set of clauses that contain variable(s) in $N_2(x)$ but not contain variable x.
- S_2: the set of clauses that contain variable(s) in $N_i(x)$, where $i \geq 3$, but not contain any variable in $N_2(x) \cup \{x\}$.

By definition, S_0, S_1, S_2 are pairwise disjoint. The primal graph of F shown in Figure 1 provides a useful perspective for understanding these notations and the subsequent proofs.

We plan to analyze the bounds for Δ_t, Δ_f, and $\Delta_t + \Delta_f$ by considering whether a clause in $S_0 \cup S_1 \cup S_2$ would be removed after we assign a value to variable x (*i.e.*, whether a clause would appear in $R(F[x = 0])$ or $R(F[x = 0])$).

Claim 1 (♣). *All clauses in S_0 and S_1 would not appear in both $R(F[x = 0])$ and $R(F[x = 1])$.*

By claim 1, we have $\Delta_t \geq |S_0| + |S_1|$ and $\Delta_f \geq |S_0| + |S_1|$. Since $|S_0| = d$ (by definition) and $|S_1| = |N_2(x)|$ (by Lemma 7), it holds that $\Delta_t, \Delta_f \geq d + |N_2(x)|$.

Next, we consider the clauses in S_2. Let D be a clause that contains variable x and a variable $z \in N_i(x)$ where $i \geq 3$. Assigning either $x = 0$ or $x = 1$ would make D become a 1-clause that only contains variable z. When clause D becomes such 1-clause, R-Rule 4 would be applied to assign a value to z, and then remove all clauses containing z according to Lemma 8. Thus, for a clause in S_2, it would not appear in at least one of $R(F[x = 0])$ and $R(F[x = 1])$. Together with previous analyses on S_0 and S_1, we have

$$\begin{aligned}\Delta_t + \Delta_f &\geq 2\,|S_0| + 2\,|S_1| + |S_2| \\ &= 2d + 2\,|N_2(x)| + |S_2|\,.\end{aligned} \tag{1}$$

To accurately characterize S_2 (and $|S_2|$), we need some additional notations. For a variable $y \in N(x)$, we define $P(y) := N(y) \cap N(x)$ and $Q(y) := N(y) \backslash (N(x) \cup \{x\})$. For instance, in the example shown in Figure 1, we have $P(z_5) = \{y_2, z_4, z_6\}$ and $Q(z_5) = \{t_2\}$. For each $i \geq 2$ we define $p_i := \sum_{y \in N_i(x)} |P(y)|$ and $q_i := \sum_{y \in N_i(x)} |Q(y)|$.

Claim 2 (♣). *For all $i \geq 2$, we have $p_i + q_i = (i-1)|N_i(x)|$.*

We write $p_{\geq 3} := \sum_{i \geq 3} p_i$ and $q_{\geq 3} := \sum_{i \geq 3} q_i$ for brevity. The size of S_2 is given in the following claim.

Claim 3 (♣). *It holds that*

$$|S_2| = \frac{1}{2}(p_{\geq 3} - p_2) + q_{\geq 3} \text{ and } (p_{\geq 3} - p_2) \bmod 2 = 0. \tag{2}$$

By putting (2) into (1) and writing $\beta := (p_{\geq 3} - p_2) + 2(p_2 + q_2) + 2q_{\geq 3}$ for convenience, we have

$$\begin{aligned}\Delta_t + \Delta_f &\geq 2d + 2(p_2 + q_2) + \frac{1}{2}(p_{\geq 3} - p_2) + q_{\geq 3} \\ &= 2d + (p_2 + q_2) + \frac{1}{2}\beta = 2d + |N_2(x)| + \frac{1}{2}\beta.\end{aligned} \tag{3}$$

Claim 4 (♣). *It holds that*

$$\frac{1}{2}\beta \geq \left\lceil \frac{1}{2} \sum_{2 \leq i \leq d} (i-1)|N_i(x)| \right\rceil + 1.$$

With (3), the lemma directly follows from Claim 4. □

With Lemma 9 in hand, we proceed to determine the branching vectors of our branching operations.

Lemma 10 (♣). *In `Alg2CNF`, the branching operation in Line 6 generates a branching vector not worse than* $(5, 11)$.

Lemma 11. *In `Alg2CNF`, the branching operation in Line 8 generates a branching vector not worse than* $(4, 11)$.

Proof. In this branching operation, we branch on a 4-variable x with $|N_4(x)| \geq 3$. Note that in this step we have $\deg(F) = 4$. By Lemma 9 with $d = 4$, we have $\Delta_t, \Delta_f \geq d = 4$ and

$$\begin{aligned}\Delta_t + \Delta_f &\geq 2d + |N_2(x)| + \left\lceil \frac{1}{2} \sum_{2 \leq i \leq d} (i-1)|N_i(x)| \right\rceil + 1 \\ &\geq 2d + \left\lceil \frac{1}{2} \left(2\,|N_2(x)| + \sum_{i=3}^{d} (i-1)|N_i(x)| \right) \right\rceil + 1 \\ &\geq 2d + \left\lceil \frac{1}{2} \left(|N_4(x)| + \sum_{2 \leq i \leq d} 2|N_i(x)| \right) \right\rceil + 1 \\ &\geq 2d + \left\lceil \frac{1}{2}(3 + 2d) \right\rceil + 1 = 15.\end{aligned}$$

By Lemma 1, the branching vector generated by this step is not worse than $(4, 11)$. □

Next, we analyze the phase three (Lines 10–11).

Lemma 12. *Phase three (lines 10-11) of `Alg2CNF` can be excuted in $\mathcal{O}^*(1.1082^m)$ time.*

Proof. When the algorithm reaches Line 10, the fomrula F is a reduced 2-CNF formula with $d(F) \leq 4$ such that for every 4-variable x, $|N_4(x)| \leq 2$.

Let $n := n(F)$, $m := m(F)$, and n_i (resp., $n_{\geq i}$) be the number of variables with degree i (resp., with degree $\geq i$) in F, where $i \in \mathbb{Z}$. Consider the primal graph $G(F)$ of formula F. Note that n is also the number of vertices in $G(F)$, and n_i (resp., $n_{\geq i}$) is also the number of vertices with degree i in $G(F)$. By Lemma 6, we have $n_1 = 0$. Since $d(F) \leq 4$, we have $n_{\geq 5} = 0$ and

$$m = \frac{2n_2 + 3n_3 + 4n_4}{2} = \frac{3}{2}(n_3 + 2n_4) - n_4 + n_2.$$

By rearranging the above equation, we get

$$n_3 + 2n_4 = \frac{2}{3}(m + n_4 - n_2) \leq \frac{2}{3}(m + n_4). \quad (4)$$

Let V_4 be the set of 4-variables in the formula. The number of clauses that contain two 4-variables is $\frac{1}{2}\sum_{x\in V_4}|N_4(x)|$, and the number of clauses that contain at most one 4-variable is at least $\sum_{x\in V_4}(4 - |N_4(x)|)$. Thus, we have

$$\begin{aligned} m &\geq \frac{1}{2}\sum_{x\in V_4}|N_4(x)| + \sum_{x\in V_4}(4 - |N_4(x)|) \\ &\geq \sum_{x\in V_4}(4 - \frac{1}{2}|N_4(x)|) \geq \sum_{x\in V_4}(4-1) = 3n_4, \end{aligned}$$

which means $n_4 \leq m/3$. By putting this into (4), we have

$$n_3 + 2n_4 \leq \frac{2}{3}(m + n_4) \leq \frac{8}{9}m. \quad (5)$$

Let ϵ be a small constant (say 10^{-9}) and n_ϵ be the corresponding integer (which is also a constant) in Theorem 1. Let $n := n(F)$ and $m := m(F)$. If $n \leq n_\epsilon$, we invoke algorithm `Brute` to solve the problem in constant time. Otherwise, if $n > n_\epsilon$, we can apply Theorem 1 and get

$$\begin{aligned} \mathrm{pw}(G(F)) &\leq n_3/6 + n_4/3 + n_{\geq 5} + \epsilon n \\ &= (n_3 + 2n_4)/6 + \epsilon n \\ &\leq (4/27 + \epsilon)\, m \qquad \text{by (5).} \end{aligned}$$

Moreover, by Theorem 1, a path decomposition of $G(F)$ with width at most $(4/27 + \epsilon)\, m$ can be constructed in polynomial time. Then, we can apply Theorem 2 to solve the problem in time $\mathcal{O}^*(2^{(4/27+\epsilon)m}) \subseteq \mathcal{O}^*(1.1082^m)$. □

Now we are ready to conclude a running-time bound of Algorithm `Alg2CNF`. By Lemma 10 and Lemma 11, branching operations in Line 6 and Line 8 generate a branching vector not worse than $(5, 11)$ and $(4, 11)$, respectively. By Lemma 12, phase three (Lines 10 and 11) takes $\mathcal{O}^*(1.1082^m)$ time. Since $\tau(5, 11) < 1.0956$ and $\tau(4, 11) < 1.1058$, we have the following result.

Theorem 4. *Algorithm `Alg2CNF` solves WMC on 2-CNF formulas in $\mathcal{O}^*(1.1082^m)$ time, where m is the number of clauses in the input formula.*

5 The Algorithm for Weighted #3-SAT

Our algorithm for WMC on 3-CNF formulas is called `Alg3CNF` and presented in Algorithm 2. The first phase is also to apply reduction rules to get a reduced instance. Note that by Lemma 3, a reduced formula is still a 3-CNF. The second phase is to branch on all 3^+-variables. When the maximum degree of the formula is at most 2, we compute a path decomposition of the dual graph of the formula and then invoke the algorithm `AlgDualPw` to solve the problem.

5.1 The Measure

The analysis of the algorithm is different from that of the algorithm for #2-SAT. In this algorithm, it may not be effective to use $m(F)$ as the measure in the analysis since we can not guarantee this measure always decreases in all our steps. For example, a variable x may only appear as a positive literal in some 3-clauses. After assigning $x = 0$, it is possible that no reduction rule is applicable and no clause is removed (*i.e.*, $m(F) = m(R(F[x = 0]))$). One of our strategies is to use the following combinatorial measure to analyze the algorithm

$$\mu(F) := m_3(F) + \alpha \cdot m_2(F),$$

where $m_i(F)(i \in \{2, 3\})$ is the number of i-clauses in formula F and $0 < \alpha < 1$ is a tunable parameter. Note that $m(F) = m_3(F) + m_2(F)$ since there is no 1-clause in a reduced formula by Lemma 4 (we can simply assume that the initial input formula is reduced). Thus, $\mu(F) \leq m(F)$. It can be verified that all the reduction rules would not increase $\mu(F)$ for any $0 < \alpha < 1$. If we can get a running time bound of $\mathcal{O}^*(c^{\mu(F)})$, with a real number $c > 1$, we immediately get a running time bound of $\mathcal{O}^*(c^{m(F)})$. We will first analyze the algorithm and obtain the branching vectors related to α, and then set the value of α to minimize the largest factor.

Algorithm 2 `Alg3CNF`(F, w, W)

Input: 3-CNF formula F, weight function w, and integer W.
Output: The weighted model count $W \cdot \mathrm{WMC}(F, w)$.

Apply reduction rules exhaustively to reduce F and update w and W accordingly.
if F is empty **then return** W.
if F contains empty clause **then return** 0.
if there is a d-variable x in F with $d \geq 3$ **then**
  Branch on x.
else
  $P \leftarrow$ path decomposition of $G^d(F)$ via Theorem 1.
  $W_{pw} \leftarrow$ `AlgDualPw`(F, w, P);
  return $W \cdot W_{pw}$

5.2 The Analysis

We first analyze lower bounds for the decrease of the measure $\mu(F)$ in a branching operation.

Lemma 13. *Let F be a reduced 3-CNF formula, x be a variable in F, and $c_k (k \in \{2, 3\})$ be the number of k-clauses containing variable x. Let $\Delta_t := \mu(F) - \mu(R(F[x = 1]))$ and $\Delta_f := \mu(F) - \mu(R(F[x = 0]))$. It holds that*

1. $\Delta_t, \Delta_f \geq c_2 \cdot \alpha + c_3 \cdot (1 - \alpha)$;
2. $\Delta_t + \Delta_f \geq c_2 \cdot 2\alpha + c_3 \cdot (2 - \alpha)$.

Proof. Let S_k^ℓ, where $k \in \{2, 3\}$ and $\ell \in \{x, \bar{x}\}$, be the set of k-clauses that contain literal ℓ. By definition, we have $c_k = |S_k^x| + |S_k^{\bar{x}}|$ for $k \in \{2, 3\}$.

Consider what happens after we assign $x = 1$. First, all clauses containing literal x (*i.e.*, clauses in S_3^x and S_2^x) are satisfied (and so removed). This decreases Δ_t by at least $|S_3^x| + |S_2^x| \cdot \alpha$. Second, all 3-clauses that contain literal $\bar{x}$ (*i.e.*, clauses in $S_3^{\bar{x}}$) become 2-clauses. This decreses Δ_t by at least $|S_3^{\bar{x}}| \cdot (1 - \alpha)$. Third, all 2-clauses that contain literal $\bar{x}$ (*i.e.*, clauses in $S_2^{\bar{x}}$) become 1-clauses, and then R-Rule 4 would be applied to remove these clauses. This decreases Δ_t

by at least $|S_2^{\bar{x}}| \cdot \alpha$. In summary, we have

$$\begin{aligned}\Delta_t &\geq |S_3^x| + |S_2^x| \cdot \alpha + \left|S_3^{\bar{x}}\right| \cdot (1-\alpha) + \left|S_2^{\bar{x}}\right| \cdot \alpha \\ &= |S_3^x| + (|S_2^x| + \left|S_2^{\bar{x}}\right|) \cdot \alpha + (c_3 - |S_3^x|) \cdot (1-\alpha) \\ &= c_2 \cdot \alpha + c_3 \cdot (1-\alpha) + |S_3^x| \cdot \alpha.\end{aligned}$$

Analogously, we have $\Delta_f \geq c_2 \cdot \alpha + c_3 \cdot (1-\alpha) + |S_3^{\bar{x}}| \cdot \alpha$. Thus, $\Delta_t, \Delta_f \geq c_2 \cdot \alpha + c_3 \cdot (1-\alpha)$ and

$$\begin{aligned}\Delta_t + \Delta_f &\geq 2 \cdot c_2 \cdot \alpha + 2 \cdot c_3 \cdot (1-\alpha) + \left|S_3^{\bar{x}}\right| \cdot \alpha + |S_3^x| \cdot \alpha \\ &= c_2 \cdot 2\alpha + c_3 \cdot 2(1-\alpha) + c_3 \cdot \alpha \\ &= c_2 \cdot 2\alpha + c_3 \cdot (2-\alpha).\end{aligned}$$

This completes the proof. □

Armed with Lemma 13, we can derive the branching vectors generated by phase two (Line 5) in the algorithm.

Lemma 14. *In* `Alg3CNF`*, the branching operation in Line 5 generates a branching vector not worse than the worst one of the following branching vectors:*

$$(3, 3-3\alpha), (2+\alpha, 2-\alpha), (1+2\alpha, 1+\alpha), \textit{ and } (3\alpha, 3\alpha).$$

Proof. Let x be a d-variable with $d \geq 3$. Let $p := c_2 \cdot 2\alpha + c_3 \cdot (2-\alpha)$ and $q := c_2 \cdot \alpha + c_3 \cdot (1-\alpha)$. By Lemma 13, we have $\Delta_t + \Delta_f \geq p$ and $\Delta_t, \Delta_f \geq q$. With Lemma 1, we know that the branching vector is not worse than $(q, p-q) = (c_2 \cdot \alpha + c_3 \cdot (1-\alpha), c_2 \cdot \alpha + c_3)$. It is evident that larger c_2 and c_3 result in superior branching vectors. Since $c_2 + c_3 = d \geq 3$, it suffices to consider the case where $c_2 + c_3 = 3$. By enumerating all four possible configurations of c_2 and c_3, we obtain the results stated in the lemma. □

Next, we analyze the time complexity of phase three (Lines 7 and 8) in the algorithm.

Lemma 15. *Phase three (Lines 7-8) of* `Alg3CNF` *can be executed in* $\mathcal{O}^*(1.1225^{\frac{\mu(F)}{\alpha}})$ *time.*

Proof. When the algorithm reaches Line 7, the branching operation is not applicable. Thus, at this point, the formula F is a reduced 3-CNF formula with $\deg(F) \leq 2$.

Let $m := m(F)$ and $\mu := \mu(F)$. Consider the dual graph $G^d(F)$ of formula F. The number of vertices in $G^d(F)$ is m.

Let $C \in F$ be a clause in formula F. We have $|C| \leq 3$. Since each variable in C has a degree of at most two, the number of clauses that share a common variable with C is at most $|C| \leq 3$. That is, for any $C \in F$, we have $|\{D \in F \mid D \neq C \text{ and } \mathrm{var}(C) \cap \mathrm{var}(D) \neq \emptyset\}| \leq 3$.

This means that in $G^d(F)$, each vertex has a degree of at most three. Let $n_i (i \in \mathbb{Z})$ be the number of vertices with degree i in $G^d(F)$. We have $n_i = 0$ for $i \geq 4$ and $n_3 \leq m$. Let ϵ be a small const (say 10^{-9}) and m_ϵ be the corresponding integer (which is also a constant) in Theorem 1. If $m \leq m_\epsilon$, we invoke brute-force algorithm `Brute` to solve the problem in constant time. Otherwise, if $m > m_\epsilon$, we can apply Theorem 1 and get

$$\begin{aligned}\mathrm{pw}(G^d(F)) &\leq n_3/6 + n_4/3 + n_{\geq 5} + \epsilon m \\ &\leq (1/6 + \epsilon)\, m \leq (1/6 + \epsilon)\, \frac{\mu}{\alpha}.\end{aligned}$$

Here, the last inequality follows from $m \leq \frac{1}{\alpha}\mu$, which can be derived by the definition of m and μ. In addition, by Theorem 1, a path decomposition of $G^d(F)$ with width at most $(1/6+\epsilon)\frac{\mu}{\alpha}$ can be constructed in polynomial time. Then, we can apply Theorem 3 to solve the problem in time $\mathcal{O}^*(2^{\frac{(1/6+\epsilon)\mu}{\alpha}}) \subseteq \mathcal{O}^*(1.1225^{\frac{\mu}{\alpha}})$. □

Phases	Branching vectors	Factors / Base $\alpha = 0.6309297$
Phase two	$(3, 3-3\alpha)$	**1.4423**
	$(2+\alpha, 2-\alpha)$	1.4324
	$(1+2\alpha, 1+\alpha)$	1.4325
	$(3\alpha, 3\alpha)$	**1.4423**
Phase three	-	$1.1225^{\frac{1}{\alpha}} = 1.2011$

Table 1: The branching vectors and corresponding factors generated by phase two of `Alg3CNF`, and the base of the time complexity in terms of $\mu(F)$ of phase three, all under $\alpha = 0.6309297$.

We are now poised to analyze the overall running time of `Alg3CNF`. The time complexity of phase two (by Lemma 14) and phase three (by Lemma 15) are summarized in Table 1. By setting $\alpha = 0.6309297$, the largest factor in phase two is minimized to 1.4423, corresponding to the branching vectors $(3, 3-3\alpha)$ and $(3\alpha, 3\alpha)$. The time complexity of phase three is $\mathcal{O}^*(1.1225^{\frac{\mu(F)}{\alpha}}) \subseteq \mathcal{O}^*(1.2011^{\mu(F)})$. With $\mu(F) \leq m(F)$, we arrive at the following result.

Theorem 5. *Algorithm* `Alg3CNF` *solves WMC on 3-CNF formulas in* $\mathcal{O}^*(1.4423^m)$ *time, where* m *is the number of clauses in the input formula.*

6 Conclusion and Discussion

In this paper, we demonstrate that Weighted Model Counting (WMC) on 2-CNF and 3-CNF formulas can be solved in $\mathcal{O}^*(1.1082^m)$ and $\mathcal{O}^*(1.4423^m)$ time, respectively, achieving significant improvements over previous results. The trivial barrier of $\mathcal{O}^*(2^m)$ for WMC on general CNF formulas cannot be overcome unless SETH fails [Cygan *et al.*, 2016]. It remains an open question whether a running time bound of $\mathcal{O}^*(c^m)$ with a constant $c < 2$ can be achieved for WMC on k-CNF formulas for any constant k.

Our algorithms first use branch-and-search to effectively eliminate certain problem structures (such as high-degree vertices). Once the remaining problem exhibits some favorable structural properties (such as having a small primal pathwidth or dual pathwidth), dynamic programming and other methods are employed to solve the problem. This approach may have potential for application in solving other problems. Furthermore, this method holds significant promise in the design of practical algorithms. In practical solving, tree decompositions have been employed in various model counters [Dudek *et al.*, 2020; Hecher *et al.*, 2020; Fichte *et al.*, 2019; Korhonen and Järvisalo, 2021; Fichte *et al.*, 2022; Fichte *et al.*, 2023b]. Therefore, the practicality of this approach warrants further investigation and exploration.

Acknowledgements

The work is supported by the National Natural Science Foundation of China, under the grants 62372095, 62172077, and 62350710215.

References

[Bacchus *et al.*, 2003] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 340–351. IEEE Computer Society, 2003.

[Beigel and Eppstein, 2005] Richard Beigel and David Eppstein. 3-coloring in time $o(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005.

[Biere *et al.*, 2021] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

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

[Chu *et al.*, 2021] Huairui Chu, Mingyu Xiao, and Zhe Zhang. An improved upper bound for SAT. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3707–3714. AAAI Press, 2021.

[Cook, 1971] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[Cygan *et al.*, 2016] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.

[Dahllöf *et al.*, 2005] Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2sat and 3sat formulae. *Theor. Comput. Sci.*, 332(1-3):265–291, 2005.

[Dubois, 1991] Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theor. Comput. Sci.*, 81(1):49–64, 1991.

[Dudek *et al.*, 2020] Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. DPMC: weighted model counting by dynamic programming on project-join trees. In Helmut Simonis, 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, 2020.

[Dueñas-Osorio *et al.*, 2017] Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4488–4494. AAAI Press, 2017.

[Fichte *et al.*, 2019] Johannes Klaus Fichte, Markus Hecher, and Markus Zisser. An improved gpu-based SAT model counter. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 of *Lecture Notes in Computer Science*, pages 491–509. Springer, 2019.

[Fichte *et al.*, 2022] Johannes Klaus Fichte, Markus Hecher, Patrick Thier, and Stefan Woltran. Exploiting database management systems and treewidth for counting. *Theory Pract. Log. Program.*, 22(1):128–157, 2022.

[Fichte *et al.*, 2023a] Johannes Klaus Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of SAT. *Commun. ACM*, 66(6):64–72, 2023.

[Fichte *et al.*, 2023b] Johannes Klaus Fichte, Markus Hecher, Michael Morak, Patrick Thier, and Stefan Woltran. Solving projected model counting by utilizing treewidth and its limits. *Artif. Intell.*, 314:103810, 2023.

[Fomin and Kratsch, 2010] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[Fomin *et al.*, 2009] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009.

[Gaspers and Sorkin, 2009] Serge Gaspers and Gregory B. Sorkin. A universally fastest algorithm for max 2-sat, max 2-csp, and everything in between. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 606–615. SIAM, 2009.

[Gomes *et al.*, 2021] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 993–1014. IOS Press, 2021.

[Hecher *et al.*, 2020] Markus Hecher, Patrick Thier, and Stefan Woltran. Taming high treewidth with abstraction, nested dynamic programming, and database technology. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2020.

[Impagliazzo and Paturi, 2001] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[Iwama, 1989] Kazuo Iwama. CNF satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, 18(2):385–391, 1989.

[Korhonen and Järvisalo, 2021] Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters (short paper). In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 8:1–8:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Kutzkov, 2007] Konstantin Kutzkov. New upper bound for the #3-sat problem. *Inf. Process. Lett.*, 105(1):1–5, 2007.

[Lozinskii, 1992] Eliezer L. Lozinskii. Counting propositional models. *Inf. Process. Lett.*, 41(6):327–332, 1992.

[Narodytska *et al.*, 2019] Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and João Marques-Silva. Assessing heuristic machine learning explanations with model counting. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2019.

[Roth, 1996] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

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

[Sang *et al.*, 2005] Tian Sang, Paul Beame, and Henry A. Kautz. Performing bayesian inference by weighted model counting. In Manuela M. Veloso and Subbarao Kambhampati, 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, 2005.

[Valiant, 1979] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Wahlström, 2008] Magnus Wahlström. A tighter bound for counting max-weight solutions to 2sat instances. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2008.

[Xiao, 2022] Mingyu Xiao. An exact maxsat algorithm: Further observations and further improvements. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1887–1893. ijcai.org, 2022.

[Zhang, 1996] Wenhui Zhang. Number of models and satisfiability of sets of clauses. *Theor. Comput. Sci.*, 155(1):277–288, 1996.

[Zhou *et al.*, 2010] Junping Zhou, Minghao Yin, and Chunguang Zhou. New worst-case upper bound for #2-sat and #3-sat with the number of clauses as the parameter. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, pages 217–222. AAAI Press, 2010.