Complementary Branching for #SAT: Bounded-Degree Bounds and Faster #3-SAT

Konstantin Kutzkov
Archon Labs
kutzkov@gmail.com

Abstract

We present *complementary branching*, a new branching approach for model counting based on the complement counting paradigm. Instead of branching on individual variables, complementary branching decomposes the counting problem for arbitrary CNF formulas into several subformulas, which naturally augments the classic DPLL branching. We show two novel results in which the main tool is complementary branching. First, we design new #SAT algorithms for sparse CNF formulas running in time $\mathcal{O}^*(2^{\alpha n})$, for some constant $\alpha < 1$. As a second application, we improve the best known deterministic upper bound for #3-SAT to $\mathcal{O}^*(1.637^n)$ by using a version of clause learning based on complementary branching. These results demonstrate that complementary branching is a powerful tool for designing faster exact algorithms for propositional model counting.

1 Introduction

The Boolean Satisfiability problem (SAT), the first problem shown to be NP-complete [Cook, 1971], is at the heart of many artificial intelligence tasks and applications. Its counting variant, the Propositional Model Counting problem (#SAT) is the problem of determining the number of satisfying assignments of a given formula. It was introduced and shown to be #P-complete by Valiant [Valiant, 1979]. Exact and approximate model counting are the backbone in probabilistic inference and graphical models [Chavira and Darwiche, 2008], probabilistic databases and probabilistic logic programming [Belle, 2017], knowledge compilation and exact inference [Darwiche and Marquis, 2002], planning and reasoning [Speck *et al.*, 2025], cryptographic security analysis [Beck *et al.*, 2020], and neuro-symbolic reasoning [Manhaeve *et al.*, 2021]. The wide range of applications has led to the creation of dedicated model counting competitions [Fichte *et al.*, 2021; Fichte and Hecher, 2025].

Not surprisingly, the design of fast algorithms for satisfiability problems has been a major research topic in the past decades. For Boolean formulas in conjunctive normal form (CNF), the three main satisfiability problems are decision SAT, its optimization extension Max-SAT, and the counting variant #SAT. A naïve algorithm for each of the problems runs in time $\mathcal{O}^*(2^n)$ by explicitly evaluating all possible 2^n variable assignments. Under the Strong Exponential Time Hypothesis [Calabro *et al.*, 2009], general SAT cannot be solved in time $\mathcal{O}(c^n)$ for some constant $c < 2$. Thus, the focus has been on structural parameters of the underlying CNF instances, for which sub-2^n algorithms can be designed.

Two natural parameters are the *clause width* k (maximum number of literals per clause) and the *clause density* $\delta = m/n$ (ratio of clauses to variables). For decision SAT, many algorithms have been designed that run significantly faster than 2^n for fixed k [Paturi *et al.*, 1997; Dantsin *et al.*, 2002]. Density-parameterized bounds can be obtained via a technique called *width reduction* [Schuler, 2005] to k-SAT algorithms. [Calabro *et al.*, 2006] established a duality between the width and density parameters for decision SAT and showed that SAT with density δ reduces to k-SAT for $k = \mathcal{O}(\log \delta)$.

This is not the case for Max-SAT and #SAT. It is an open question if Max-k-SAT can be solved in less than 2^n steps for fixed k but there are many algorithms for general CNF instances that run in time $\mathcal{O}^*(2^{n(1-\frac{1}{f(\delta)})})$ for some monotonically growing function f. On the contrary, #k-SAT is trivially solvable in time $\mathcal{O}^*(c_k^n)$, for $c_k < 2$, but the values c_k are too large for width reduction to work, and almost no #SAT algorithms have been specifically designed for sparse instances. This is perhaps surprising given the practical importance of sparse formulas as evident in model counting competitions [Fichte *et al.*, 2021; Fichte and Hecher, 2025].

Our Contribution. We present *complementary branching*, a branching approach for model counting based on the complement counting equality $\#(F \wedge G) = \#F - \#(F \wedge \neg G)$. By negating a CNF subformula we obtain a DNF, which can then be naturally branched on. This allows branching on subformulas rather than individual variables, and proves particularly effective when variables of low degree appear in long clauses as we can assign many variables at once.

As a main application of complementary branching, we design a novel #SAT algorithm for CNF formulas of bounded variable degree Δ and no restriction on clause width. The algorithm runs in time $\mathcal{O}^*(2^{(1-\frac{1}{3D})n})$ for $D = \max(\Delta, 3)$ and can be extended to solving #SAT for formulas with density $\delta = m/n$ in time $\mathcal{O}^*(2^{(1-\frac{1}{c\delta})n})$ for some constant $c > 0$.

As a second application, we extend complementary branching to a variation of clause learning. By applying it to several tight cases in the analysis from [Kutzkov, 2007], we improve the upper bound for #3-SAT from $\mathcal{O}^*(1.6423^n)$ to $\mathcal{O}^*(1.637^n)$. While incremental, this is the first improvement to the bound in nearly two decades.

2 Notation and Definitions

A *Boolean variable* (or simply variable) can take the value 1 (True) or 0 (False). For a variable x, the positive literal is x and the negative literal is $\bar{x}$. The sign of a literal is denoted by $\mathrm{sign}(\cdot)$, with $\mathrm{sign}(x) = 1$ and $\mathrm{sign}(\bar{x}) = 0$.

A *clause* is a disjunction (OR) of literals. A *CNF formula* (Conjunctive Normal Form) is a conjunction (AND) of clauses, while a *DNF formula* (Disjunctive Normal Form) is a disjunction of conjunctions of literals.

The negation of a Boolean formula F is denoted by $\neg F$. Note that, in general, the negation of a CNF formula is a DNF formula, and vice versa.

For a formula F, we denote by $\mathrm{var}(F)$ the set of variables in F. Let $n = |\mathrm{var}(F)|$ be the number of variables and m the number of clauses. The *density* of F is defined as

$$\delta = \frac{m}{n}.$$

For a clause C, $\mathrm{var}(C)$ denotes the set of variables that occur in C.

An *assignment* for a variable set V is a mapping $\sigma : V \to \{0, 1\}$. A literal x is satisfied by σ if $\sigma(x) = 1$, and $\bar{x}$ is satisfied if $\sigma(x) = 0$. A clause is satisfied if at least one of its literals is satisfied. An assignment σ is a *model* of a formula F if it satisfies all clauses in F, which we denote by $\sigma \models F$.

For a variable $x \in \mathrm{var}(F)$, we write $F[x = 1]$ and $F[x = 0]$ for the formulas obtained from F by setting x to 1 and 0, respectively. We also define

$F_a = \{C \in F : a \in \mathrm{var}(C)\}$ and $F_{-a} = \{C \in F : a \notin \mathrm{var}(C)\}$, i.e., F_a is the subformula of F consisting of the clauses that contain a, and F_{-a} is the subformula consisting of the clauses that do not contain a.

Definition 1 (#SAT)**.** *Given a Boolean formula F, the #SAT problem asks for the number of satisfying assignments of F, that is*

$$\#\mathrm{SAT}(F) := |\{\sigma : \sigma \models F\}|.$$

Definition 2 (#k-SAT)**.** *For an integer $k \geq 2$, the #k-SAT problem is the restriction of #SAT to formulas in k-CNF, i.e., CNF formulas where each clause has at most k literals.*

In the weighted variant, each literal is assigned a positive weight, the weight of an assignment is the product of its literal weights and the weighted model count of a formula is the sum of the weights of all satisfying assignments, see [Peng *et al.*, 2025]. For ease of presentation, we will consider the unweighted version but all our results extend to weighted model counting.

The degree of a variable v is the number of clauses containing v, and by $d_k(v)$ we denote the number of clauses of length k containing v.

Branching Algorithms. Branching algorithms solve combinatorial problems by recursively decomposing them into subproblems. At each step, the algorithm uses some criterion to divide the current formula into subformulas, the so-called branches, then recursively solves the resulting subproblems and aggregates their solutions.

Definition 3 (Branching Vector)**.** *Let $T(n)$ be the running time of a branching algorithm on an instance of size n. If a branching operation creates k subproblems of sizes $n - t_1, n - t_2, \ldots, n - t_k$ respectively, then $(t_1, t_2, \ldots, t_k)$ is called the* branching vector *of this operation.*

Definition 4 (Branching Number)**.** *The* branching number $\tau(t_1, t_2, \ldots, t_k)$ *of a branching vector $(t_1, t_2, \ldots, t_k)$ is the unique positive root of the equation*

$$x^{-t_1} + x^{-t_2} + \cdots + x^{-t_k} = 1.$$

The branching number determines the exponential factor in the running time such that if the largest branching number among all branching operations is τ, then the algorithm runs in time $\mathcal{O}^*(\tau^n)$.

Throughout the paper, log denotes the base-2 logarithm. We use $\mathcal{O}^*$ notation to suppress polynomial factors, i.e., $\mathcal{O}^*(f(n)) = \mathcal{O}(f(n) \cdot \mathrm{poly}(n))$.

3 Complementary Branching

As a warm-up, we illustrate complementary branching in its simplest form and design a *clause branching* #SAT algorithm that runs in time $\mathcal{O}^*(2^m)$ and can be seen as a simplification of the inclusion-exclusion technique [Lozinskii, 1992].

Let F be a CNF formula and let C be a clause such that $F = G \wedge C$. Using the basic identity

$$\#SAT(G) = \#SAT(G \wedge C) + \#SAT(G \wedge \neg C)$$

we obtain

$$\#SAT(F) = \#SAT(G) - \#SAT(G \wedge \neg C).$$

Thus, we can always select a clause C and branch as "remove C" vs. "falsify C", the recursion base being that an empty formula over n unassigned variables has 2^n models.

We can extend the above idea such that we obtain strong branchings with respect to n in certain cases. Consider $F = F_{-a} \wedge (a \vee b \vee c)$ where a appears only in this clause. Branching on a gives $\#(F_{-a})$ for a=1 and $\#(F_{-a} \wedge (b \vee c))$ for a=0. Observe that $\#(F_{-a}) = \#(F_{-a} \wedge (b \vee c)) + \#(F_{-a} \wedge \bar{b} \wedge \bar{c})$, so $\#(F) = 2 \cdot \#(F_{-a}) - \#(F_{-a} \wedge \bar{b} \wedge \bar{c})$. Thus, we need to compute $\#(F_{-a})$ and $\#(F_{-a} \wedge \bar{b} \wedge \bar{c})$, which yields a branching number $\tau(1, 3)$.

The next two lemmas formally present the concept of complementary counting, and how it can be used to design a new branching technique that we call DNF branching.

Lemma 1 (Complementary counting)**.** *Let F be a Boolean formula and $a \in \mathrm{var}(F)$. Then*

$$\#\mathrm{SAT}(F) = 2 \cdot \#\mathrm{SAT}(F_{-a}) -$$

$$\big(\#\mathrm{SAT}(F_{-a} \wedge \neg F_a[a = 0]) + \#\mathrm{SAT}(F_{-a} \wedge \neg F_a[a = 1])\big)$$

Proof. Partition by the value of a:

$$\#\mathrm{SAT}(F) = \#\mathrm{SAT}(F_{-a} \wedge F_a[a=0]) + \#\mathrm{SAT}(F_{-a} \wedge F_a[a=1])$$

For any Boolean formulas $G, H \subseteq F$, it holds $\#\mathrm{SAT}(G) = \#\mathrm{SAT}(G \wedge H) + \#\mathrm{SAT}(G \wedge \neg H)$. Setting $G = F_{-a}$ and $H = F_a[a=0]$ yields

$$\#\mathrm{SAT}(F_{-a} \wedge F_a[a=0]) = \#\mathrm{SAT}(F_{-a}) - \#\mathrm{SAT}(F_{-a} \wedge \neg F_a[a=0])$$

and similarly

$$\#\mathrm{SAT}(F_{-a} \wedge F_a[a=1]) = \#\mathrm{SAT}(F_{-a}) - \#\mathrm{SAT}(F_{-a} \wedge \neg F_a[a=1]).$$

Summing both equalities proves the lemma. □

Note that $a \notin \mathrm{var}(F_{-a})$, and if F is over n variables then F_{-a} is over $n-1$ variables. If all clauses of F_{-a} are satisfied, then $\#\mathrm{SAT}(F_{-a}) = 2^{n-1}$. Similarly, if all clauses are satisfied in $F_a[a=1]$, then $\#\mathrm{SAT}(\neg F_a[a=1]) = 0$.

Lemma 2 (DNF branching)**.** *Let F be in CNF and $H = C_1 \wedge \cdots \wedge C_t$ be a conjunction of clauses. Then*

$$\#\mathrm{SAT}(F \wedge \neg H) = \sum_{i=1}^{t} \#\mathrm{SAT}\big(F \wedge C_1 \wedge \cdots \wedge C_{i-1} \wedge \neg C_i\big).$$

Proof. We show that

$$F \wedge \neg H = \bigvee_{i=1}^{t} \Big(F \wedge C_1 \wedge \cdots \wedge C_{i-1} \wedge \neg C_i\Big).$$

and that the formulae in the disjunction are pairwise exclusive. Any assignment κ satisfying $F \wedge \neg H$ satisfies at least one $\neg C_i$, therefore κ must also satisfy at least one of the formulas in the disjunction. Also, since $\neg H$ is satisfied there exists a unique smallest index i such that $\kappa(\neg C_i) = 1$ and $\kappa(C_1) = 1, \ldots, \kappa(C_{i-1}) = 1$. Further, it holds that κ satisfies exactly the i-th formula as for all subsequent formulae we have $\kappa(C_i) = 0$, and thus $\kappa(F \wedge C_1 \wedge \cdots \wedge C_i \wedge \ldots \neg C_j) = 0$ for $j > i$. Therefore the formulae in the disjunction are mutually exclusive, no model is counted twice and the model count of $F \wedge \neg H$ equals the sum on the right-hand side. □

Example. Consider a formula F in CNF such that the only clauses in F containing the variable a are $(\bar{a} \vee l_1 \vee l_2 \vee l_3)$, $(\bar{a} \vee l_4 \vee l_5 \vee l_6)$, and $(a \vee r_1 \vee r_2 \vee r_3)$. Branching on a yields the two subformulas $F_{-a} \wedge (r_1 \vee r_2 \vee r_3)$ and $F_{-a} \wedge (l_1 \vee l_2 \vee l_3) \wedge (l_4 \vee l_5 \vee l_6)$.

By complementary counting and using that $\neg(l) = (\bar{l})$ for unit clauses we obtain:

$$\#\big(F_{-a} \wedge (r_1 \vee r_2 \vee r_3)\big) = \#(F_{-a}) - \#\big(F_{-a} \wedge \bar{r}_1 \wedge \bar{r}_2 \wedge \bar{r}_3\big).$$

For the other branch, let $C_1 = (l_1 \vee l_2 \vee l_3)$ and $C_2 = (l_4 \vee l_5 \vee l_6)$. Then

$$\#\big(F_{-a} \wedge C_1 \wedge C_2\big) = \#(F_{-a}) - \#\big(F_{-a} \wedge \neg(C_1 \wedge C_2)\big).$$

Applying the DNF branching lemma to $\neg(C_1 \wedge C_2)$ gives

$$\#\big(F_{-a} \wedge \neg(C_1 \wedge C_2)\big) = \#\big(F_{-a} \wedge \neg C_1\big) + \#\big(F_{-a} \wedge C_1 \wedge \neg C_2\big)$$

which is equivalent to

$$\#\big(F_{-a} \wedge \bar{l}_1 \wedge \bar{l}_2 \wedge \bar{l}_3\big) + \#\big(F_{-a} \wedge (l_1 \vee l_2 \vee l_3) \wedge \bar{l}_4 \wedge \bar{l}_5 \wedge \bar{l}_6\big).$$

In the above example we obtain a branching number of $\tau(1,4,4,4) \approx 1.6581$ by computing $\#\mathrm{SAT}(F_{-a})$ and assigning the variables in the unit clauses in the other three branches. Branching in a similar way on variables of low degree appearing in longer clauses will be the key ingredient to designing a fast algorithm for instances with low density.

It should be noted that the proofs of Lemma 1 and Lemma 2 use only that we partition assignments into disjoint subsets and add their respective counts. Evaluating the weights of literals (including negated ones) does not affect the asymptotic complexity of the algorithm. Hence both lemmas naturally extend to weighted model counting.

4 An Algorithm for Sparse #SAT

Properties of Branching Numbers. The following two lemmas provide bounds on branching numbers that appear in our algorithms. The proofs are provided in the appendix.

Lemma 3. *Let τ_k denote the branching number of the vector $(1, 2, \ldots, k)$, $k \geq 3$. Then*

$$2^{1-2^{-k}} \leq \tau_k \leq 2^{1-\frac{1}{2\ln 2}2^{-k}}.$$

Lemma 4. *Let $d \geq 3$ and $k \geq \log(2d)$. For the branching vector $(1, k{+}1, \ldots, k{+}1)$ with d copies of $k{+}1$, the branching number τ satisfies*

$$\tau \leq 2^{1-\frac{1}{2d}}.$$

4.1 An Algorithm for Instances of Bounded Degree

In Algorithm 1 we present #BoundedSAT, an algorithm for counting the number of satisfying assignments for Boolean CNF formulas of bounded degree Δ. The key insight is that if all variables have degree at most Δ then we can always obtain a branching number $\tau_\Delta < 2$.

If there is a clause of length at most k, we branch on it (short clause branching). This yields the branching vector $(1, 2, \ldots, k)$ whose branching number is strictly less than 2 for every fixed k. If no short clause exists then we can apply complementary counting by branching on a variable a of bounded degree at most Δ which appears only in clauses of length at least $k+1$. By choosing an appropriate $k = k(\Delta)$ we obtain

$$\#\mathrm{SAT}(F) = 2 \cdot \#\mathrm{SAT}(F_{-a}) - (\#\mathrm{SAT}(F_{-a} \wedge \neg F_a[a=0]) + \#\mathrm{SAT}(F_{-a} \wedge \neg F_a[a=1])),$$

and expand each negated term with DNF branching. Since all clauses have length at least $k+1$ in this case, each DNF branch fixes at least k additional literals. This leads to a branching vector of the form $(1, k+1, \ldots, k+1)$ with d k-branches, which has branching number < 2 for suitable choices of $k = k(\Delta)$ and $d = d(\Delta)$.

Theorem 1. *The number of satisfying assignments of a CNF formula over n variables of maximum variable degree Δ can be determined in time $\mathcal{O}^*(2^{(1-\frac{1}{3D})n})$ where $D = \max(\Delta, 3)$.*

Algorithm 1 #BoundedSAT

Input: CNF formula F over n remaining variables, clause length k
Output: $\#\mathrm{SAT}(F)$

if F contains an empty clause **then**
  return 0
end if
if F has no clauses **then**
  return 2^n
end if
if F contains a DNF term from complementary counting **then**
  Branch as described in Lemma 2
end if
if there exists a clause C with $|C| \leq k$ **then**
  Apply short clause branching on C, obtaining subinstances $F_1, \ldots, F_{|C|}$
  return $\sum_{i=1}^{|C|} \#\mathrm{BoundedSAT}(F_i, n-i, k)$
else ▷ `complementary counting`
  Select a variable a of minimum degree
  return $2 \cdot \#\mathrm{BoundedSAT}(F_{-a}, n-1, k) -$
    $\#\mathrm{BoundedSAT}(F_{-a} \wedge \neg F_a[a=1], n-1, k) -$
    $\#\mathrm{BoundedSAT}(F_{-a} \wedge \neg F_a[a=0], n-1, k)$
end if

Proof. The algorithm recursively solves the problem using backtracking and uses polynomial space in the input size. We show that for a suitable choice of the clause length parameter k, #BoundedSAT solves the #SAT problem in the given time. Let $k = \log(2D)$ where $D = \max(\Delta, 3)$. We consider two cases.

Case 1: If there exists a clause C with $|C| \leq k$, then according to Lemma 3, clause branching yields a branching number $\tau = 2^{1-\frac{1}{4\ln 2 \cdot D}} < 2^{1-\frac{1}{3D}}$.

Case 2: Otherwise, all clauses have length at least $k+1 = \log(2D)+1$. Select a variable a of degree at most $\Delta \leq D$. All clauses containing a have length at least $k+1$, so by complementary counting and DNF branching we obtain a $(1, k+1, \ldots, k+1)$-branching with at most D branches. Since $D \geq 3$ and $k = \log(2D)$, Lemma 4 applies and gives a branching number of $2^{1-\frac{1}{2D}}$.

In both cases the branching number is at most $2^{1-\frac{1}{3D}}$. □

4.2 Extending to Constant Clause Density

We extend the algorithm for maximum degree to handle formulas of low clause density δ as follows. While there exists a variable v of degree at least $4d$, for $d = \max(\delta, 3)$, we branch on v. Otherwise, the formula has maximum degree at most $4d$ and we apply #BoundedSAT.

The intuition for the complexity analysis of the above approach is as follows. If we branch on balanced variables that appear as at least $2d$ positive and at least $2d$ negative literals, we can bound the size of tree to 2^ℓ such that $\ell \leq n/2$, and thus the total running time to $\mathcal{O}^*(2^{\ell+(1-\frac{1}{12d})(n-\ell)}) = \mathcal{O}^*(2^{(1-\frac{1}{24d})n})$. But if we branch only on pure variables of degree more than $4d$, we decrease the number of clauses in only one of the branches. Therefore, we introduce a new complexity measure that tracks progress in both variables and clauses.

Theorem 2. *Let F be a CNF formula over n variables and m clauses with clause density $\delta = m/n \geq 1$. There exists a constant $c > 0$ such that the number of satisfying assignments of F can be determined in time $\mathcal{O}^*(2^{(1-\frac{1}{c\delta})n})$ and polynomial space.*

Proof. Define

$$\mu(F) = n + \frac{m}{4d}$$

Branching on a variable of degree $> 4d$ reduces μ by at least 1 in one child (we assign one variable) and by at least 2 in the other (we assign one variable and the deletion of $> 4d$ clauses adds at least one additional unit). Thus, a high-degree branching step gives a binary split in which μ decreases by at least 1 in one branch and at least 2 in the other. With $\delta = m/n \leq d$ we have

$$\mu(F) \leq n + \frac{\delta n}{4d} \leq \frac{5}{4}n.$$

Let S be the number of *units of progress* until we reach a leaf where all degrees are $\leq 4d$. Clearly, an upper bound is $S \leq \lceil \mu(F) \rceil \leq \frac{5}{4}n$.
Distinguish between left (assigning a variable) and right branchings (assigning a variable and removing $> 4d$ clauses). This gives us a (1,2)-splitting with respect to μ. Let P be a root to leaf path on which S units of progress have been made. Let t be the number of assigned variables on P, and ℓ and r be the number of left and right branchings on P. Observe that $t = \ell + r$ and $S = \ell + 2r$. Hence, $t = S - r$, and the number of leafs with t assigned variables and r right branches is $\binom{t}{r} = \binom{S-r}{r}$. Let $q = 2^{(1-\frac{1}{12d})}$. At each branch we run #BoundedSAT on $n - t = n - (S - r)$ variables. Thus, we can bound the total running time $T(n)$ to

$$T(n) = \sum_{r=0}^{S} \binom{S-r}{r} q^{n-(S-r)}$$

Let us write $U(S) = \sum_{r=0}^{\lfloor S \rfloor} \binom{S-r}{r} q^r$. Then we have $T(n) = q^{n-S} U(S)$.

Using Pascal's rule $\binom{S-r}{r} = \binom{S-1-r}{r} + \binom{S-1-r}{r-1}$ and a change of summation index, with some simple algebra we obtain that U is a Fibonacci-type recurrence:

$$U(0) = 1, \quad U(1) = 1,$$
$$U(S) = U(S-1) + qU(S-2) \ (S \geq 2).$$

For such a sequence, the characteristic equation is $x^2 - x - q = 0$ and it has roots $\lambda_\pm := \frac{1 \pm \sqrt{1+4q}}{2}$, see, e.g., [Graham *et al.*, 1989, Chap. 5.4]). Overall, we obtain

$$U(S) = \sum_{r=0}^{S} \binom{S-r}{r} q^r = \frac{\lambda_+^{S+1} - \lambda_-^{S+1}}{\sqrt{1+4q}}.$$

We have $1 < q < 2$, hence $1 < \lambda_+ < 2$ and $-1 < \lambda_- < 0$. Therefore for $S \geq 1$,

$$U(S) = \frac{\lambda_+^{S+1} - \lambda_-^{S+1}}{\sqrt{1+4q}} \leq \frac{2\lambda_+^{S+1}}{\sqrt{1+4q}}.$$

Define $\rho(q) := \frac{\lambda_+}{q} = \frac{1+\sqrt{1+4q}}{2q}$. Thus, for the total running time we obtain

$$T(n) = q^{n-S}U(S) \leq q^{n-S}\frac{2\lambda_+^{S+1}}{\sqrt{1+4q}} = q^n C\rho(q)^S$$

for some constant $C > 0$, and we have $T(n) = O\big((q\rho(q)^{\frac{5}{4}})^n\big)$. In the appendix we show that for $q = 2^{1-\frac{1}{12d}}$ we have $q\rho(q)^{5/4} \leq 2^{1-1/(73d)}$, thus the theorem holds for $c = 73$.

In addition, we should note that we pessimistically model every high-degree split by the minimal (1,2) progress in μ for branching only on pure variables of high degree. The presented proof technique extends to more balanced branching of the form $(1+i/4, 1+(4-i)/4)$ where $1 \leq i \leq 3$ clauses are eliminated in one branch and $4-i$ in the other. □

4.3 Discussion

The classes of #SAT instances for which sub-2^n algorithms are known are not many: fixed-width #k-SAT, very sparse formulas with $m < n$ [Lozinskii, 1992], and polynomial time algorithms for bounded treewidth CNFs [Bacchus *et al.*, 2003; Samer and Szeider, 2010]. Schuler's width-reduction [Schuler, 2005] applies to #SAT, but the best #k-SAT bounds come from branching on a clause of length k. The lower bound on the branching number in Lemma 3 shows that width reduction cannot yield a sub-2^n bound.

We are aware of only one algorithm designed for sparse #SAT, and it has a better running time than $O^*(2^n)$ only for very sparse instances with $\delta \leq 2.25$ [Laakkonen *et al.*, 2024]. We find this rather surprising given the long history of (polynomial or exponential space) exact deterministic algorithms for sparse instances for SAT [Arvind and Schuler, 2003; Dantsin and Wolpert, 2005; Wahlström, 2005; Calabro *et al.*, 2006; Peng and Xiao, 2023] and Max-SAT [Dantsin and Wolpert, 2006; Kulikov and Kutzkov, 2007; Scott and Sorkin, 2007; Golovnev and Kutzkov, 2014; Chen and Santhanam, 2015; Sakai *et al.*, 2017].

It should be noted that the Max-SAT algorithm by [Chen and Santhanam, 2015] applies to general sparse CNF formulas and can be adapted to #SAT to yield a running time $\mathcal{O}^*(2^{(1-\frac{1}{c\delta})n})$ with $c = 800$. The algorithm is essentially the classic DPLL splitting algorithm that greedily selects high-degree variables to branch on until only unit clauses remain. Using shrinkage analysis [Santhanam, 2010], one bounds the running time to $\mathcal{O}^*(2^{(1-\frac{1}{c\delta})n})$. Note that the greedy approach is similar to our constant density algorithm but there are some important differences. First, we use as a base case our bounded-degree algorithm that exploits specific #SAT properties, and obtain much smaller constant for c. Of course, these constants capture worst cases that are very unlikely to occur in real instances but the gap highlights the contribution of our strong bounded-degree bound. Second, the Chen–Santhanam proof is based on a global probabilistic argument and offers structural insights about sparse CNF formulas but no direct guidance for the design of new #SAT algorithms. In contrast, our DNF branching and explicit complexity measure combining variables and clauses can be naturally translated into novel branching heuristics.

5 An Improved Upper Bound for #3-SAT

Complementary branching can be also used to improve the upper bound for #3-SAT. We build on the algorithm by [Kutzkov, 2007] which combines a reduction from sparse instances to #2-SAT and a careful branching case analysis for denser formulas. We extend complementary branching to a clause learning technique that enables the creation of new 2-clauses and strengthen the tight cases in this case analysis.

Solving sparse #3-SAT instances. The algorithm of [Kutzkov, 2007] for instances of average degree at most 5, let us call it #3-SAT-SPARSE, branches on variables of high degree in 3-clauses and then solves the resulting #2-SAT instances. Substituting the current best #2-SAT upper bound of $\mathcal{O}^*(1.2377^n)$ [Wahlström, 2008] into the analysis of #3-SAT-SPARSE yields the following lemma.

Lemma 5. *A #3-SAT instance over n variables with density less than or equal to 5/3 can be solved in $\mathcal{O}^*(1.637^n)$.*

5.1 The Complexity Measure

For formulas with density $> 5/3$, we use the complexity measure from [Kutzkov, 2007]: $\mu(F) = n - \lambda(F_2)$, where $\lambda(F_2)$ is a positive weight for the subformula of F consisting of 2-clauses. The set of all 2-clauses is divided into three subsets: $\mathcal{S}$ (for single), $\mathcal{T}$ (for two) and $\mathcal{R}$ (for rest). $\mathcal{S}$ is built from arbitrary 2-clauses in F_2, so that no variable in the $\mathcal{S}$-clauses occurs more than once, $\mathcal{T}$ may contain at most two 2-clauses and the rest are in $\mathcal{R}$. The $\mathcal{S}$-clauses and $\mathcal{T}$-clauses are weighted with $\varepsilon > 0$. The $\mathcal{R}$-clauses are not weighted. For $\varepsilon < 1$ it holds $\lambda(F_2) = (|\mathcal{S}|+|\mathcal{T}|)\varepsilon \leq (n/2+2)\varepsilon < n/2+2$, thus the complexity $\mu(F) = 0$ implies there are no more than constant number of unassigned variables in the formula.

5.2 Complementary Clause Learning

Lemma 6. *Let F be a CNF formula and let a be a variable that occurs only positively in F. Write $F = F_{-a} \wedge F_a$, where F_a is the conjunction of all clauses containing a, and let H be any CNF formula equivalent to $\neg F_a[a=0]$. Then*

$$\#\mathrm{SAT}(F) = 2\#\mathrm{SAT}(F[a=0]) + \#\mathrm{SAT}\big(F[a=1] \wedge H\big)$$

Proof. Denote $G := F_a[a=0]$. Branching on a gives

$$F[a=1] = F_{-a}, \qquad F[a=0] = F_{-a} \wedge G.$$

Hence

$$\#\mathrm{SAT}(F) = \#\mathrm{SAT}(F[a=1]) + \#\mathrm{SAT}(F[a=0]) = \#\mathrm{SAT}(F_{-a}) + \#\mathrm{SAT}(F_{-a} \wedge G).$$

Apply complementary counting (Lemma 1) to the instance F_{-a} with constraint G:

$$\#\mathrm{SAT}(F_{-a}) = \#\mathrm{SAT}(F_{-a} \wedge G) + \#\mathrm{SAT}(F_{-a} \wedge \neg G).$$

By assumption H is a CNF equivalent to $\neg G$, so

$$\#\mathrm{SAT}(F_{-a}) = \#\mathrm{SAT}(F_{-a} \wedge G) + \#\mathrm{SAT}(F_{-a} \wedge H).$$

Substituting into the expression for $\#\mathrm{SAT}(F)$ yields

$$\#\mathrm{SAT}(F) = \big(\#\mathrm{SAT}(F_{-a} \wedge G) + \#\mathrm{SAT}(F_{-a} \wedge H)\big) + \#\mathrm{SAT}(F_{-a} \wedge G),$$

which simplifies to

$$\#\mathrm{SAT}(F) = 2\#\mathrm{SAT}(F_{-a} \wedge G) + \#\mathrm{SAT}(F_{-a} \wedge H) = 2\#\mathrm{SAT}(F[a=0]) + \#\mathrm{SAT}(F[a=1] \wedge H)$$ □

Algorithm 2 #3-SAT-General

Input: 3-CNF formula F
Output: $\#\mathrm{SAT}(F)$

if F contains an empty clause **then**
  return 0
end if
if F has no clauses **then**
  return 2^n
end if
if density $\delta(F) \leq 5/3$ **then**
  return #3-SAT-SPARSE(F) ▷ `Lemma 5`
end if
Maintain $F_2, F_3, \mathcal{S}, \mathcal{T}$ as in the analysis
if $F_2 = \emptyset$ **then** ▷ `Case 0`
  Let x be a max-degree variable in F_3
  Branch on x and update $F_2, F_3, \mathcal{S}, \mathcal{T}$
  return sum of recursive calls
if $F_2 \subseteq \mathcal{S}$ and $\mathcal{T} = \emptyset$ **then** ▷ `Case 1`
  Choose x of max degree in F_2 and branch on x
  (in degree-2 patterns use Lemma 6)
  return sum of recursive calls
if $\mathcal{T}$ full and all $d_2(x) \leq 2$ **then** ▷ `Case 2`
  If $(a \vee b), (a \vee c)$ only, apply Lemma 6
  Else choose x with $d_2(x) = 2$ such that
    the number of deleted 2-clauses is minimized
  Branch on x and recurse
if $\exists x$ with 3 occurrences in $\mathcal{S} \cup \mathcal{T}$ **then** ▷ `Case 3`
  Branch on x and recurse

Observe that the above lemma implies that for a pure literal a we mimic branching in the standard way on a. However, we "learn" the CNF H from $\neg F_a[a = 0]$ and instead of $F[a = 1]$ we consider $F_{-a} \wedge H$ which is particularly useful if a has a low degree. For example, let $F_a = (a \vee b) \wedge (a \vee x_1 \vee x_2)$. Then $H = (\bar{b} \vee \bar{x}_1) \wedge (\bar{b} \vee \bar{x}_2)$, and by adding the two 2-clauses we decrease the complexity $\mu(F[a = 1])$ by 2ε.

5.3 The Algorithm

In Algorithm 2 we present the #3-SAT algorithm which considers different cases depending on the maximum degree of the formula and the cardinality of the 2-clauses sets $\mathcal{S}$ and $\mathcal{T}$. F_k denotes the subformula of F consisting of k-clauses.

Case 0. The number of added 2-clauses is at least six. We analyze only the worst case when all six 2-clauses are added in one branch. It is easy to verify that the more symmetric the complexity decrease in both branches, the smaller the branching number. The analysis for the other cases can be immediately concluded from this one. If variable b occurs $3 \leq k \leq 5$ times in the added 2-clauses then we branch on b and the recursion is computed to $\tau(1, 2 + (6 - k)\varepsilon, k + 2)$. For $k = 6$ we obtain six 2-clauses that all share the same variable b. If b is not a pure literal, then immediately branching on b would yield a branching number of at least $\tau(1, 3, 7) < 1.52$. Otherwise, the literal b is dominated by the literal a, or vice versa, such that by assigning $a = 1$, respectively $b = 1$, there are no more appearances of b, respectively a. Thus, the branching is at least $\tau(1, 2) < 1.62$.

Therefore we conclude that no variable occurs more than twice in the added six 2-clauses. First of all we observe that each four of the considered 2-clauses will fit in $\mathcal{S}$ and $\mathcal{T}$ as there always exist two variable-disjoint 2-clauses which can be set in $\mathcal{S}$. If we have the 2-clauses $(l_1 \vee l_2), (\bar{l}_1 \vee l_3)$ we easily obtain $\tau(1, 3 + 3\varepsilon, 3 + 3\varepsilon)$. Let's now consider the case we have 2-clauses $(l_1 \vee l_2), (l_1 \vee l_3)$. If l_2 or l_3 does not occur in another of the six 2-clauses or l_2 and l_3 are in the same 2-clause we branch on l_1. As per [Kutzkov, 2007], such variables of degree two are called *suitable* variables. The corresponding recursion equation is computed to $\tau(1, 2+4\varepsilon, 4+3\varepsilon)$. If there exists no suitable variable and not all the six 2-clauses fit in $\mathcal{S}$ and $\mathcal{T}$, we conclude a "deadlock" situation, i.e. six variables forming six 2-clauses so that each variable occurs twice: e.g. $(l_1 \vee l_2), (l_3 \vee l_4), (l_1 \vee l_3), (l_2 \vee l_5), (l_4 \vee l_6), (l_5 \vee l_6)$. In this configuration we distinguish two subcases. If some variable, say l_1, appears only in its two 2-clauses $(l_1 \vee l_2)$ and $(l_1 \vee l_3)$ and nowhere else in the formula, then by Lemma 6 we can apply complementary counting to the l_1-branch and obtain an additional 2-clause of the form $(\bar{l}_2 \vee \bar{l}_3)$ in the $l_1 = 1$ branch. Branching subsequently on l_2 in the $l_1 = 1$ branch yields a branching number stronger than $\tau(2, 3, 3) < 1.53$. Otherwise every l_i has an additional occurrence outside these six 2-clauses, and branching on such a variable immediately yields a recursion of at least $\tau(1 - 2\varepsilon, 3 - 3\varepsilon)$.

If none of the above applies, all six 2-clauses can be added to $\mathcal{S} \cup \mathcal{T}$ and weighted with ε. Thus, we compute the branching number to $\tau(1, 1 + 6\varepsilon)$.

Case 1. The goal in this case is to show that we have a worst-case recursion of $\tau(1-\varepsilon, 2+2\varepsilon)$. Let $(a \vee b)$ be a 2-clause in F_2. If one of a, b (say a) occurs only in this clause, we branch on the other variable and obtain a symmetric $(2 - \varepsilon, 2 - \varepsilon)$-branching. If a has degree 2 in F, then besides $(a \vee b)$ it appears in a single additional clause, say $(a \vee \ell_1 \vee \ell_2)$. By Lemma 6 we can apply complementary counting in the $a = 1$ branch and add 2-clauses $(\bar{b} \vee \bar{\ell}_1)$ and $(\bar{b} \vee \bar{\ell}_2)$. This yields a branching number of at least $\tau(1, 2) < 1.62$.

If both a and b have degree 3, we branch a and in the $a = 1$ branch we can use complementary counting on the two 3-clauses containing b. Let these be $(b \vee l_1 \vee l_2), (b \vee l_3 \vee l_4)$. In the general case, branching on b and applying Lemma 6, results in a branching number of $\tau(1 + 2\varepsilon, 1 + 4\varepsilon)$, thus an overall branching of $\tau(2 + \varepsilon, 2 + 3\varepsilon, 2 + \varepsilon)$. If we cannot add four 2-clauses, then we have a specific structure such as a shared variable among the l_i variables or 2-clauses in $\mathcal{S}$ in which at least two of the l_i's appear. In all these cases we can either assign one more variable or have a subsequent strong branching assigning 2 variables in each branch.

Finally, if at least one of a, b has degree at least 4, then branching on that variable creates three new 2-clauses in one branch. If we cannot add the three 2-clauses to $\mathcal{S} \cup \mathcal{T}$, then we can have a subsequent (1,3)-branching. We then obtain a branching number at least $\tau(1 - \varepsilon, 3, 5 - 2\varepsilon)$. Otherwise, the branching number is the desired $\tau(1 - \varepsilon, 2 + 2\varepsilon)$.

Case 2. The case when a literal and its negation both occur in F_2 has an obvious bound of $\tau(2 - 3\varepsilon, 2 - 3\varepsilon)$ since the setting of a variable deletes at most two 2-clauses and one of the 2-clauses is shared by the assigned variables. Otherwise

we have a literal which occurs twice; w.l.o.g. consider the 2-clauses $(a \vee b)$ and $(a \vee c)$. If a appears in an additional clause (either a 2-clause or a 3-clause), then branching on a yields a recursion of at least $\tau(1-2\varepsilon, 3-3\varepsilon)$, since in one branch we delete two weighted 2-clauses and in the other we gain (at least) an extra weighted 2-clause from the additional occurrence. If a appears only in $(a \vee b)$ and $(a \vee c)$, we apply Lemma 6 with $G = b \wedge c$ to replace the $a = 0$ branch with the complementary instance $F_{-a} \wedge (\bar{b} \vee \bar{c})$. This guarantees in the complementary branch there is a newly learned weighted 2-clause, so the recursion is bounded by $\tau(1-\varepsilon, 3-4\varepsilon)$.
Case 3. The recursion equation $\tau(1-3\varepsilon, 4-5\varepsilon)$ is trivial since we delete all $\mathcal{T}$-clauses.

Theorem 3. *The #3-SAT problem on formulas over n variables can be solved in time $\mathcal{O}^*(1.637^n)$.*

Proof. For instances of density at most $5/3$, Lemma 5 gives a running time of $\mathcal{O}^*(1.637^n)$. It therefore suffices to consider formulas of density $> 5/3$ and analyze the algorithm #3SAT-General using the complexity measure $\mu(F) = n - \lambda(F_2)$.

The case analysis above shows that for every recursive call the measure μ decreases according to one of the following branching vectors:

- Case 0 (no 2-clauses) yields $\tau(1, 2{+}3\varepsilon, 5)$, $\tau(1, 2{+}2\varepsilon, 6)$, $\tau(1, 2{+}\varepsilon, 7)$, $\tau(1, 3{+}3\varepsilon, 3{+}3\varepsilon)$, $\tau(1, 2{+}4\varepsilon, 4{+}3\varepsilon)$, and $\tau(1, 1{+}6\varepsilon)$.
- Case 1 ($F_2 \subseteq \mathcal{S}, \mathcal{T} = \emptyset$) yields $(2-\varepsilon, 2-\varepsilon)$, $\tau(2+\varepsilon, 2+3\varepsilon, 2+\varepsilon)$, $\tau(1{-}\varepsilon, 2{+}2\varepsilon)$ and $\tau(1-\varepsilon, 3, 5-2\varepsilon)$.
- Case 2 (a literal of degree two in F_2) yields $\tau(2{-}3\varepsilon, 2{-}3\varepsilon)$, $\tau(1{-}2\varepsilon, 3{-}3\varepsilon)$ and $\tau(1{-}\varepsilon, 3{-}4\varepsilon)$.
- Case 3 (a variable of degree three in $\mathcal{S} \cup \mathcal{T}$) yields $\tau(1{-}3\varepsilon, 4{-}5\varepsilon)$.

Fix $\varepsilon = 0.1545$. The numerical evaluation of the branching numbers listed above shows that the maximum is at most 1.6343, and the tight cases are $\tau(1, 2{+}4\varepsilon, 4{+}3\varepsilon)$, $\tau(1, 1{+}6\varepsilon)$, and $\tau(1{-}2\varepsilon, 3{-}3\varepsilon)$. Therefore, the total running time of Algorithm 2 on formulas of density $> 5/3$ is bounded by $\mathcal{O}^*(1.6343^n)$. Combining this with #3-SAT-SPARSE from Lemma 5 gives an overall running time of $\mathcal{O}^*(1.637^n)$ for #3-SAT. □

5.4 Discussion

There is a long history of exact deterministic algorithms for #2-SAT and #3-SAT, with running times analyzed with respect to the number of variables or the number of clauses [Zhang, 1996; Dahllöf *et al.*, 2005; Fürer and Kasiviswanathan, 2007; Kutzkov, 2007; Wahlström, 2008; Zhou *et al.*, 2010; Peng *et al.*, 2025]. When parameterized by the number of clauses, the best known bounds are $\mathcal{O}^*(1.1082^m)$ for #2-SAT and $\mathcal{O}^*(1.4423^m)$ for #3-SAT.

Note that an improved upper bound for #2-SAT with respect to the number of variables would improve the bound $\mathcal{O}^*(1.637^n)$ for sparse #3-SAT instances and thus yield a better overall bound for #3-SAT.

Randomized algorithms can achieve an e^{ε}-approximation of the model count with stronger exponential bases than our deterministic bound, but incur an additional ε^{-2} factor in running time [Thurley, 2012; Schmitt and Wanka, 2013].

Inst Nr	m	n	Qual (%)	Avg τ
197	4.9M	6.4M	44.3	1.380
193	1.8M	2.2M	56.9	1.381
195	806k	1.1M	26.4	1.386
199	751k	980k	42.0	1.380
043	3,549	10k	2.9	1.548
007	2,575	1,441	2.0	1.497
041	1,910	1,920	34.1	1.544

Table 1: MC2025 Track 1 instances benefiting from complementary branching. Variables with degree ≤ 5 appearing only in clauses of length ≥ 4 qualify. Shown are all instances where $\geq 1\%$ of variables qualify.

6 Conclusions and Future Directions

We introduced complementary branching, a novel branching paradigm for model counting that naturally complements the classical DPLL branching. Using this technique, we designed new algorithms for sparse #SAT instances and obtained a new record running time for #3-SAT. These results demonstrate that complementary branching is a powerful and versatile tool for designing faster exact algorithms for model counting.

While our results are theoretical, a major research direction is to leverage the introduced complementary branching techniques in modern #SAT solvers. We analyzed 200 MC2025 benchmark instances [Fichte and Hecher, 2025] for weighted and unweighted model counting benefits. Below is a summary of our findings, details can be found in the appendix:

- **Complementary clause branching** as described in Section 3. In about 8% of the instances less than 1% of the clauses contribute to more than 10% of the overall clause length. In 11% of the instances there are clauses with hundreds of variables and removing such long clauses early on might yield structural benefits.
- **DNF branching** as described in Lemma 2. We observe that 15 instances in the two tracks will directly benefit from DNF branching. We search for variables of degree at most 5 that appear only in clauses of length at least 4, and compute the branching numbers for these variables using DNF branching, see Lemma 2. Table **??** shows a list of instances that have $\geq 1\%$ of such variables, achieving average branching numbers $\tau \approx 1.38$. Notably, the largest test instances benefit the most.
- **Clause learning** as described in Lemma 6. Pure variables are abundant among the test instances, and in many instances there is a significant number of pure variables of low degree. We computed that for 20 out of 200 instances more than 10% of the variables are pure with degree below the median clause length, allowing us to create many new short clauses. These new clauses can thus further shrink the search space and guide #SAT solvers.

Of course, the above is just a first glimpse into the potential benefits of complementary branching, and one needs to study how the technique can be combined with other methods. In particular, DNF branching and clause learning are likely to be more beneficial at deeper levels of the branching tree where even more low-degree variables emerge.

References

[Arvind and Schuler, 2003] Vikraman Arvind and Rainer Schuler. The quantum query complexity of 0-1 knapsack and associated claw problems. In *Algorithms and Computation, 14th International Symposium, ISAAC*, pages 168–177, 2003.

[Bacchus *et al.*, 2003] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and Complexity Results for #SAT and Bayesian Inference. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2003.

[Beck *et al.*, 2020] Gabrielle Beck, Maximilian Zinkus, and Matthew Green. Automating the development of chosen ciphertext attacks. In *29th USENIX Security Symposium*, pages 1821–1837, 2020.

[Belle, 2017] Vaishak Belle. Open-universe weighted model counting. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3701–3708, 2017.

[Calabro *et al.*, 2006] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006)*, pages 252–260, 2006.

[Calabro *et al.*, 2009] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85, 2009.

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

[Chen and Santhanam, 2015] Ruiwen Chen and Rahul Santhanam. Improved Algorithms for Sparse MAX-SAT and MAX-k-CSP. In *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 33–45, 2015.

[Cook, 1971] Stephen A Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[Dahllöf *et al.*, 2005] Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2SAT and 3SAT formulae. *Theoretical Computer Science*, 332(1):265–291, 2005.

[Dantsin and Wolpert, 2005] Evgeny Dantsin and Alexander Wolpert. An Improved Upper Bound for SAT. In *Theory and Applications of Satisfiability Testing*, pages 400–407, 2005.

[Dantsin and Wolpert, 2006] Evgeny Dantsin and Alexander Wolpert. MAX-SAT for Formulas with Constant Clause Density Can Be Solved Faster Than in $O(2^n)$ Time. In *Theory and Applications of Satisfiability Testing - SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 266–276, 2006.

[Dantsin *et al.*, 2002] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Int. Res.*, 17(1):229–264, 2002.

[Fichte and Hecher, 2025] Johannes K. Fichte and Markus Hecher. The model counting competitions 2021-2023, 2025.

[Fichte *et al.*, 2021] Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *ACM J. Exp. Algorithmics*, 26, 2021.

[Fürer and Kasiviswanathan, 2007] Martin Fürer and Shiva Prasad Kasiviswanathan. Algorithms for counting 2-sat solutions and colorings with applications. In *Algorithmic Aspects in Information and Management*, pages 47–57, 2007.

[Golovnev and Kutzkov, 2014] Alexander Golovnev and Konstantin Kutzkov. New exact algorithms for the 2-constraint satisfaction problem. *Theoretical Computer Science*, 526:18–27, 2014.

[Graham *et al.*, 1989] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1989.

[Kulikov and Kutzkov, 2007] Alexander S. Kulikov and Konstantin Kutzkov. New bounds for MAX-SAT by clause learning. In *CSR 2007*, volume 4649 of *Lecture Notes in Computer Science*, pages 194–204, 2007.

[Kutzkov, 2007] Konstantin Kutzkov. New upper bound for the #3-SAT problem. *Information Processing Letters*, 105(1):1–5, 2007.

[Laakkonen *et al.*, 2024] Tuomas Laakkonen, Konstantinos Meichanetzidis, and John van de Wetering. A graphical #SAT algorithm for formulae with small clause density. In Proceedings of the 21st International Conference on *Quantum Physics and Logic*, volume 406 of *Electronic Proceedings in Theoretical Computer Science*, 2024.

[Lozinskii, 1992] Eliezer L Lozinskii. Counting propositional models. *Information Processing Letters*, 41(6):327–332, 1992.

[Manhaeve *et al.*, 2021] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298:103504, 2021.

[Paturi *et al.*, 1997] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 566–574. IEEE, 1997.

[Peng and Xiao, 2023] Junqiang Peng and Mingyu Xiao. Fast Algorithms for SAT with Bounded Occurrences of

Variables. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 2004–2012, 2023.

[Peng *et al.*, 2025] Junqiang Peng, Zimo Sheng, and Mingyu Xiao. New algorithms for #2-SAT and #3-SAT. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 2666–2674, 2025.

[Sakai *et al.*, 2017] Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. Improved exact algorithms for mildly sparse instances of Max SAT. *Theoretical Computer Science*, 697:58–68, 2017.

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

[Santhanam, 2010] Rahul Santhanam. Fighting Perebor: New and Improved Algorithms for Formula and QBF Satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 183–192, 2010.

[Schmitt and Wanka, 2013] Manuel Schmitt and Rolf Wanka. Exploiting independent subformulas: A faster approximation scheme for #k-SAT. *Information Processing Letters*, 113(9):337–344, 2013.

[Schuler, 2005] Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, 54(1):40–44, 2005.

[Scott and Sorkin, 2007] Alexander D. Scott and Gregory B. Sorkin. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optimization*, 4(3):260–287, 2007.

[Speck *et al.*, 2025] David Speck, Markus Hecher, Daniel Gnad, Johannes K. Fichte, and Augusto B. Corrêa. Counting and reasoning with plans. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):26688–26696, 2025.

[Thurley, 2012] Marc Thurley. An approximation algorithm for #k-SAT. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, pages 78–87, 2012.

[Valiant, 1979] Leslie G Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[Wahlström, 2005] Magnus Wahlström. An algorithm for the SAT problem for formulae of linear length. In *Algorithms - ESA 2005, 13th Annual European Symposium*, volume 3669 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2005.

[Wahlström, 2008] Magnus Wahlström. A tighter bound for counting max-weight solutions to 2-SAT instances. In *Parameterized and Exact Computation: Third International Workshop, IWPEC 2008*, pages 202–213, 2008.

[Zhang, 1996] Wenhui Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 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 *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 217–222, 2010.