New Algorithms for Parity-SAT and Its Bounded-Occurrence Versions

Sanjay Jain
School of Computing, National University of Singapore, Singapore

Junqiang Peng[1]
University of Electronic Science and Technology of China, Chengdu, China

Frank Stephan
Department of Mathematics, National University of Singapore, Singapore
School of Computing, National University of Singapore, Singapore

Haoyun Tang
School of Computing, National University of Singapore, Singapore

Mingyu Xiao
University of Electronic Science and Technology of China, Chengdu, China

Abstract

Parity-SAT is the problem of determining whether a given CNF formula has an odd number of satisfying assignments. As a canonical $\oplus$P-complete problem, it represents a fundamental variant of the exact model counting problem (#SAT). Under the Strong Exponential Time Hypothesis (SETH), Parity-SAT admits no $O^*((2-\varepsilon)^n)$-time or $O^*((2-\varepsilon)^m)$-time algorithm for any constant $\varepsilon > 0$, where n and m denote the numbers of variables and clauses, respectively. Thus, breaking the 2^n or 2^m barrier appears impossible in full generality.

In this work, we revisit this barrier through structural restrictions and a refined exploitation of parity. We study Parity-d-occ-SAT, where each variable appears in at most d clauses, and obtain three main results. First, we design a randomized $O^*(2^{m(1-1/O(d))})$-time algorithm, thereby breaking the 2^m barrier for every fixed d. Second, for the special case $d = 2$, we develop a significantly sharper branching algorithm running in $O^*(1.1193^n)$ time or $O^*(1.3248^m)$ time. Third, leveraging the structural insights underlying the $d = 2$ case, we obtain an $O^*(1.1052^L)$-time algorithm for general Parity-SAT, where L denotes the formula length. All algorithms use only polynomial space. Notably, our running-time bounds are better than the best known bounds for the corresponding exact counting counterparts, highlighting a genuine algorithmic advantage of parity over counting. Conceptually, our results demonstrate that parity admits finer structural reductions and more efficient branching than exact model counting, and that bounded occurrence can be systematically leveraged to circumvent classical exponential barriers.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Parity-SAT, Exact Exponential Algorithms

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.20

Related Version *Full Version*: https://arxiv.org/abs/2605.14093

Funding Junqiang Peng and Mingyu Xiao were supported by the National Natural Science Foundation of China, under the grants 62372095 and 62502078. Sanjay Jain and Frank Stephan were supported by Singapore Ministry of Education (MOE) AcRF Tier 2 grant MOE-000538-01. Additionally, Sanjay Jain was supported by NUS grant E-252-00-0021-01. Additionally, Frank Stephan was supported by AcRF Tier 1 grants A-0008454-00-00 and A-0008494-00-00.

[1] corresponding author

1 Introduction

The Satisfiability (SAT) problem, determining whether a given Conjunctive Normal Form (CNF) formula has a satisfying assignment, is a cornerstone of Computer Science and the prototypical NP-complete problem [13]. Its counting variant, Model Counting (#SAT), requires computing the exact number of satisfying assignments (called models). #SAT is #P-complete [44] and serves as a fundamental tool in probabilistic inference [38, 3, 39, 11], network reliability estimation [18], and explainable AI [32]. A significant modular variant of #SAT is Parity-SAT, which asks whether a formula has an odd number of models. Parity-SAT is the canonical complete problem for the complexity class $\oplus$P introduced by Papadimitriou and Zachos [33] and is shown to be NP-hard under randomized reductions by Valiant and Vazirani [46]. This problem also plays a pivotal role in the proof of Toda's theorem [42], a landmark result in the study of counting complexity.

Given the inherent hardness of these problems, research has shifted toward finding *moderately exponential-time* algorithms. Typically, instances are parameterized by the number of variables n, with the goal of achieving a running time of $O^*(c^n)$ for some constant $c < 2$. A brute-force search over 2^n assignments solves SAT, #SAT, and Parity-SAT in $O^*(2^n)$ time. However, it is not easy to break the trivial "2^n-barrier".

Under the Strong Exponential Time Hypothesis (SETH) [27], no $O^*((2-\varepsilon)^n)$ algorithm exists for SAT for any $\varepsilon > 0$. By the isolation lemma [8, 43], this barrier extends to Parity-SAT as well. These lower bounds have motivated the study of *restricted* formula classes, such as k-SAT (where clauses have size at most k) and d-occ-SAT (where variables appear in at most d clauses). For k-SAT, it has been extensively studied and we provide a review on it later in this section. In this paper, we focus on d-occ-SAT and denote its counting and parity counterparts by #d-occ-SAT and Parity-d-occ-SAT, respectively.

It is known that d-occ-SAT can be solved in $O^*(2^{n(1-1/O(\log d))})$ time [9, 16]. Remarkably, this upper bound carries over to the exact counting variants: #d-occ-SAT admits a deterministic algorithm running in $O^*(2^{n(1-1/O(\log d))})$ time, albeit requiring exponential space [10]. Naturally, this algorithm also applies to Parity-d-occ-SAT.

The computational landscape shifts significantly when instances are parameterized by the number of clauses m. While SAT admits an $O^*(1.2226^m)$-time algorithm [12], its parity (and counting) counterparts face a rigid barrier: although they can be solved in $O^*(2^m)$ time via Inclusion-Exclusion [28, 30], Cygan et al. [15] showed that this 2^m-barrier is insurmountable under SETH even for Parity-SAT. Moreover, for the bounded-occurrence restrictions, to the best of our knowledge, no known algorithms break the 2^m-barrier even for Parity-2-occ-SAT.

Our Results. In this work, we study fast algorithms for Parity-SAT and Parity-d-occ-SAT. First, we demonstrate that the 2^m-barrier is surmountable for the class of bounded-occurrence formulas by giving a randomzied $O^*(2^{m(1-1/O(d))})$-time algorithm for Parity-d-occ-SAT. Second, for the specific case of $d = 2$, we design a refined algorithm for Parity-2-occ-SAT that runs in $O^*(1.1193^n)$ time or $O^*(1.3248^m)$ time. Finally, utilizing the results and algorithmic ideas developed for Parity-2-occ-SAT, we obtain an $O^*(1.1052^L)$-time algorithm for the general Parity-SAT, where L is the formula length (i.e., sum of the sizes of clauses). We remark that the parameter L is also well-studied in the literature, as detailed in the related work discussion later in this section. All our algorithms require only polynomial space, and our bounds are better than the known results for their counting counterparts. This underscores that parity counting, being a special case of exact counting, potentially offers greater tractability. A summary of known results and our contributions is provided in Table 1.

Table 1 Summary of known results and our results. In this table, "Para." stands for "Parameter"; "det." and "rand." stand for deterministic and randomized algorithms, respectively; "poly-sp." and "exp-sp." denote polynomial and exponential space.

Para.	**#SAT**	**Parity-SAT**
L	$O^*(1.1082^L)$ det., exp-sp. [4, 5, 36]	$O^*(1.1052^L)$ det., poly-sp. [**Section 6**]

Para.	**#2-occ-SAT**	**#d-occ-SAT**	**Parity-2-occ-SAT**	**Parity-d-occ-SAT**
n	$O^*(1.2282^n)$ det., exp-sp. (derived by $L \leq 2n$)	$O^*(2^{n(1-1/O(\log d))})$ rand., poly-sp. [26] det., exp-sp. [10]	$O^*(1.1193^n)$ det., poly-sp. [**Section 5**]	$O^*(2^{n(1-1/O(\log d))})$ rand., poly-sp. [26] det., exp-sp. [10]
m	$O^*(2^m)$ det., poly-sp. [28, 30]		$O^*(1.3248^m)$ det., poly-sp. [**Section 5**]	$O^*(2^{m(1-1/O(d))})$ rand., poly-sp. [**Section 4**]

Our results mainly rely on the exploitation of parity. A simple example is that if an unassigned variable appears in no clauses, it acts as a free variable that doubles the total number of satisfying assignments; consequently, one can immediately infer that the parity of the formula is zero. Building upon this and other parity-specific properties, we develop several reduction rules and branching rules used in our algorithms. Furthermore, unlike most branching algorithms that solely branch on variables, we employ an additional clause-based branching rule, which may be of independent interest.

Other Related Works. It is known that k-SAT can be solved in $O^*(2^{n(1-1/O(k))})$ time [41, 35, 34]. For #k-SAT, the same runtime bound is achieved either deterministically with exponential space [10] or via a randomized polynomial-space algorithm [26]. Particular attention has also been given to small values of k. Specifically, 3-SAT admits a randomized $O^*(1.306973^n)$-time [40] and a deterministic $O^*(1.32793^n)$-time algorithm [29], and (weighted) #2-SAT is solvable in $O^*(1.2377^n)$ time [47] or $O^*(1.1082^m)$ time using exponential space [36].

Beyond n and m, the formula length L is another widely studied parameter. Currently, SAT and #SAT can be solved in $O^*(1.0638^L)$ [37] and $O^*(1.1082^L)$ [36] time, respectively. The latter bound is obtained by reducing a #SAT instance of length L to a weighted #2-SAT instance with $m = L$ clauses [4, 5] and calling the $O^*(1.1082^m)$ time algorithm for weighted #2-SAT [36]; while the reduction itself only uses polynomial space, the invoked algorithm [36] requires exponential space.

We also highlight that the parity counting versions of many classic problems have been extensively studied, including Perfect Matching [44], Hamiltonian Cycle [7], and k-Path [14]. Some of these problems admit polynomial-time algorithms, while others remain computationally hard. Consequently, parity counting problems have attracted significant attention from the perspectives of classical complexity theory, parameterized complexity, and fine-grained complexity (see e.g., [2, 45, 14, 23, 1, 6]).

Due to space limitations, proofs of lemmas marked with ♣ are deferred to the full version.

2 Preliminaries

Notations. Let V be a set of *Boolean variables*. A Boolean variable (or simply *variable*) $x \in V$ can be assigned a value of 1 (TRUE) or 0 (FALSE). A variable x has two corresponding *literals*: the positive literal x and the negative literal $\overline{x}$. We use $\overline{x}$ to denote the negation of literal x, so $\overline{\overline{x}} = x$. A *clause* on V is a set of literals on V. Note that a clause might be empty. A *CNF formula* (or simply *formula*) $\varphi = \{C_1, C_2, \ldots, C_m\}$ over V is a set of clauses on V, also denoted by $\varphi = \bigwedge_{i=1}^{m} C_i$. We denote by $\mathrm{var}(\varphi)$ the variable set of φ. For a literal ℓ, $\mathrm{var}(\ell)$ denotes its corresponding variable. For a clause C, $\mathrm{var}(C)$ denotes the set of variables appearing in the clause (i.e., $x \in \mathrm{var}(C)$ if $x \in C$ or $\overline{x} \in C$). We say a variable x *co-occurs* with a variable y if x and y share a clause.

An *assignment* for a 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 evaluates to 1 under σ. An assignment for $\mathrm{var}(\varphi)$ is called a *satisfying assignment* of φ if σ satisfies all clauses in φ. We use $\mathrm{Parity}(\varphi)$ to denote the parity of the number of satisfying assignments of φ (i.e., it is 1 if the number of satisfying assignments is odd, and 0 otherwise).

The *degree* of a variable x in formula φ is the total number of occurrences of literals x and $\overline{x}$ in φ. A variable is called a *d-variable* if its degree is exactly d, and a *d^+-variable* if its degree is at least d. A variable x is called (i, j)-variable if literal x appears i times and literal $\overline{x}$ appears j times.

The *length* of a clause C, denoted by $|C|$, is the number of literals in C. A clause is called a *k-clause* (resp. *k^--clause* and *k^+-clause*) if its length is exactly k (resp. at most k and at least k). A formula φ is called a *k-CNF formula* if each clause in φ has length at most k.

For a formula φ, define $n(\varphi) = |\mathrm{var}(\varphi)|$ as the number of variables, $m(\varphi) = |\varphi|$ as the number of clauses, and $L(\varphi) = \sum_{C \in \varphi} |C|$ as the total length of the formula. We may simply write n, m, and L if the context is clear.

Branch-and-Search. A *branch-and-search* algorithm first applies reduction rules to simplify the instance, and then recursively solves it via branching operations. To evaluate the time complexity, we associate each problem instance with a non-negative measure μ. Let $T(\mu)$ denote an upper bound on the number of leaves in the search tree generated by the algorithm for any instance with measure at most μ. A branching operation generates l subinstances whose solutions collectively determine the solution to the original instance, and the measure decreases by at least $a_i > 0$ in the i-th branch, yielding the recurrence relation

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

This recurrence is represented by a *branching vector* $(a_1, \ldots, a_l)$. The *branching factor* of this recurrence, denoted by $\tau(a_1, \ldots, a_l)$, is defined as the unique positive real root of the characteristic equation $1 - \sum_{i=1}^{l} x^{-a_i} = 0$. If the branching factor of every operation in the algorithm is bounded by a constant γ, then the overall size of the search tree is bounded by $O^*(\gamma^\mu)$. For a comprehensive background on exact exponential algorithms, we refer the reader to [21]. The following standard property allows us to analytically upper-bound branching factors, which is a direct consequence of Lemma 2.2 and Lemma 2.3 in [21].

▶ **Lemma 1.** *For any reals $a_1, a_2 > 0$ and $p > q > 0$, if $a_1 + a_2 \geq p$ and $\min(a_1, a_2) \geq q$, then $\tau(a_1, a_2) \leq \tau(p - q, q)$.*

3 Reduction Rules and Branching Rules

In this section, we introduce some basic reduction rules and branching rules that will be frequently used in algorithms in subsequent sections.

Before presenting the rules, we define the notation for the formula simplifications and modifications involved. For a literal ℓ, we use $\varphi[\ell = 1]$ to denote the formula obtained from φ by assigning $\ell = 1$. This simplifies φ by removing all satisfied clauses (those containing ℓ) and deleting all falsified literals (occurrences of $\overline{\ell}$) from the remaining clauses. Accordingly, we define $\varphi[\ell = 0] = \varphi[\overline{\ell} = 1]$. Extending this to a clause $C = (\ell_1 \vee \cdots \vee \ell_k)$, we define $\varphi[C = 0] = \varphi[\ell_1 = 0, \ldots, \ell_k = 0]$ (falsifying C by assigning all its literals to 0) and $\varphi[C = 1] = \varphi \wedge C$ (requiring C to be satisfied by adding it to the formula). We use a combined notation to represent a sequence of operations in order (e.g., $\varphi[C_1 = 1, C_2 = 0, \ell = 1]$ denotes the formula obtained by adding C_1, falsifying C_2, and assigning $\ell = 1$).

3.1 Reduction Rules

A reduction rule (R-Rule) is correct if it takes a formula φ as input and outputs a formula φ' with $\text{Parity}(\varphi) = \text{Parity}(\varphi')$, or correctly reports the value of $\text{Parity}(\varphi)$. The correctness of most rules is easily seen, and we will briefly justify some of them. A formal statement of correctness is given in Lemma 3 below, and the proof is deferred to the full version. The first five reduction rules are standard, which can also be applied to exact counting.

▶ **R-Rule 1** (Base case). *If formula φ contains an empty clause, report* $\text{Parity}(\varphi) = 0$.

▶ **R-Rule 2** (Elimination of duplicated literals). *If a clause C contains duplicated literals ℓ, remove all but one ℓ in C.*

▶ **R-Rule 3** (Elimination of tautology). *If a clause C contains two complementary literals ℓ and $\overline{\ell}$, remove clause C.*

▶ **R-Rule 4** (Elimination of subsumptions). *If there are two clauses C and D such that $C \subseteq D$, remove clause D.*

▶ **R-Rule 5** (Elimination of 1-clauses). *If there is a 1-clause (ℓ), assign $\ell = 1$.*

Next, we introduce four rules (i.e., R-Rules 6, 7, 8, 9) that are specific for Parity-SAT.

▶ **R-Rule 6** (Elimination of 0-variables). *If there is an unassigned variable x that does not appear in any clause, report that* $\text{Parity}(\varphi) = 0$.

The correctness of R-Rule 6 follows from a simple parity-based observation: if there is a 0-variable, then the parity of the number of satisfying assignments is even. This is because such a free variable doubles the number of solutions of the formula without the variable.

We say that a literal x *dominates* a variable y (where $y \neq \text{var}(x)$) if every clause containing variable y also contains literal x. Observe that assigning $x = 1$ makes y a 0-variable in the resulting formula $\varphi[x = 1]$. This implies that $\text{Parity}(\varphi[x = 1]) = 0$ and so $\text{Parity}(\varphi) = \text{Parity}(\varphi[x = 0])$. Thus, we have the following two reduction rules:

▶ **R-Rule 7** (Elimination of 1-variables). *If there is a 1-variable x and let $(x \vee C)$ be the clause containing x, then assign $x = 1$ and assign all literals in C to 0.*

▶ **R-Rule 8** (Domination). *If there is a literal x that dominates some variable y, assign $x = 0$.*

Technically, R-Rule 7 is a special case of R-Rule 8. If variable x solely appears in $(x \vee C)$, R-Rule 8 assigns all literals in C to 0 (as they dominate x), leaving (x) and forcing $x = 1$ via R-Rule 5. We state R-Rule 7 explicitly for conceptual clarity and frequent subsequent use.

Two literals ℓ_a and ℓ_b (where $\mathrm{var}(\ell_a) \neq \mathrm{var}(\ell_b)$) are *twins* if every clause in φ contains ℓ_a (resp., $\overline{\ell_a}$) if and only if it also contains ℓ_b (resp., $\overline{\ell_b}$).

▶ **R-Rule 9.** *If two literals ℓ_a and ℓ_b are twins in φ, remove the variable $\mathrm{var}(\ell_b)$ from the variable set (i.e., remove ℓ_b and $\overline{\ell_b}$ from the clauses, and $\mathrm{var}(\varphi) \leftarrow \mathrm{var}(\varphi) \setminus \{\mathrm{var}(\ell_b)\}$).*

The following two rules, R-Rule 10 and R-Rule 11, are also applicable for exact counting.

▶ **R-Rule 10.** *If there is a clause $(\ell \vee C)$ and another clause $(\overline{\ell} \vee C \vee D)$, remove the literal $\overline{\ell}$ from $(\overline{\ell} \vee C \vee D)$.*

For two literals ℓ_a and ℓ_b, *setting* $\ell_a = \ell_b$ means replacing all occurrences of ℓ_a with ℓ_b (and $\overline{\ell_a}$ with $\overline{\ell_b}$) in the formula, and then removing the variable $\mathrm{var}(\ell_a)$ from the variable set.

▶ **R-Rule 11.** *If there are two 2-clauses $(x \vee y)$ and $(\overline{x} \vee \overline{y})$ in φ, set $x = \overline{y}$ and apply R-Rule 3 exhaustively.*

We use $\mathsf{brute\text{-}force}(\varphi)$ to denote the brute-force algorithm that returns the parity of the number of satisfying assignments of formula φ by enumerating all possible assignments. Note that $\mathsf{brute\text{-}force}(\varphi)$ runs in $O(1)$ time if φ has only a constant number of variables.

▶ **R-Rule 12** (isolate). *If formula φ can be partitioned into two non-empty subformulas φ_1 and φ_2, i.e., $\varphi = \varphi_1 \wedge \varphi_2$, with $\mathrm{var}(\varphi_1) \cap \mathrm{var}(\varphi_2) = \emptyset$ and $|\mathrm{var}(\varphi_1)| \leq 10$, do the following: (1) $p \leftarrow \mathsf{brute\text{-}force}(\varphi_1)$; (2) If $p = 0$, report $\mathrm{Parity}(\varphi) = 0$; (3) Remove φ_1 from φ.*

▶ **R-Rule 13** (semi-isolate). *If formula φ can be partitioned into two non-empty subformulas φ_1 and φ_2, i.e., $\varphi = \varphi_1 \wedge \varphi_2$, with $\mathrm{var}(\varphi_1) \cap \mathrm{var}(\varphi_2) = \{x\}$ and $|\mathrm{var}(\varphi_1)| \leq 10$, do the following: (1) $p_1 \leftarrow \mathsf{brute\text{-}force}(\varphi_1[x = 1])$ and $p_0 \leftarrow \mathsf{brute\text{-}force}(\varphi_1[x = 0])$; (2) If $p_1 = p_0 = 0$, report $\mathrm{Parity}(\varphi) = 0$. (3) Remove φ_1 from φ; (4) If $p_0 = 1$ and $p_1 = 0$, assign $x = 0$. If $p_0 = 0$ and $p_1 = 1$, assign $x = 1$. If $p_0 = p_1 = 1$, leave x as an unassigned variable.*

Since φ_1 and φ_2 share only the variable x, they become disjoint once x is assigned. Thus, the parity calculation splits over the two possible assignments to x: $\mathrm{Parity}(\varphi) \equiv \mathrm{Parity}(\varphi_1[x = 0]) \cdot \mathrm{Parity}(\varphi_2[x = 0]) + \mathrm{Parity}(\varphi_1[x = 1]) \cdot \mathrm{Parity}(\varphi_2[x = 1]) \pmod 2$. Substituting p_0 and p_1, we obtain $\mathrm{Parity}(\varphi) \equiv p_0 \cdot \mathrm{Parity}(\varphi_2[x = 0]) + p_1 \cdot \mathrm{Parity}(\varphi_2[x = 1]) \pmod 2$. The four cases directly evaluate this equation for all possible combinations of (p_0, p_1).

We remark that the threshold number 10 in the above two rules is chosen arbitrarily to exclude small isolated or semi-isolated subformulas in our subsequent analysis. It only needs to be a sufficiently large constant; using a larger or smaller safe value will not affect the current analysis.

▶ **Definition 2** (reduced formulas). *We use $R(\varphi)$ to denote the resulting formula after exhaustively applying the above reduction rules on φ. A formula φ is* reduced *if $\varphi = R(\varphi)$.*

▶ **Lemma 3 (♣).** *All the reduction rules are correct, i.e., $\mathrm{Parity}(\varphi) = \mathrm{Parity}(R(\varphi))$.*

▶ **Lemma 4 (♣).** *It takes polynomial time and space to transform any formula φ to $R(\varphi)$.*

▶ **Lemma 5 (♣ properties of reduced formula).** *For a reduced formula φ, it holds that*

1. *All variables in φ are 2^+-variables;*
2. *All clauses in φ are 2^+-clauses;*
3. *Any two clauses in φ have at most one common 2-variable;*
4. *Any two 2-clauses can have at most one common variable;*
5. *For any subformula φ' with $|\mathrm{var}(\varphi')| \leq 10$, it holds that $|\mathrm{var}(\varphi') \cap \mathrm{var}(\varphi \setminus \varphi')| \geq 2$.*

3.2 Branching Rules

Let x be a variable in formula φ. The simplest branching rule is to branch on $x = 0$ and $x = 1$, following from $\text{Parity}(\varphi) = \text{Parity}(\varphi[x = 0]) \oplus \text{Parity}(\varphi[x = 1])$; we refer to this as *simple branching*. We also utilize the following branching rule based on a variable:

▶ **Lemma 6** (♣ variable branching). *Let x be a d-variable in formula φ that is contained in clauses $(\ell_{x,1} \vee C_1), (\ell_{x,2} \vee C_2), ..., (\ell_{x,d} \vee C_d)$, where $\ell_{x,i} \in \{x, \overline{x}\}$ for each $i \in [d]$. Then*

$$\text{Parity}(\varphi) = \bigoplus_{i=1}^{d} \text{Parity}(\varphi[C_1 = \cdots = C_{i-1} = 1, C_i = 0, \ell_{x,i} = 1]).$$

Apart from branching on variables, we can also perform branching based on clauses.

▶ **Lemma 7** (clause branching). *Let C be a clause in formula φ. Let $\varphi \setminus \{C\}$ denote the formula obtained by removing C from φ. Then*

$$\text{Parity}(\varphi) = \text{Parity}(\varphi \setminus \{C\}) \oplus \text{Parity}((\varphi \setminus \{C\})[C = 0]).$$

Proof. Let S be the set of satisfying assignments for the relaxed formula $\varphi \setminus \{C\}$. We partition S into two disjoint sets: $S_{\text{sat}} = \{\sigma \in S \mid \sigma(C) = 1\}$ and $S_{\text{unsat}} = \{\sigma \in S \mid \sigma(C) = 0\}$. Observe that S_{sat} is exactly the set of satisfying assignments for φ, and S_{unsat} corresponds to the satisfying assignments for $(\varphi \setminus \{C\})[C = 0]$. Since $|S| = |S_{\text{sat}}| + |S_{\text{unsat}}|$, the lemma follows by taking modulo 2. ◀

In what follows, we assume any branched formula φ is reduced; a formula φ' created in a sub-branch is not necessarily reduced, so we use $R(\varphi')$ to denote its reduced one.

4 Faster-than-2^m Algorithms for Parity-d-occ-SAT

In this section, we present algorithms for Parity-d-occ-SAT that improve upon the 2^m-barrier, where m is the number of clauses. To this end, we first give a Turing reduction from the general Parity-SAT (which includes Parity-d-occ-SAT) to its restricted version on *positive formulas*, i.e., formulas containing only positive literals. We note that Parity-SAT on positive formulas cannot be solved in subexponential time under the randomized Exponential Time Hypothesis (see Appendix A for details).

▶ **Lemma 8.** *Suppose there is an algorithm $\mathcal{A}_{pos}$ that solves Parity-SAT on positive formulas in time $O^*(r^m)$. Then, Parity-SAT can be solved in time $O^*(\max(1.6181, r)^m)$.*

Proof. First, for any variable x that appears solely as negative literals $\overline{x}$ in φ, we flip the variable (i.e., replace every occurrence of $\overline{x}$ with x). If the formula φ (with variables flipped) is already a positive formula, we directly apply algorithm $\mathcal{A}_{\text{pos}}$ to solve it in $O^*(r^m)$ time.

Otherwise, there must exist a variable x that has both positive and negative literals, i.e., x is a (a, b)-variable with $a, b \geq 1$. Let C be a clause containing the literal x. There exists at least one other clause D containing the complementary literal $\overline{x}$. We apply the clause branching rule in Lemma 7 on C. In the first branch $(\varphi \setminus \{C\})$, the clause C is removed, so the number of clauses decreases by 1. In the second branch $(\varphi[C = 0])$, all literals in C are assigned to 0. In particular, x is assigned to 0 and clause D is satisfied. Thus, both C and D are removed. The number of clauses decreases by at least 2. This yields a branching factor of $\tau(1, 2) < 1.6181$. The overall running time is therefore bounded by $O^*(\max(1.6181, r)^m)$. ◀

The above lemma suggests that, to break the 2^m-barrier, it suffices to solve this restricted version efficiently. In our case, we will use the following corollary:

▶ **Corollary 9.** *Suppose there exists an algorithm $\mathcal{A}_{pos}$ that solves Parity-d-occ-SAT on positive formulas in time $O^*(r_d^m)$. Then, Parity-d-occ-SAT can be solved in time $O^*(\max(1.6181, r_d)^m)$.*

Now we are left with solving Parity-d-occ-SAT on positive formulas. We observe that applying the variable branching from Lemma 6 to positive formulas yields a running time of $O^*(2^{m(1-1/O(2^d))})$. Specifically, applying this rule on a variable of degree d results in a measure decrease of $d - i + 1$ in the i-th branch (where d clauses are satisfied and $i - 1$ clauses are added back). This generates a branching factor of $\phi_d = \tau(d, d-1, \ldots, 1)$. Here, ϕ_d is the unique real solution of $x^d(2-x) = 1$ in the interval $(1, 2)$, known as the d-th order Fibonacci constant, which is $2^{(1-1/O(2^d))}$. With this simple algorithm for positive formulas, we conclude that Parity-d-occ-SAT can be solved in time $O^*(\phi_d^m) \subseteq O^*(2^{m(1-1/O(2^d))})$. That is, for any fixed d, the running time has a base strictly smaller than 2.

Next, we demonstrate that by leveraging existing results, we can improve the asymptotic bound to $O^*(2^{m(1-1/O(d))})$, as desired.

▶ **Theorem 10.** *There is a randomized algorithm that solves Parity-d-occ-SAT with m clauses in $O^*(2^{m(1-1/O(d))})$ time and polynomial space.*

Proof. By Corollary 9, it suffices to give an $O^*(2^{m(1-1/O(d))})$-time algorithm for Parity-d-occ-SAT on positive formulas. To this end, we reduce this problem with m clauses to #d-SAT with $N = m$ variables via a chain of reductions, and then apply the randomized algorithm in [26] to solve the instance in $O^*(2^{N(1-1/O(d))})$ time and polynomial space.

We first formalize the intermediate problems that will be used in the chain of reduction. Given a universe U and a family $\mathcal{F}$ of subsets of U, a *hitting set* of $(U, \mathcal{F})$ is a subset $H \subseteq U$ such that for every $S \in \mathcal{F}$, $H \cap S \neq \emptyset$, while a *set cover* of $(U, \mathcal{F})$ is a subcollection $\mathcal{C} \subseteq \mathcal{F}$ such that $\bigcup_{S \in \mathcal{C}} S = U$. The Parity-HS (resp. Parity-SC) problem asks to compute the parity of the number of hitting sets (resp. set covers) of $(U, \mathcal{F})$. We denote by Parity-d-occ-HS the restriction of Parity-HS where each element $u \in U$ occurs in at most d sets of $\mathcal{F}$. Furthermore, we define Parity-d-HS and Parity-d-SC as the restrictions of Parity-HS and Parity-SC, respectively, where each set $S \in \mathcal{F}$ has size at most d.

An overview of the chain of reductions is illustrated in Figure 1.

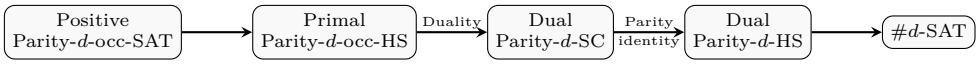


Figure 1 The chain of reductions from Parity-d-occ-SAT on positive formulas to #d-SAT.

Consider a positive formula φ with variable set V and clause set $\mathcal{C}$ (where $|\mathcal{C}| = m$). We can view φ as a set system $(V, \mathcal{C})$. Since each variable appears at most d times, computing Parity(φ) is equivalent to solving Parity-d-occ-HS on $(V, \mathcal{C})$. To solve this, we consider the *dual* set system $(\mathcal{C}, V^*)$, where the universe is the set of clauses $\mathcal{C}$, and V^* contains, for each variable $x \in V$, the set of clauses containing x. Crucially, hitting sets in the primal system $(V, \mathcal{C})$ correspond bijectively to set covers in the dual system $(\mathcal{C}, V^*)$. Moreover, the frequency constraint in the primal system implies that every set in the dual family V^* has size at most d. Thus, the problem is equivalent to Parity-d-SC on a universe of size m.

It was shown by Cygan et al. [15, Theorem 4.3] that for any set system, the parity of the number of set covers equals the parity of the number of hitting sets. Applying this identity to our dual system, we are left with solving Parity-d-HS on the universe $\mathcal{C}$ of size m. Finally, this naturally reduces to solving #d-SAT with m variables, which completes the proof. ◀

5 An Algorithm for Parity-2-occ-SAT

In this section, we present a fast algorithm for Parity-2-occ-SAT. We analyze the time complexity of this algorithm with respect to two parameters: the number of variables n and the number of clauses m, showing that Parity-2-occ-SAT can be solved in $O(1.1193^n)$ time or $O^*(1.3248^m)$ time. We remark that Parity-2-occ-SAT typically lies in the domain where $m \leq n$. Let $\bar{k}$ be the average clause length; then the formula length is $\bar{k}m \leq 2n$. Since unit clauses can be eliminated (e.g., by our R-Rule 5), we have $\bar{k} \geq 2$, which implies $m \leq n$. Thus, a runtime bound of $O^*(2^{cn})$ for some constant c is not as good as a bound of $O^*(2^{cm})$ for Parity-2-occ-SAT. Besides, we note that this problem cannot be solved in subexponential time under the randomized Exponential Time Hypothesis (see Appendix A for details).

Our algorithm consists of two steps that utilize the clause branching in Lemma 7. In the first step, we deal with 4^+-clauses, and directly branch on a clause with the largest size. In the second step, all clauses have a size of at most 3, and we employ a branching strategy that aims to decompose the formula into disjoint subformulas so that we can efficiently solve it in a divide-and-conquer manner. Specifically, the selection of the clause we branch on is guided by a bisection (i.e., a balanced separator) of a specific graph representation of the formula (the definitions will be given later). We note that this paradigm of utilizing graph bisections to guide branching decisions has been successfully applied to design fast polynomial-space algorithms for various problems (e.g., [22, 25, 24]).

5.1 Dealing with 4^+-clauses

In this step, we pick up a 4^+-clause C and branch on: (B1) remove clause C; (B2) remove clause C and set all literals in C to 0.

Two clauses $C, D \in \varphi$ are called *neighbors* if they share at least one common variable, i.e., $\mathrm{var}(C) \cap \mathrm{var}(D) \neq \emptyset$. For a clause $C \in \varphi$, we let $N_\varphi(C)$ denote the set of its neighbors, and define $N_\varphi(C, 2) := \{D \mid D \in N_\varphi(C), |D| = 2\}$ to be the set of 2-clauses adjacent to C.

▶ **Lemma 11.** *Let C be a clause in formula φ. Let $\varphi_1 = \varphi \setminus \{C\}$ and $\varphi_2 = \varphi[C = 0] \setminus \{C\}$. It holds that*

- $m(\varphi) - m(R(\varphi_1)) \geq |C| + 1$ *and* $m(\varphi) - m(R(\varphi_2)) \geq 1$.
- $n(\varphi) - n(R(\varphi_1)) \geq |\mathrm{var}(C \cup N_\varphi(C))|$ *and* $n(\varphi) - n(R(\varphi_2)) \geq |C| + |N_\varphi(C, 2)|$.

Proof. We first note that the exhaustive application of the reduction rules introduces neither new clauses nor new variables. Thus, for any intermediate formula φ', it holds that $m(R(\varphi')) \leq m(\varphi')$ and $n(R(\varphi')) \leq n(\varphi')$.

In the first branch, let $\varphi_1 = \varphi \setminus \{C\}$. The explicit removal of C decreases the number of clauses by 1. Because φ is a reduced formula, every variable in $\mathrm{var}(C)$ is a 2-variable and has exactly one other occurrence outside of C. By Property 3 of Lemma 5, any two clauses share at most one 2-variable. Thus, the remaining $|C|$ occurrences of the variables in $\mathrm{var}(C)$ belong to exactly $|C|$ distinct clauses in $N_\varphi(C)$. Upon the removal of C, all variables in $\mathrm{var}(C)$ become 1-variables. According to R-Rule 7, each 1-variable x is assigned a value of 1, which satisfies and removes its unique remaining clause in $N_\varphi(C)$. Since these $|C|$ clauses are mutually distinct, their removal yields $m(\varphi) - m(R(\varphi_1)) \geq |C| + 1$. Furthermore, by R-Rule 7, all other literals in the clause containing x are set to 0. Since this process iterates over all $x \in \mathrm{var}(C)$, all variables in $\mathrm{var}(N_\varphi(C))$ are assigned and eliminated. Therefore, $n(\varphi) - n(R(\varphi_1)) \geq |\mathrm{var}(C \cup N_\varphi(C))|$.

In the second branch, let $\varphi_2 = \varphi[C = 0] \setminus \{C\}$. The explicit removal of C immediately yields $m(\varphi) - m(R(\varphi_2)) \geq 1$. For the number of variables, all literals in C are assigned to 0; thus, the $|C|$ variables in $\mathrm{var}(C)$ are eliminated. We then consider the clauses in $N_\varphi(C, 2)$. For each 2-clause $D \in N_\varphi(C, 2)$, let $\mathrm{var}(D) = \{x, y\}$ where $x \in \mathrm{var}(C)$ and $y \notin \mathrm{var}(C)$. The assignment to x either satisfies D or falsifies the literal of x in D. If D is satisfied, it is removed; the remaining 2-variable y loses an occurrence, becomes a 1-variable, and is subsequently assigned and eliminated via R-Rule 7. If the literal of x is falsified, D shrinks to a 1-clause, forcing y to be assigned and eliminated via R-Rule 5. In either case, the other variable y in D is eliminated. This guarantees an additional decrease of at least $|N_\varphi(C, 2)|$ variables. Consequently, $n(\varphi) - n(R(\varphi_2)) \geq |C| + |N_\varphi(C, 2)|$. ◀

▶ **Lemma 12 (♣).** *Let C be a clause with $|C| \geq 4$ in a reduced formula φ. Branching on C yields a branching factor of at most $\tau(5, 1) < 1.3248$ with respect to the number of clauses.*

▶ **Lemma 13.** *Let C be a clause with $|C| \geq 4$ in a reduced formula φ. Branching on C yields a branching factor of at most $\tau(9, 4) < 1.1193$ with respect to the number of variables.*

Proof. Let $\varphi_1 = \varphi \setminus \{C\}$ and $\varphi_2 = \varphi[C = 0] \setminus \{C\}$. Let $\Delta_{n,1}$ and $\Delta_{n,2}$ be the decreases in the number of variables in the respective branches. By Lemma 11, we have $\Delta_{n,1} \geq |\mathrm{var}(C \cup N_\varphi(C))|$ and $\Delta_{n,2} \geq |C| + |N_\varphi(C, 2)|$.

Let $E = \mathrm{var}(N_\varphi(C)) \setminus \mathrm{var}(C)$ be the set of external variables, and $c_2 = |N_\varphi(C, 2)|$. Since $|C| \geq 4$, we have $\Delta_{n,2} \geq |C| + c_2 \geq 4$ and $\Delta_{n,1} \geq |C| + |E| \geq 4$. To get the desired branching factor $\tau(9, 4)$ via Lemma 1, it suffices to show that their sum $\Delta_{n,1} + \Delta_{n,2} \geq 13$. Since $\Delta_{n,1} + \Delta_{n,2} \geq 2|C| + c_2 + |E| \geq 8 + c_2 + |E|$, it remains to prove $c_2 + |E| \geq 5$.

Assume for contradiction that $c_2 + |E| \leq 4$. We partition E into two sets based on their occurrences in $N_\varphi(C)$: k_1 variables appearing exactly once, and k_2 variables appearing exactly twice. Thus, $|E| = k_1 + k_2 \leq 4$. Counting the literal occurrences in $N_\varphi(C)$, the $|C|$ clauses require at least $2c_2 + 3(|C| - c_2) = 3|C| - c_2$ occurrences. These are supplied by $\mathrm{var}(C)$ (exactly $|C|$ occurrences) and E (exactly $k_1 + 2k_2$ occurrences). This yields $3|C| - c_2 \leq |C| + k_1 + 2k_2$, which simplifies to $2|C| \leq c_2 + k_1 + 2k_2$.

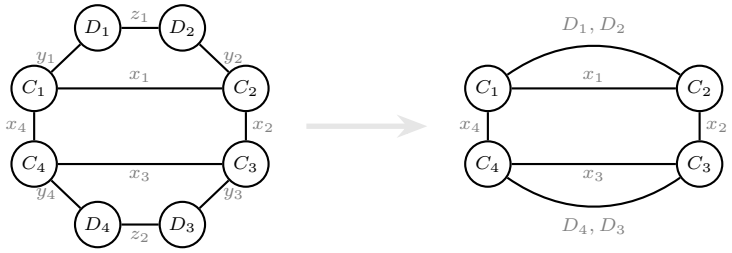
Recall that we assume for contraidction that $c_2 + |E| = c_2 + k_1 + k_2 \leq 4$. Because $|C| \geq 4$, we have $8 \leq 2|C| \leq (c_2 + k_1 + k_2) + k_2 \leq 4 + k_2$, which implies $k_2 \geq 4$. With $k_1 + k_2 = |E| \leq 4$, we have $k_2 = 4$, $k_1 = 0$, and $c_2 = 0$ (along with $|C| = 4$). However, k_1 represents the number of variables shared between the induced subformula $C \cup N_\varphi(C)$ and the rest of φ. If $k_1 = 0$, this subformula, which contains exactly $|C| + |E| = 8 \leq 10$ variables, shares 0 variables with the rest of the formula. This contradicts Property 5 of Lemma 5, which states that any small subformula must share at least 2 variables with the rest.

Therefore, $c_2 + |E| \geq 5$. Consequently, $\Delta_{n,1} + \Delta_{n,2} \geq 13$. With $\min(\Delta_{n,1}, \Delta_{n,2}) \geq 4$, the branching factor is at most $\tau(13 - 4, 4) = \tau(9, 4) < 1.1193$. ◀

5.2 Solving 3-CNF instances via a Bisection Approach

Now there are only 3-clauses and 2-clauses. We adopt a graph-theoretic approach to design and analyze the algorithm. In what follows, we first introduce the specific graph representation used throughout the algorithm and then present the algorithm.

Graph Representation. The *dual graph* G_φ of a formula φ is the graph where each vertex corresponds to a clause in φ, and two distinct vertices C_i and C_j are connected by an edge if and only if $\mathrm{var}(C_i) \cap \mathrm{var}(C_j) \neq \emptyset$. Lemma 5 ensures that any pair of clauses shares at most one 2-variable. Since every variable appears exactly twice, each variable corresponds to a unique edge between two clauses.



Dual Graph G_φ **Multigraph G^*_φ**

Figure 2 An illustrative example of the dual graph G_φ (left) and the multigraph G^*_φ (right) for φ, which consists of four 3-clauses $C_1 = (x_1 \vee \overline{x_4} \vee y_1), C_2 = (\overline{x_1} \vee x_2 \vee y_2), C_3 = (x_2 \vee x_3 \vee y_3), C_4 = (x_3 \vee x_4 \vee y_4)$ and four 2-clauses $D_1 = (\overline{y_1} \vee z_1), D_2 = (z_1 \vee y_2), D_3 = (y_3 \vee z_2), D_4 = (\overline{z_2} \vee y_4)$. Each edge in G_φ corresponds to a single variable; in G^*_φ, chains of 2-clauses are replaced with (potentially parallel) edges by smoothing out all degree-2 vertices D_j.

For any clause C, consider the effects of either removing it from the formula or assigning values to its variables. Let D be a neighboring clause of C and x be the shared variable between C and D. If C is removed, the shared variable x becomes a 1-variable, triggering R-Rule 7 to assign values to other variables in D to satisfy and remove D. If the shared variable x is assigned a value, D is either satisfied or reduced in length by at least one; in particular, if D is a 2-clause, shrinking to a 1-clause triggers R-Rule 5 to assign a value to its remaining variable. Thus, if D is a 2-clause, it is eliminated in both scenarios and affects its other neighbor in the same manner. Ultimately, this results in the removal of a chain of 2-clauses (i.e., a path of degree-2 vertices in G_φ).

Given the above chain reaction, we construct a *multigraph* $G^*_\varphi = (V^*, E^*)$ by smoothing out all degree-2 vertices in G_φ, where

- the vertex set V^* consists of the $m_3(\varphi)$ degree-3 vertices in G_φ (i.e., all 3-clauses in φ);
- for two vertices $C_i, C_j \in V^*$, each edge between them in E^* corresponds to a distinct connection in G_φ, which is either a direct edge (i.e., sharing a variable) or a path of degree-2 vertices (i.e., a chain of 2-clauses).

See Figure 2 for an illustrative example.

Note that any connected component in G_φ consisting entirely of degree-2 vertices is absent from G^*_φ. This is harmless, as it induces an isolated 2-CNF formula, which can be solved in polynomial time by the following lemma.

▶ **Lemma 14 (♣).** *Parity-2-occ-SAT on 2-CNF formulas can be solved in polynomial time.*

Furthermore, G^*_φ might contain self-loops if a 3-clause $C = (x \vee y \vee z)$ connects to itself via a chain of 2-clauses with endpoints y and z. To eliminate such loops, let φ_1 be the subformula formed by C and this chain, and $\varphi_2 = \varphi \setminus \varphi_1$. Note that $\mathrm{var}(\varphi_1) \cap \mathrm{var}(\varphi_2) = \{x\}$, and both $\varphi_1[x = 0]$ and $\varphi_1[x = 1]$ are 2-CNF formulas. This allows us to apply R-Rule 13, with the only modification that $\mathrm{Parity}(\varphi_1[x = 0])$ and $\mathrm{Parity}(\varphi_1[x = 1])$ are computed in polynomial time via Lemma 14. Thus, there is no self-loop in G^*_φ for a reduced formula φ.

Bisection. A *bisection* of a graph $G = (V, E)$ is a partition (V_1, V_2) of the vertex set V such that their sizes are as equal as possible, i.e., $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ and $||V_1| - |V_2|| \leq 1$. The *size* of a bisection is the number of edges with one endpoint in V_1 and the other in V_2. The minimum size over all possible bisections of G is called the *bisection width* of the graph. We rely on the following upper bound on the bisection width of cubic graphs.

Algorithm 1 `Bisection-Solve`(φ, A, B).

Input: A reduced formula φ, and a disjoint partition (A, B) of the 3-clauses in φ.
Output: The parity of the number of satisfying assignments of φ (1 for odd, 0 for even).

▷ Let $\varepsilon > 0$ be a fixed small constant and n_ε be the integer defined in Theorem 15.

if $|V(G^*_\varphi)| \leq n_\varepsilon$ **then**
    Solve the instance by iteratively branching on 3-clauses and solving the resulting 2-CNF instances using Lemma 14, and **return** the result.
if $B = \emptyset$ **then** ▷ bisection step
    $(A', B') \leftarrow$ `Monien-Preis`(G^*_φ);
    return `Bisection-Solve`(φ, A', B');
if there are no edges between A and B **then** ▷ divide-and-conquer step
    Let φ_A and φ_B be the disjoint subformulas induced by A and B, respectively.
    return `Bisection-Solve`$(\varphi_A, A, \emptyset) \wedge$ `Bisection-Solve`$(\varphi_B, B, \emptyset)$.

▷ branching step

Select an edge between A and B, and pick its endpoint 3-clause $C \in A \cup B$, alternating the selection of C between A and B at each successive level of the recursion.
Let $\varphi_1 = \varphi \setminus \{C\}$ and $\varphi_2 = \varphi[C = 0] \setminus \{C\}$.
for $i \in \{1, 2\}$ **do**
    Exhaustively apply reduction rules on φ_i.
    Let $A_i \subseteq A$ and $B_i \subseteq B$ be the subsets of clauses that remain as 3-clauses in φ_i.
return `Bisection-Solve`$(\varphi_1, A_1, B_1) \oplus$ `Bisection-Solve`(φ_2, A_2, B_2).

▶ **Theorem 15** (Monien and Preis [31]). *For any $\varepsilon > 0$, there exists an integer n_ε such that for any cubic graph G with $|V(G)| \geq n_\varepsilon$, the bisection width of G is at most $(\frac{1}{6} + \varepsilon)|V(G)|$.*

As noted in [20, 22], there is a polynomial-time algorithm (denoted by `Monien-Preis`(G)) that computes the corresponding bisection. We note that this result also applies to multigraphs (a proof of this claim is deferred to the full version).

The Algorithm. Now we are ready to describe our algorithm, presented in Algorithm 1. The algorithm follows a divide-and-conquer strategy where the branching process is guided by a partition initialized as a bisection of the multigraph G^*_φ. Specifically, it maintains this disjoint partition (A, B) of the 3-clauses, aiming to eliminate variables shared between these sets (i.e., edges between them) through branching. The algorithm is initially called with A being the set of all 3-clauses and $B = \emptyset$. Thus, initially or whenever the partition becomes trivial (i.e., $B = \emptyset$), the `Monien-Preis` procedure is invoked on G^*_φ to compute a bisection of guaranteed size. If no edges exist between A and B, the formula decomposes into two disjoint subformulas φ_A and φ_B, and the algorithm recursively solves them as independent instances. Otherwise, the algorithm selects an edge connecting A and B, and applies clause branching on an endpoint 3-clause C of this edge. Here, we alternate the selection of C between A and B across recursion levels to better balance the number of 3-clauses eliminated in both sides, which is crucial for bounding the overall worst-case running time.

The Analysis. Let $m_3(\varphi)$ be the number of 3-clauses in φ. Then we have $|V(G^*_\varphi)| = m_3(\varphi)$ and $L(\varphi) \geq 3m_3(\varphi)$. Since we are considering Parity-2-occ-SAT, $L(\varphi) \leq 2n(\varphi)$. Thus, $m_3(\varphi) \leq \frac{2}{3}n(\varphi)$. We will first obtain a running time bound in terms of $m_3(\varphi)$ and then transform it to a running time bound in terms of $m(\varphi)$ or $n(\varphi)$.

▶ **Lemma 16.** *Parity-2-occ-SAT on 3-CNF formulas can be solved in $O^*(1.1487^{m_3})$ time, where m_3 is the number of 3-clauses.*

Proof. Let us fix a sufficiently small constant $\varepsilon > 0$ (e.g., $\varepsilon = 10^{-9}$) and let n_ε be as defined in Theorem 15. If the formula φ has $m_3(\varphi) \leq n_\varepsilon$, we can solve it in polynomial time by exhaustively applying clause branching on this constant-sized set of 3-clauses (in each branch, $m_3(\varphi)$ decreases by at least one) and solving the resulting 2-CNF instances by Lemma 14.

When $m_3(\varphi) > n_\epsilon$, the algorithm executes Line 3 onwards. Before delving into the detailed proof, we provide a high-level outline. We will first define a new measure ρ that is initially bounded by $m_3(\varphi)$. Thus, establishing a runtime bound of $O^*(\gamma^\rho)$ for some $\gamma > 1$ yields a runtime bound of $O^*(\gamma^{m_3(\varphi)})$. We then demonstrate that (i) the bisection step (Line 3) does not increase the measure; (ii) the divide-and-conquer step (Line 6) does not affect the exponential part of the time complexity; and (iii) the branching step yields a branching factor of at most 1.1487.

We adopt the following measure for any valid input (φ, A, B) of the algorithm:

$$\rho(\varphi, A, B) = \max(|A|, |B|) + (3 - \varepsilon')|S|,$$

where $|A|$ and $|B|$ are the number of 3-clauses in the respective partitions, and $|S|$ is the number of edges between A and B. We choose ε' such that $(3-\varepsilon')(\frac{1}{6}+\varepsilon) \leq \frac{1}{2} - \frac{1}{n_\epsilon} < \frac{1}{2} - \frac{1}{m_3(\varphi)}$. Crucially, since the algorithm is initially invoked with $B = \emptyset$ and $A = V(G^*_\varphi)$, the initial measure is exactly $\rho(\varphi, A, \emptyset) = |A| = m_3(\varphi)$. Furthermore, all reduction rules do not introduce new variables or clauses; thus, applying them does not increase the measure ρ.

We first show that the bisection step does not increase the measure. When $B = \emptyset$, the algorithm applies the `Monien-Preis` procedure on G^*_φ, generating a new partition (A', B') and the corresponding S' with $|A'|, |B'| \leq \frac{|A|}{2} + 1$ and $|S'| \leq (\frac{1}{6} + \varepsilon)|A|$. Since $(3 - \varepsilon')(\frac{1}{6} + \varepsilon) \leq \frac{1}{2} - \frac{1}{m_3(\varphi)} = \frac{1}{2} - \frac{1}{|A|}$, the new measure $\rho(\varphi, A', B')$ is bounded by:

$$\rho(\varphi, A', B') \leq \left(\frac{|A|}{2} + 1\right) + (3 - \varepsilon')\left(\frac{1}{6} + \varepsilon\right)|A| \leq \frac{|A|}{2} + 1 + \frac{|A|}{2} - 1 = |A| = \rho(\varphi, A, B).$$

Let $T(\rho)$ be the maximum number of leaves in the search tree generated by the algorithm on a state with measure ρ, and let $\gamma > 1$ be the largest branching factor generated by the subsequent branching step. We will prove that $T(\rho) \in O^*(\gamma^\rho)$.

We show that the divide-and-conquer step does not affect the exponential part of the time complexity in terms of our measure ρ. To this end, we prove by induction on the measure ρ that $T(\rho) \leq m_3(\varphi)\gamma^\rho \in O^*(\gamma^\rho)$. In this step, $S = \emptyset$, and the formula decomposes into disjoint subformulas φ_A and φ_B. Let $\rho_A := \rho(\varphi_A, A, \emptyset)$ and $\rho_B := \rho(\varphi_B, B, \emptyset)$ be the measures of the two subproblems, respectively. Since $\rho = \max(|A|, |B|)$, we have $\rho_A = |A| \leq \rho$ and $\rho_B = |B| \leq \rho$. By the inductive hypothesis, the total cost of solving these two subformulas independently is:

$$T(\rho) \leq T(\rho_A) + T(\rho_B) \leq |A|\gamma^{\rho_A} + |B|\gamma^{\rho_B} \leq (|A| + |B|)\gamma^\rho = m_3(\varphi)\gamma^\rho.$$

It remains to show that the branching step generates a branching factor of at most $\gamma < 1.1487$. As established in the paragraph "Graph Representation", when a clause is removed from the formula or its variables are assigned, the operation triggers a chain reaction that propagates through its shared variables. In the multigraph G^*_φ, this corresponds to the deletion of the corresponding vertex and the removal of all its incident edges.

When $S \neq \emptyset$, the algorithm selects an edge in S connecting $C_A \in A$ and $C_B \in B$. Assume that we branch on C_A. Let $\varphi_1 = \varphi \setminus \{C_A\}$ and $\varphi_2 = \varphi[C_A = 0] \setminus \{C_A\}$. For each branch $i \in \{1, 2\}$, let A_i and B_i be the subsets of clauses that remain as 3-clauses in $R(\varphi_i)$. To formalize the measure drop, let $\Delta_{i,A} = |A| - |A_i|$, $\Delta_{i,B} = |B| - |B_i|$, and $\Delta_{i,S} = |S| - |S_i|$ denote the reductions in the respective parts. We show that for each branch $i \in \{1, 2\}$, either $\Delta_{i,S} \geq 2$, or $\Delta_{i,S} = 1$, $\Delta_{i,A} \geq 3$ and $\Delta_{i,B} \geq 1$.

If C_A is incident to two or more edges in S, the chain reaction in both φ_1 and φ_2 removes at least two edges from S, guaranteeing $\Delta_{i,S} \geq 2$ for $i \in \{1, 2\}$.

If C_A is incident to exactly one edge in S, since C_A is a 3-clause, its two remaining incident edges must connect to other clauses within A. If they connect to distinct clauses, the chain reaction in either φ_i propagates along both edges, satisfying or shrinking at least two additional clauses in A, yielding $\Delta_{i,A} \geq 3$. Alternatively, if the two internal edges connect to the same clause C'_A, the chain reaction either removes C'_A or shrinks C'_A to a 1-clause. In the latter case, the 1-clause will also be removed. Because C'_A is a 3-clause, its removal triggers a further chain reaction along its third incident edge and removes or shrinks a third 3-clause. If this third 3-clause belongs to A, we have $\Delta_{i,A} \geq 3$; otherwise, we remove one more edge in S and have $\Delta_{i,S} \geq 2$. Furthermore, the chain reaction propagates along the single edge in S connecting C_A to C_B. This eliminates the edge in S, yielding $\Delta_{i,S} \geq 1$, and eventually satisfies or shrinks C_B, ensuring $\Delta_{i,B} \geq 1$. Consequently, for both $i \in \{1, 2\}$, we guarantee that either $\Delta_{i,S} \geq 2$, or $\Delta_{i,S} = 1$, $\Delta_{i,A} \geq 3$ and $\Delta_{i,B} \geq 1$.

Recall that the algorithm employs a two-level alternating branching strategy. It first branches on $C_A \in A$ to obtain φ_1 and φ_2. Then, in each resulting subformula φ_i, it selects a clause in B_i incident to S and branches on it to generate subformulas $\varphi_{i,1}$ and $\varphi_{i,2}$. Let $\Delta_{i,j,A}$, $\Delta_{i,j,B}$, and $\Delta_{i,j,S}$ denote the respective component reductions in this second step for $j \in \{1, 2\}$. By the same argument as above, the second level guarantees that either $\Delta_{i,j,S} \geq 2$, or $\Delta_{i,j,S} \geq 1$, $\Delta_{i,j,B} \geq 3$, and $\Delta_{i,j,A} \geq 1$. Combining the two levels across any of the four branches in the search tree, it holds for the cumulative reduction for A, B, S that either (a) $\Delta_{i,S} + \Delta_{i,j,S} \geq 4$; or (b) $\Delta_{i,A} + \Delta_{i,j,A} \geq 1$, $\Delta_{i,B} + \Delta_{i,j,B} \geq 1$, and $\Delta_{i,S} + \Delta_{i,j,S} \geq 3$; or (c) $\Delta_{i,A} + \Delta_{i,j,A} \geq 4$, $\Delta_{i,B} + \Delta_{i,j,B} \geq 4$, and $\Delta_{i,S} + \Delta_{i,j,S} \geq 2$. The total measure drop in terms of ρ is at least $\min\{4(3-\varepsilon'), 1+3(3-\varepsilon'), 4+2(3-\varepsilon')\} = 10-3\varepsilon'$. This uniform drop across all four branches yields a branching factor of at most $\tau(10-3\varepsilon', 10-3\varepsilon', 10-3\varepsilon', 10-3\varepsilon') < 1.1487$. ◀

Since $m_3(\varphi) \leq m(\varphi)$ and $m_3(\varphi) \leq \frac{2}{3}n(\varphi)$, by Lemma 16 we have

▶ **Lemma 17.** *Parity-2-occ-SAT on 3-CNFs with m clauses and n variables can be solved using polynomial space in $O^*(1.1487^m)$ time or $O^*(1.0969^n)$ time.*

5.3 The Final Results

By Lemma 12, branching on 4^+-clauses yields a branching factor of at most $\tau(5, 1) < 1.3248$ with respect to the number of clauses m, which dominates the $O(1.1487^m)$ time bound for solving the resulting 3-CNF instances given by Lemma 17. Thus, we have

▶ **Theorem 18.** *Parity-2-occ-SAT with m clauses can be solved in $O^*(1.3248^m)$ time and polynomial space.*

Similarly, the branching factor of $\tau(9, 4) < 1.1193$ in terms of the number of variables n from Lemma 13 dominates the $O(1.0969^n)$ bound for 3-CNF instances. Thus, we obtain

▶ **Theorem 19.** *Parity-2-occ-SAT with n variables can be solved in $O(1.1193^n)$ time and polynomial space.*

6 An Algorithm for Parity-SAT in Terms of L

In this section, we present an algorithm for the general Parity-SAT problem that runs in $O(1.1052^L)$ time and polynomial space. Our algorithm processes variables in decreasing order of their degrees. By iteratively branching on and eliminating variables of high degrees, the algorithm ultimately reduces the given instance to Parity-2-occ-SAT. At this stage, we invoke the algorithm developed in the Section 5 to solve the remaining part. In our analysis, we employ the measure-and-conquer method [19]. Due to space limitations, the detailed algorithm/branching steps and the analysis are deferred to the full version. In what follows, we first introduce the measure used in our analysis, and then present what each step of the algorithm handles and its complexity.

6.1 The Measure

For a formula φ, let $n_i(\varphi)$ be the number of i-variables. We adopt the following measure:

$$\mu(\varphi) := \sum_{i\geq 1} w_i \cdot n_i(\varphi), \text{where } w_1 = 0, w_2 = 1.5, \text{and } w_i = i \text{ for } i \geq 3.$$

Since $w_i \leq i$ for all $i \geq 1$, it holds that $\mu(\varphi) \leq \sum_{i\geq 1} i \cdot n_i(\varphi) \leq L(\varphi)$. Thus, a running-time bound of $O^*(c^{\mu(\varphi)})$ for some $c > 1$ implies a running-time bound of $O^*(c^{L(\varphi)})$.

Intuitively, since our algorithm processes variables in decreasing order of their degrees and relies on an efficient subroutine for Parity-2-occ-SAT, this measure allows us to smoothly propagate the efficiency from the low-degree base case up to higher degrees.

For every integer $i \geq 1$, let $\delta_i := w_i - w_{i-1}$ denote the marginal weight, i.e., the decrease in the measure when a variable's degree drops from i to $i-1$. By the definition of the weights, we have $\delta_2 = \delta_3 = 1.5$ and $\delta_i = 1$ for all $i \geq 4$. Furthermore, the following properties hold:

$$w_3 = 2w_2, \quad w_2 = \delta_3, \quad \text{and} \quad \delta_i \leq \delta_{i-1} \text{ for every } i \geq 3.$$

▶ **Lemma 20 (♣).** *For any formula φ, applying any reduction rule does not increase $\mu(\varphi)$.*

6.2 The Algorithm

Table 2 Overview of branching factors for each step in the algorithm.

Steps	Branching Objects	Vectors	Factors
Step 1	4^+-variables	$(12, 4)$	1.1003
Step 2	4^+-clauses containing 3-variables	$(7.5, 7.5)$	1.0969
Step 3	3-variables with both positive literals and negative literals	Step 3.1: $(6, 9)$ Step 3.2: $(10.5, 4.5)$	1.0983 1.1031
Step 4	3-variables contained in 2-clauses	$(10.5, 4.5)$	1.1031
Step 5	Remaining 3-variables	Step 5.1: $(13.5, 3)$ Step 5.2: $(15, 12, 9)$	1.1052* 1.0983
Step 6	2-variables (reduce to Parity-2-occ-SAT)	$1.1193^{\frac{1}{1.5}}$ (Theorem 19)	1.0781

Our algorithm consists of six steps. An overview of the branching objects, vectors, and the resulting factors for each step is summarized in Table 2.

In the algorithm, Step 1 first applies simple branching to eliminate all 4^+-variables. Next, Steps 2–5 deal with 3-variables, which is the core of the algorithm, by progressively purifying their local structures. Specifically, Step 2 applies clause branching (Lemma 7) to eliminate 4^+-clauses. Step 3 handles variables with both positive and negative occurrences, applying clause branching if the negative literal appears in a 3-clause (Step 3.1) and simple branching if it is in a 2-clause (Step 3.2). Step 4 applies simple branching to 3-variables contained in 2-clauses. Finally, Step 5 resolves the remaining well-structured cases, where all 3-variables are positive and appear only in 3-clauses. Step 5.1 applies simple branching to any 3-variable whose clauses either contain a 2-variable or share another 3-variable. For the remaining variables, whose three clauses contain only distinct 3-variables, Step 5.2 applies a tailored variable branching (Lemma 6). After eliminating all 3-variables, Step 6 reduces the instance to Parity-2-occ-SAT. At this point, the remaining formula consists entirely of 2-variables, meaning $\mu(\varphi) = w_2 n(\varphi) = 1.5n(\varphi)$. Consequently, it can be solved in $O^*(1.1193^{\frac{\mu(\varphi)}{1.5}}) \subseteq O^*(1.0781^{\mu(\varphi)})$ time.

The bottleneck of the algorithm arises in Step 5.1, which yields a worst-case branching factor of 1.1052.

▶ **Theorem 21 (♣).** *Parity-SAT can be solved in $O(1.1052^L)$ time and polynomial space, where L is the formula length.*

At a high level, our analysis quantifies the measure decrease in each branch. We trace which variables are assigned or experience a degree drop to compute the measure decrease. Recall that dealing with 3-variables constitutes the core part of our algorithm. Setting $\delta_3 = \delta_2 = w_2 = 1.5$ significantly simplifies the analysis in these steps: in many cases, we do not need to distinguish whether a variable's degree drops from 3 to 2, or from 2 to 1, because both cases yield the same measure decrease of $w_2 = 1.5$. We primarily exploit the cascading measure drops from two specific scenarios: (1) when a clause shrinks into a 1-clause, triggering R-Rule 5 to assign its variable; and (2) when a variable becomes a 1-variable (e.g., when a 2-variable loses one occurrence due to the removal of clause containing it, or a 3-variable loses two occurrences simultaneously), triggering R-Rule 7 to force assignments on variables in its remaining clause. Other reduction rules are mainly used to guarantee certain structural properties of the reduced formula, enabling us to deduce tighter bounds. The detailed branching steps and their analyses are deferred to the full version.

7 Conclusion and Discussion

In this paper, we investigated algorithms for Parity-SAT and its bounded-occurrence version, Parity-d-occ-SAT. By systematically exploiting the properties of parity, we developed polynomial-space algorithms that achieve better running time bounds than the best-known bounds for their exact counting counterparts. Specifically, we broke the 2^m-barrier for Parity-d-occ-SAT for any fixed d with an $O^*(2^{m(1-1/O(d))})$-time algorithm, achieved refined bounds of $O(1.1193^n)$ and $O^*(1.3248^m)$ time for Parity-2-occ-SAT, and established an $O(1.1052^L)$-time algorithm for general Parity-SAT.

A natural question is whether one can break the 2^m-barrier for #d-occ-SAT, even for the base case of $d = 2$. In the parity setting, Lemma 8 allows us to safely restrict our attention to positive formulas. It is worth noting that since the clause branching rule in Lemma 7 inherently preserves the exact number of satisfying assignments, a counting variant

of Lemma 8 indeed holds. However, our subsequent algorithms for handling these positive formulas rely fundamentally on parity: the simple $O^*(2^{m(1-1/O(2^d))})$-time algorithm applies the parity-specific variable branching in Lemma 6, while the improved $O^*(2^{m(1-1/O(d))})$-time algorithm exploits the parity equivalence between hitting sets and set covers. Neither of these techniques can be directly lifted to the exact counting counterpart.

References

1 Amir Abboud, Shon Feller, and Oren Weimann. On the fine-grained complexity of parity problems. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 5:1–5:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.5.

2 Vikraman Arvind and Piyush P. Kurur. Graph isomorphism is in SPP. *Inf. Comput.*, 204(5):835–852, 2006. doi:10.1016/J.IC.2006.02.002.

3 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS 2003)*, pages 340–351, 2003. doi:10.1109/SFCS.2003.1238208.

4 Max Bannach, Erik D. Demaine, Timothy Gomez, and Markus Hecher. #p is sandwiched by one and two #2dnf calls: Is subtraction stronger than we thought? In *Proceedings of the 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2025)*, pages 31–43, 2025. doi:10.1109/LICS65433.2025.00010.

5 Max Bannach, Erik D. Demaine, Timothy Gomez, and Markus Hecher. A novel reduction from #sat to #2sat based on symmetry: *Simply Drop the Large Clauses*. In *Proceedings of the 2026 Symposium on Simplicity in Algorithms (SOSA 2026)*, pages 254–265, 2026. doi:10.1137/1.9781611978964.19.

6 Andreas Björklund, Holger Dell, and Thore Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 231–242, 2015. doi:10.1007/978-3-662-47672-7_19.

7 Andreas Björklund and Thore Husfeldt. The parity of directed hamiltonian cycles. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 727–735, 2013. doi:10.1109/FOCS.2013.83.

8 Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k-sat: An isolation lemma for k-cnfs. *J. Comput. Syst. Sci.*, 74(3):386–393, 2008. doi:10.1016/J.JCSS.2007.06.015.

9 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC 2006)*, pages 252–260, 2006. doi:10.1109/CCC.2006.6.

10 Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1246–1255, 2016. doi:10.1137/1.9781611974331.CH87.

11 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008. doi:10.1016/J.ARTINT.2007.11.002.

12 Huairui Chu, Mingyu Xiao, and Zhe Zhang. An improved upper bound for SAT. *Theor. Comput. Sci.*, 887:51–62, 2021. doi:10.1016/J.TCS.2021.06.045.

13 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158, 1971. doi:10.1145/800157.805047.

14 Radu Curticapean, Holger Dell, and Thore Husfeldt. Modular counting of subgraphs: Matchings, matching-splittable graphs, and paths. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA 2021)*, pages 34:1–34:17, 2021. doi:10.4230/LIPIcs.ESA.2021.34.

15 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. doi:10.1145/2925416.

16 Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability - Second Edition*, pages 669–692. IOS Press, 2021. doi:10.3233/FAIA200999.

17 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.

18 Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*, pages 4488–4494, 2017. doi:10.1609/AAAI.V31I1.11178.

19 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. doi:10.1145/1552285.1552286.

20 Fedor V. Fomin and Kjartan Høie. Pathwidth of cubic graphs and exact algorithms. *Inf. Process. Lett.*, 97(5):191–196, 2006. doi:10.1016/J.IPL.2005.10.012.

21 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.

22 Serge Gaspers and Gregory B. Sorkin. Separate, measure and conquer: Faster polynomial-space algorithms for max 2-csp and counting dominating sets. *ACM Trans. Algorithms*, 13(4):44:1–44:36, 2017. doi:10.1145/3111499.

23 Leslie Ann Goldberg and Marc Roth. Parameterised and fine-grained subgraph counting, modulo 2. *Algorithmica*, 86(4):944–1005, 2024. doi:10.1007/S00453-023-01178-0.

24 Gordon Hoi, Sanjay Jain, Ammar Fathin Sabili, and Frank Stephan. A bisection approach to subcubic maximum induced matching. In *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*, pages 257–272, 2024. doi:10.1007/978-981-97-0566-5_19.

25 Gordon Hoi, Sanjay Jain, and Frank Stephan. A faster exact algorithm to count X3SAT solutions. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP 2020)*, pages 375–391, 2020. doi:10.1007/978-3-030-58475-7_22.

26 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for ac^0. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 961–972, 1992. doi:10.1137/1.9781611973099.77.

27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

28 Kazuo Iwama. CNF satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, 18(2):385–391, 1989. doi:10.1137/0218026.

29 Sixue Liu. Chain, generalization of covering code, and deterministic algorithm for k-SAT. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 88:1–88:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.88.

30 Eliezer L. Lozinskii. Counting propositional models. *Inf. Process. Lett.*, 41(6):327–332, 1992. doi:10.1016/0020-0190(92)90160-W.

31 Burkhard Monien and Robert Preis. Upper bounds on the bisection width of 3- and 4-regular graphs. *J. Discrete Algorithms*, 4(3):475–498, 2006. doi:10.1016/J.JDA.2005.12.009.

32 Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and João Marques-Silva. Assessing heuristic machine learning explanations with model counting. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019)*, pages 267–278, 2019. doi:10.1007/978-3-030-24258-9_19.

33 Christos H. Papadimitriou and Stathis Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276, 1983. doi:10.1007/BFb0009651.

34 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.

35 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS 1997)*, pages 566–574, 1997. doi:10.1109/SFCS.1997.646146.

36 Junqiang Peng, Zimo Sheng, and Mingyu Xiao. New algorithms for #2-sat and #3-sat. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025)*, pages 2666–2674, 2025. doi:10.24963/IJCAI.2025/297.

37 Junqiang Peng and Mingyu Xiao. Further improvements for SAT in terms of formula length. *Inf. Comput.*, 294:105085, 2023. doi:10.1016/J.IC.2023.105085.

38 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996. doi:10.1016/0004-3702(94)00092-1.

39 Tian Sang, Paul Beame, and Henry A. Kautz. Performing bayesian inference by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, pages 475–482, 2005. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-075.php.

40 Dominik Scheder. PPSZ is better than you think. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 205–216, 2021. doi:10.1109/FOCS52979.2021.00028.

41 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 410–414, 1999. doi:10.1109/SFFCS.1999.814612.

42 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. doi:10.1137/0220053.

43 Patrick Traxler. The time complexity of constraint satisfaction. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC 2008)*, pages 190–201, 2008. doi:10.1007/978-3-540-79723-4_18.

44 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

45 Leslie G. Valiant. Accidental algorithms. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 509–517, 2006. doi:10.1109/FOCS.2006.7.

46 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

47 Magnus Wahlström. A tighter bound for counting max-weight solutions to 2sat instances. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC 2008)*, pages 202–213, 2008. doi:10.1007/978-3-540-79723-4_19.

A A Hardness Result for Parity-2-occ-SAT

The randomized Exponential Time Hypothesis (rETH) states that there is a constant $\varepsilon > 0$ such that no randomized algorithm can decide 3-SAT in time $O^*(2^{\varepsilon n})$ with error probability at most 1/3 (see also [17, Section 1.3]).

▶ **Theorem 22.** *Under rETH, there is no randomized algorithm that solves Parity-2-occ-SAT on positive formulas in $2^{o(n)}$ time, where n is the number of variables.*

Proof. Let $G = (V, E)$ be a graph with $n_v = |V|$ vertices and $m = |E|$ edges. A vertex cover of G is a set of vertices that includes at least one endpoint of every edge of G. An independent set of G is a set of vertices such that there is no edge between any two vertices. It is a basic property that $S \subseteq V$ is a vertex cover if and only if $V \setminus S$ is an independent set. An edge cover of G is a set of edges such that every vertex of G is an endpoint of at least

one edge of the set. For a vertex $v \in V$, we denote by $\delta(v) = \{e \in E \mid v \in e\}$ the set of edges incident to v. Parity-Vertex-Cover (resp., Parity-Independent-Set, Parity-Edge-Cover) is the problem of determining whether the number of vertex covers (resp., independent sets, edge covers) of G is odd. Let $\#vc(G)$ denote the number of vertex covers of G and $\#ec(G)$ the number of edge covers of G.

It is known that under rETH, Parity-Independent-Set cannot be solved in $2^{o(m)}$ time, where m is the number of edges [17, Proof of Theorem 1.2]. Since the complement of an independent set is a vertex cover, Parity-Independent-Set is equivalent to Parity-Vertex-Cover, which also implies a $2^{o(m)}$ lower bound for the latter. We first show that Parity-Edge-Cover is equivalent to Parity-Vertex-Cover. Then, we reduce Parity-Edge-Cover to Parity-2-occ-SAT on positive formulas in a natural way. We claim that

$$\#vc(G) \equiv \#ec(G) \pmod 2. \tag{1}$$

Then, the equivalence of Parity-Vertex-Cover and Parity-Edge-Cover follows from Eq. 1. The correctness of Eq. 1 follows from [15, Theorem 4.3]. Here, we note the proof for our case for completeness. Consider the following Inclusion-Exclusion formulation of $\#ec(G)$:

$$\begin{aligned}\#ec(G) &= 2^{|E(G)|} - \sum_{u \in V(G)} 2^{|E(G-\{u\})|} + \sum_{u,v \in V(G)} 2^{|E(G-\{u,v\})|} - \cdots \\ &= \sum_{S \subseteq V(G)} (-1)^{|S|} \cdot 2^{|E(G-S)|}.\end{aligned}$$

Thus, we have

$$\#ec(G) \equiv \sum_{\substack{S \subseteq V(G) \\ |E(G-S)|=0}} 1 \equiv \#vc(G) \pmod 2,$$

which completes the proof of Eq. 1. Next, we reduce Parity-Edge-Cover to Parity-2-occ-SAT.

Given a graph $G = (V, E)$, for each edge $e \in E$ we create a variable x_e. The formula is then defined as $\phi_G = \bigwedge_{v \in V(G)} (\bigvee_{e \in \delta(v)} x_e)$. Clearly, each variable appears twice positively, and each satisfying assignment to ϕ_G corresponds to an edge cover of G and vice versa.

In terms of parameters, the number of variables in ϕ_G is $n = |E| = m$. Since Parity-Vertex-Cover on m edges requires $2^{\Omega(m)}$ time under rETH and our reduction is parity-preserving with $n = m$, any algorithm solving Parity-2-occ-SAT in $2^{o(n)}$ time would directly imply a $2^{o(m)}$ algorithm for Parity-Vertex-Cover, contradicting rETH. ◀

We note that the above reduction also applies to show the $\oplus$P-Completeness of Parity-2-occ-SAT on positive formulas.