A Fast Algorithm for MaxSAT above Half Number of Clauses

Junqiang Peng, **Mingyu Xiao***
University of Electronic Science and Technology of China, Chengdu, China
jqpeng0@foxmail.com, myxiao@uestc.edu.cn

Abstract

We study the following parameterization of the MaxSAT problem: Given a CNF formula $\mathcal{F}$ with m clauses, decide whether at least $m/2+\mu$ clauses in $\mathcal{F}$ could be satisfied, where μ is the excess of the number of satisfied clauses over the trivial lower bound $m/2$ and is taken as the parameter. This perspective is known as the "above guarantee" parameterization. Since its introduction by Mahajan and Raman [1999], the analysis of parameterization above guarantee has become a highly active and fruitful line of research. In this paper, we develop a new algorithm with runtime $O^*(2.1479^{\mu})$, significantly improving the previous best upper bound $O^*(5.4064^{\mu})$ for this important problem. Here, the O^* notation omits polynomial factors.

1 Introduction

The BOOLEAN SATISFIABILITY problem (SAT), as the first proven NP-complete problem [Cook, 1971], is a cornerstone of computational complexity theory. Its optimization version, the MAXIMUM SATISFIABILITY problem (MaxSAT), holds an equal level of significance. Given a CNF formula, SAT is to find an assignment of the variables to satisfy all clauses, while MaxSAT is to satisfy the maximum number of clauses (or at least k clauses for an input integer k). SAT, MaxSAT, and many of their variants are arguably the most influential problems due to their significant applications in Computer Science, Artificial Intelligence, and many other fields. Although most of these problems are shown computationally hard [Schaefer, 1978; Garey and Johnson, 1979], their significance still drives various approaches to solve them, including heuristic, approximation, exact, and parameterized algorithms, etc. We refer to [Biere *et al.*, 2021; Fichte *et al.*, 2023] for more comprehensive surveys.

There has been extensive research on moderately exact exponential algorithms for SAT, MaxSAT, and their variants. The trivial algorithm that enumerates all possible assignments runs in time $O^*(2^n)$, where n is the number of variables. The popular Strong Exponential Time Hypothesis [Impagliazzo and Paturi, 2001] asserts that no algorithm solves SAT and MaxSAT in time $O^*((2-\epsilon)^n)$ for any constant $\epsilon > 0$. Therefore, different restricted versions and various measures or parameters have been considered [Dantsin and Hirsch, 2021; Samer and Szeider, 2021].

In recent years, several upper bounds for these problems have been improved. For SAT, Chu et al. (2021) presented an $O^*(1.2226^m)$-time algorithm, where m is the number of clauses in the formula, and Peng and Xiao (2023) designed an $O^*(1.0641^L)$-time algorithm, where L is the length of the formula. For MaxSAT, the upper bounds have been improved to $O^*(1.2886^m)$ [Xiao, 2022] and $O^*(1.0911^L)$ [Brilliantov *et al.*, 2023], respectively. Prior to this, most of these bounds had not seen any progress for decades. While pursuing better exact exponential algorithms for important NP-hard problems is worthwhile on its own theoretical side, this might also be potentially helpful for practical solving. Several techniques and principles behind these algorithms have been used in practical solvers [Chu *et al.*, 2023; Li *et al.*, 2023; Fichte *et al.*, 2023]. In addition, the bottlenecks and revealed structural obstructions might suggest some directions for understanding hard instances [Al-Yahya *et al.*, 2023].

We will study exact algorithms for MaxSAT. When measuring the computational complexity of MaxSAT, in addition to parameters n (the number of variables), m (the number of clauses) and L (the length of the input) mentioned above, there is another natural parameter: the number of satisfied clauses k. From the first algorithm with runtime $O^*(1.6181^k)$ by Mahajan and Raman [1999], the upper bound in terms of k has been improved several times. Currently, the best result is $O^*(1.3248^k)$ [Chen *et al.*, 2017]. For the case that k is small, MaxSAT can be solved efficiently. However, MaxSAT is only interesting for the case that k is relatively large ($k > \lceil m/2 \rceil$), since for any CNF formula with m clauses, either the all-true or the all-false assignment can satisfy at least $\lceil m/2 \rceil$ clauses. Motivated by this, Mahajan and Raman [1999] introduced the following "above guarantee" parameterization of MaxSAT with parameter μ, which is the excess of the number of satisfied clauses k over the lower bound $\lceil m/2 \rceil$.

MAXSAT ABOVE HALF (MaxSAT-AH)
Input: A CNF formula $\mathcal{F}$ with n variables and m clauses, and a positive integer μ (the parameter).
Question: Does there exist an assignment of n variables that satisfy at least $\lceil m/2 \rceil + \mu$ clauses?

*Contact Author

Can we solve MaxSAT-AH efficiently when μ is small, i.e., when k is close to $\lceil m/2 \rceil$? Mahajan and Raman [1999] affirmatively answered this question by proving that an $O^*(c^k)$-time algorithm implies an $O^*(c^{6\mu})$-time algorithm. Previously, MaxSAT-AH can be solved in time $O^*(5.4064^{\mu})$, which was derived from the current best result $O^*(1.3248^k)$ by using the above result [Chen *et al.*, 2017]. To the best of our knowledge, all previous improvements in the runtime bound with respect to μ were derived from improvements with respect to k. In other words, for the parameter μ, the paradigm for improvements introduced in [Mahajan and Raman, 1999] had stood as the sole one for more than two decades. This poses the following question: is there another way to achieve further improvements on the runtime bound with respect to μ?

We also highlight that since the introduction of the "above guarantee" parameterization for MaxSAT and MaxCut by Mahajan and Raman [1999], analysis of parameterization above guarantee has become a highly active and fruitful line of research, not only for MaxSAT-related problems [Gutin and Yeo, 2017], but also for many other important NP-hard problems. Examples include VERTEX COVER [Lokshtanov *et al.*, 2014; Garg and Philip, 2016], MULTIWAY CUT [Cygan *et al.*, 2013], and several others [Mahajan *et al.*, 2009]. For more comprehensive surveys, we refer to [Gutin and Yeo, 2017; Gutin and Mnich, 2022]. Currently, the "above guarantee" parameterization has become a standard and active research topic in Parameterized Complexity.

Our Contribution. In this paper, we prove that MaxSAT-AH can be solved in time $O^*(2.1479^{\mu})$ by developing a new algorithm that is tailored to the "above guarantee" parameter μ. This result significantly improves the previous $O^*(5.4064^{\mu})$. Our algorithm does not use the paradigm introduced by Mahajan and Raman [1999]. Thus, this is the first algorithm parameterized by μ that does not invoke algorithms parameterized by k in over two decades.

Our algorithm follows the branch-and-bound paradigm. Most previous algorithms for SAT and MaxSAT are also branch-and-bound algorithms and frequently use the simplest branching scheme that branches on a variable into two sub-branches by assigning value 1 or 0 to it. To get a running-time bound in terms of a parameter, we need to analyze how much the parameter decreases in each sub-branch. However, for the parameter $\mu = k - \lceil m/2 \rceil$, it may not always decrease even in a simple branching operation. For example, in a sub-branch, if the decrease of k is d, the decrease of m may be more than $2d$. The situation will get more complicated if the branching rule needs to assign several variables simultaneously. In general, there are several structural obstructions that prevent our parameter from decreasing, and these are the reasons why the existing branch-and-bound algorithms for MaxSAT can not be directly used to analyze a runtime bound with respect to μ. To address this issue, we develop two novel reduction rules, namely R-Rule 5 and R-Rule 6, to overcome these obstructions. As a result, our algorithm manages to efficiently decrease the parameter μ in each sub-branch, even if we apply some complicated branching operations.

In the last step of our algorithm, we handle a special case of the problem by reducing it to the MINIMUM SET COVER problem (MSC). This approach has already been used in the literature [Bliznets and Golovnev, 2012; Chen *et al.*, 2017; Xiao, 2022]. However, our case is different from those in the literature, and the methods used in [Bliznets and Golovnev, 2012; Chen *et al.*, 2017; Xiao, 2022] are not efficient enough to achieve our result. For this issue, we still employ this approach but make certain adaptations, which enable us to utilize the existing algorithm for MSC [van Rooij and Bodlaender, 2011] more effectively to avoid a bottleneck.

Finally, we can prove that the worst branch in our algorithm will generate a branching vector not worse than $(0.5, 1.5)$ (branching into two sub-branches: one reduces μ by at least 0.5 and the other reduces μ by at least 1.5), which implies our upper bound $O^*(2.1479^{\mu})$.

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

2 Preliminaries

A *boolean variable* (or simply *variable*) can be assigned value 1 (TRUE) or 0 (FALSE). A variable 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, and thus $\overline{\overline{x}} = x$. A *clause* is a set of literals. Given a set V of variables, a *CNF formula* (or simply *formula*) is a set of clauses on V. An *assignment* for V is a map $A : V \to \{0, 1\}$. A clause is *satisfied* by an assignment if at least one literal in it gets value 1 under the assignment. An assignment is an *optimal assignment* to a formula $\mathcal{F}$ if it satisfies the maximum number of clauses in $\mathcal{F}$. An *instance* of the MaxSAT problem, denoted by $(\mathcal{F}, k)$, asks whether there is an assignment for the variables that satisfies at least k clauses in $\mathcal{F}$.

For two sets of literals C_1 and C_2, we may simply write their union as C_1C_2. A clause containing a single literal x may be simply written as x. For a set of literals C, we use $\overline{C}$ to denote the set of negations of literals in C and use $C = 1$ (resp., $C = 0$) to denote that each literal in C is assigned value 1 (resp., 0). For a formula $\mathcal{F}$, we denote $\mathcal{F}[C = 1]$ as the resulting formula obtained from $\mathcal{F}$ by removing all clauses containing some literal in C and removing all literals in C from all clauses in $\mathcal{F}$, and let $\mathcal{F}[C = 0] = \mathcal{F}[\overline{C} = 1]$.

In a formula $\mathcal{F}$, a variable x is called a d-variable (resp., a d^+-variable) if x appears d (resp., at least d) times in $\mathcal{F}$, and a literal z is called an (i, j)-*literal* (resp., an (i^+, j)-*literal* or (i^-, j)-*literal*) if z appears i (resp., at least i or at most i) times and $\overline{z}$ appears j times in $\mathcal{F}$. Similarly, we can define (i, j^+)-literal, (i^+, j^+)-literal, etc. The *length* of a clause C, denoted by $|C|$, is the number of literals in C. A clause is a k-*clause* (resp., k^+-*clause*) if the length of it is k (resp., at least k). A clause C is called *unit clause* (resp., *non-unit clause*) if $|C| = 1$ (resp., $|C| \geq 2$). A literal x is called a *singleton* if all of its negations $\overline{x}$ are contained in unit clauses, and *non-singleton* otherwise. Let C be a set of literals in formula $\mathcal{F}$, we use $\mathrm{CLS}(C)$ to denote the set of all clauses in $\mathcal{F}$ that contain some literal in C, and use $\mathrm{UT}(C)$ to denote the set of all clauses that only contain literals in C, i.e.,

$$\mathrm{CLS}(C) = \{D \mid D \in \mathcal{F} \text{ and } D \cap C \neq \emptyset\}, \text{ and}$$
$$\mathrm{UT}(C) = \{D \mid D \in \mathrm{CLS}(C) \text{ and } D \subseteq C\}.$$

2.1 Branch-and-Bound Algorithms

Our algorithm is a branch-and-bound algorithm, which first exhaustively applies some reduction rule to reduce the instance and then searches for a solution by branching.

A reduction rule transforms, in polynomial time, an instance $\mathcal{I} = (\mathcal{F}, k)$ into another instance $\mathcal{I}' = (\mathcal{F}', k')$. A reduction rule is *correct* if $\mathcal{I}$ is a YES-instance if and only if $\mathcal{I}'$ is a YES-instance. We will use $(\mathcal{F}, k) \to (\mathcal{F}', k')$ to denote a reduction rule. In our algorithm, $k > k'$ holds for all reduction rules, and so they can only be applied a polynomial number of times and do not exponentially increase the running-time bound.

When no reduction rule can be applied, we do branching. The branching operation will generate a search tree and the size of the tree is related to the exponential part of the running-time bound. Let $\mu(\cdot)$ be the measure of an instance. In our algorithm, the measure will not increase in each step, and the problem can be solved in polynomial time if the measure becomes non-positive. A branching rule will branch on the current instance $\mathcal{I}$ into several sub-instances $\mathcal{I}_1, \ldots, \mathcal{I}_l$ such that $\mu(\mathcal{I}_i) < \mu(\mathcal{I})$ for each index $i \in [l]$. The branching rule is called *safe* if it holds that $\mathcal{I}$ is a YES-instance if and only if at least one of the sub-instances is a YES-instance. Let $T(\mu)$ be an upper bound on the size of the search tree generated by the algorithm on any instance with the measure of at most μ. For a branching rule, we will obtain a recurrence relation $T(\mu) \leq \sum_{i\in[l]} T(\mu - d_i)$, where $d_i = \mu - \mu(\mathcal{I}_i)$ for $i \in [l]$. This is also represented by a *branching vector* $(d_1, \ldots, d_l)$. The largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-d_i}$ is called the *branching factor* of the branching vector. If the largest branching factor among that of all branching vectors is γ, then $T(\mu) = O(\gamma^\mu)$. More details about analyzing branching algorithms can be found in [Fomin and Kratsch, 2010]. We say that one branching vector is *not worse* than another if its corresponding branching factor is not greater than that of the latter.

2.2 The Measure

It should be emphasized that once the measure $\mu \leq 0$, i.e., $k \leq \lceil m/2 \rceil$, the algorithm can report that the instance is a YES-instance and terminate since there is always an assignment satisfying at least $\lceil m/2 \rceil$ clauses. To facilitate the analysis, we will use the measure $\mu = k - m/2$ instead of $k - \lceil m/2 \rceil$. This will not affect the running-time bound since there is only a constant difference between them.

3 Reduction Rules

In this section, we introduce reduction rules (R-Rules for short) used in our algorithm. Reduction rules are typically developed to handle some special structures of the instance, and then we may be able to design more efficient branching rules for the problem. Although previous algorithms for MaxSAT have various reduction rules, our algorithm only employs four of them. This is because we must ensure that when applying a reduction rule, our measure μ does not increase, thus guaranteeing efficient running time. As a result, certain reduction rules from previous algorithms may not be applicable to our algorithm. Our algorithm consists of a total of six reduction rules, with the first four being well-known and used in the literature. The last two rules, which are newly developed, serve as crucial components of our algorithm.

R-Rule 1. $(\mathcal{F}' \wedge x\overline{x}C, k) \to (\mathcal{F}', k-1)$.

R-Rule 2 (Bansal and Raman 1999)**.** *If there is an* $(1,1)$*-literal* x *contained in clauses* xC *and* $\overline{x}D$*, then* $(\mathcal{F}' \wedge xC \wedge \overline{x}D, k) \to (\mathcal{F}' \wedge CD, k-1)$.

R-Rule 3 (Bansal and Raman 1999)**.** $(\mathcal{F}' \wedge xC \wedge \overline{x}C, k) \to (\mathcal{F}' \wedge C, k-1)$.

In R-Rule 3, we allow $C = \emptyset$. For this case, it is equivalent to deleting two clauses $\{x\}$ and $\{\overline{x}\}$ and decreasing k by 1.

R-Rule 4 (Chen and Kanj 2004)**.** *If there is an* (a,b)*-literal* x *contained in at least* b *unit clauses* x*, then* $(\mathcal{F}, k) \to (\mathcal{F}[x=1], k-a)$.

Note that if there is an $(1^+, 0)$-literal x, we can simply assign $x = 1$. Actually, this is a special case of R-Rule 4.

The following R-Rules 5 and 6 are newly introduced and crucial in our algorithm. They can guarantee a good complexity for our branching rules introduced later.

R-Rule 5. *Let* x *be an* $(a,1)$*-non-singleton and* $\overline{x}D$ *be the unique clause containing literal* $\overline{x}$*. Assume that there are* p *clauses containing some literal from* $\overline{D}$ *but no literal* x *or* $\overline{x}$*, and* q *clauses only containing literals from* D*. If* $p \leq q$*, then* $(\mathcal{F}, k) \to (\mathcal{F}[x=1], k-a)$.

Lemma 1. *R-Rule 5 is correct.*

Proof. It is sufficient to prove that there is optimal assignment A' to $\mathcal{F}$ such that x is assigned to value 1.

Let A be an optimal assignment to $\mathcal{F}$ that sets $x = 0$. If D is satisfied by A, then replacing $x = 0$ with $x = 1$ in A would not decrease the number of satisfied clauses since $\overline{x}D$ is the unique clause containing literal $\overline{x}$ that is already satisfied. Next, we assume that $D = 0$ in assignment A.

Let A' be an assignment obtained from A by replacing $xD = 0$ with $\overline{xD} = 0$ (i.e., each literal in xD is assigned 1 in A'). We show that A' is still an optimal assignment to $\mathcal{F}$.

Let $\mathcal{G}$ be the set of clauses containing literals in xD or $\overline{xD}$, i.e., $\mathcal{G} = \mathrm{CLS}(\{x\} \cup \{\overline{x}\} \cup D \cup \overline{D})$. Note that the number of satisfied clauses in $\mathcal{F} \setminus \mathcal{G}$ under assignments A and A' is equal. Therefore, we focus on the clauses in $\mathcal{G}$. On the one hand, at least q clauses in $\mathcal{G}$ are unsatisfied under assignment A because there are q clauses only containing literals in D, and assignment A sets $D = 0$. On the other hand, under the assignment A', at most p clauses in $\mathcal{G}$ would be unsatisfied. This is due to the fact that all clauses containing literals in xD are satisfied, and there are at most p clauses containing literals in $\overline{xD}$ but no literals in xD, which are the only possible clauses that might be unsatisfied. Since $p \leq q$, the number of satisfied clauses in $\mathcal{F}$ under assignment A' is not less than that under assignment A. This implies that A' is also an optimal assignment to $\mathcal{F}$ such that $x = 1$. □

R-Rule 6. *Let* x *be an* (a,b)*-literal such that there are* $a-1$ *unit clauses* $\{\overline{x}\}$*, and let* $xC_1, \cdots, xC_a$ *be the clauses containing literal* x*. Assume that there are* p *clauses containing some literal in* $\bigcup_{i=1}^{a} \overline{C_i}$ *but no literal* x *or* $\overline{x}$*, and* q *clauses only containing literals in* $\bigcup_{i=1}^{a} C_i$*. If* $p \leq q + a - 1$*, then* $(\mathcal{F}, k) \to (\mathcal{F}[x=0], k-b)$.

Lemma 2. *R-Rule 6 is correct.*

Proof. It suffices to show that there is an optimal assignment A' to $\mathcal{F}$ where x is assigned to value 0.

Let A be an optimal assignment to $\mathcal{F}$ that sets $x = 1$. If at least one of $C_i (1 \leq i \leq a)$ is satisfied under A, then replacing $x = 1$ with $x = 0$ in A would not decrease the number of satisfied clauses. This is because setting $x = 0$ satisfies $a - 1$ unit clauses $\{\overline{x}\}$, and makes at most $a - 1$ clauses among $xC_1, \cdots, xC_a$ become unsatisfied. Next, we assume that assignment A sets $L_C = 0$, where $L_C = \bigcup_{i=1}^{a} C_i$.

Let A' be an assignment obtained from A by replacing $\overline{x}L_C = 0$ with $x\overline{L_C} = 0$ (i.e., $x = 0$ and each literal in L_C is assigned 1 in A'). We show that A' is still an optimal assignment to $\mathcal{F}$.

Let $\mathcal{G}$ be the set of clauses containing some literals in $\overline{x}L_c$ or $x\overline{L_c}$, i.e., $\mathcal{G} = \mathrm{CLS}(\{x\} \cup \{\overline{x}\} \cup L_c \cup \overline{L_c})$. We only need to consider the clauses in $\mathcal{G}$ since assignment A and A' satisfy an equal number of clauses in $\mathcal{F} \setminus \mathcal{G}$. There are $a-1$ unit clauses $\{\overline{x}\}$ and q clauses only containing literals in L_C, which are all unsatisfied under assignment A since $x = 1$ and $L_C = 0$ in A. Thus, the number of clauses in $\mathcal{G}$ that are not satisfied under assignment A is at least $q+a-1$. However, assignment A' only results in a maximum of p unsatisfied clauses in $\mathcal{G}$. This is because under assignment A', all clauses containing literals in $\overline{x}L_C$ (which include all clauses with literal x or $\overline{x}$) are satisfied. Additionally, there are at most p clauses that contain some literals in $x\overline{L_C}$ but no literals in $\overline{x}L_C$, which are the only possible clauses that might not be satisfied under assignment A'. Since $p \leq q + a - 1$, the number of satisfied clauses in $\mathcal{F}$ under assignment A' is not less than that under assignment A. Therefore, assignment A' is also an optimal assignment to $\mathcal{F}$ where $x = 0$. □

Definition 1. *A formula is* reduced *if none of the above reduction rules can be applied to the formula.*

Lemma 3 (♣). *For any formula $\mathcal{F}$, applying any reduction rule will not increase the measure μ. Moreover, it takes polynomial time and polynomial space to apply the reduction rules to transfer $\mathcal{F}$ to a reduced one.*

Lemma 4 (♣). *In a reduced formula, all variables are 3^+-variables.*

Lemma 5. *Let $\mathcal{F}$ be a reduced formula and S be a set of distinct literals without complementary literals. Then, after assigning value 1 to all literals in S, the measure decreases by at least $\frac{|\mathrm{CLS}(S)|-|\mathrm{UT}(\overline{S})|}{2}$, i.e., $\mu(\mathcal{F}) - \mu(\mathcal{F}[S = 1]) \geq \frac{|\mathrm{CLS}(S)|-|\mathrm{UT}(\overline{S})|}{2}$.*

Proof. After assigning value 1 to all literals in S, all clauses containing some literals from S (i.e., $\mathrm{CLS}(S)$) would be satisfied and be removed from the formula. Besides, all clauses only containing literals from $\overline{S}$ (i.e., $\mathrm{UT}(\overline{S})$) would be removed from the formula since all literals in them are assigned value 0. Note that $\mathrm{CLS}(S)$ and $\mathrm{UT}(\overline{S})$ are disjoint. Consequently, after assigning value 1 to all literals in S, parameter m, the number of clauses, decreases by $|\mathrm{CLS}(S)|+|\mathrm{UT}(\overline{S})|$, and parameter k, the number of clauses that need to be satisfied, decreases by $|\mathrm{CLS}(S)|$. Therefore, the measure $\mu = k - m/2$ decreases by at least $|\mathrm{CLS}(S)| - \frac{|\mathrm{CLS}(S)|+|\mathrm{UT}(\overline{S})|}{2} = \frac{|\mathrm{CLS}(S)|-|\mathrm{UT}(\overline{S})|}{2}$. □

Lemma 6. *Let $\mathcal{F}$ be a reduced formula and x be an (a, b)-literal in it. If there are t unit clauses x (resp., $\overline{x}$) in $\mathcal{F}$, then branching on x results in a branching vector not worse than $(\frac{a}{2}, \frac{b-t}{2})$ (resp., $(\frac{a-t}{2}, \frac{b}{2})$).*

Proof. If $t = 0$, then the lemma holds by applying Lemma 5 to both sub-branches. Consider the case that there are $t > 0$ unit clauses $\{x\}$. Since R-Rule 3 can not be applied, there is no unit clause $\{\overline{x}\}$. Thus, by Lemma 5, the measure decreases by $\frac{a-0}{2}$ (resp., $\frac{b-t}{2}$) after assigning $x = 1$ (resp., $x = 0$). Similarly, for the case that there is $t > 0$ unit clauses $\{\overline{x}\}$, the branching vector is not worse than $(\frac{a-t}{2}, \frac{b}{2})$. □

4 The Main Algorithm

Our algorithm consists of three main parts: (1) applying the reduction rules iteratively until obtaining a reduced formula; (2) checking if $\mu \leq 0$ and, if so, reporting that the current instance is a YES-instance and terminating the algorithm; (3) employing branching rules to search for a solution. We have six branching steps, each of which is going to handle some type of literals. The branching vectors and branching factors in each step are listed in Table 1.

Steps	Literal Types	Vectors	Factors
1	6^+-variables $(4,1)$-literals	$(0.5, 1.5)$	**2.1479**
2	$(3,2)$-literals	$(1.5, 0.5)$ $(1,2)$	**2.1479** 1.6181
3	$(a \geq 2, 1)$-nsin	$(1,1)$	2
4	$(2,2)$-literals $(2,1)$-sin	$(1,1)$ $(1, 1.5)$ $(0.5, 1.5)$	2 1.7549 **2.1479**
5	$(3,1)$-sin in 2-clauses $(3,1)$-sin in 3-clauses	$(1, 1.5)$ $(1, 1.5, 2)$	1.7549 **2.1479**
6	$(3,1)$-sin in 4^+-clauses, reduced to MSC	-	2.1467

Table 1: A summary of branchings in the algorithm, where bottleneck cases are bold. In the table, "nsin" stands for "non-singletons" and "sin" stands for "singletons".

Before giving the detailed branching steps, we first show two frequently used properties.

Lemma 7. *Let x be an $(a, 1)$-literal with the unique clause containing literal $\overline{x}$ being $\overline{x}D$. It is safe to branch with (B1) $x = 1$; (B2) $x = 0$ and $D = 0$.*

Proof. Let A be an optimal assignment to $\mathcal{F}$ that sets $x = 0$. If D is satisfied by A, then setting $x = 1$ in A would not decrease the number of satisfied clauses since $\overline{x}D$ is the unique clause containing literal $\overline{x}$ that is already satisfied. Hence, there exists an optimal assignment with either $x = 1$ or with $x = 0$ and $D = 0$. □

Lemma 8. *Let x be an (a, b)-literal such that there are $a-1$ unit clauses $\{\overline{x}\}$. Let the clauses containing literal x be $xC_1, \cdots, xC_a$. It is safe to branch with (B1) $x = 0$; (B2) $x = 1$ and $C_i = 0$ for $1 \leq i \leq a$.*

Proof. Let A be an optimal assignment to $\mathcal{F}$ that sets $x = 1$. If at least one of $C_i (1 \leq i \leq a)$ is satisfied under A, setting $x = 1$ would make $a - 1$ unit clauses $\{\overline{x}\}$ to be satisfied, but only at most $a - 1$ clauses among $xC_1, \cdots, xC_a$ to be unsatisfied. Hence, there exists an optimal assignment with either $x = 0$ or with $x = 1$ and $C_i = 0$ for $1 \leq i \leq a$. □

Next, we are going to present our branching steps. When introducing one branching step, we assume that the current instance is a reduced instance and no previous branching step can be applied to it.

4.1 Step 1: 6^+-Variables and $(4, 1)$-Literals

Directly branching on a high-degree variable leads to a good branching vector. In this step, we deal with such variables.

B-Rule 1. *If there is a 6^+-variable or a $(4, 1)$-literal x, then branch with (B1) $x = 1$; (B2) $x = 0$.*

Lemma 9. *The branching vector of B-Rule 1 is not worse than $(1.5, 0.5)$.*

Proof. The following three cases cover all possible configurations for x.

Case 1: x is a $(4^+, 1)$-literal. There is no unit clause x since R-Rule 4 is not applicable, and there is at most one unit clause $\overline{x}$. By Lemma 6, we know that the branching vector is not worse than $(\frac{4-1}{2}, \frac{1}{2}) = (1.5, 0.5)$.

Case 2: x is a $(4^+, 2)$-literal. There is at most one unit clause x by R-Rule 4 and if there is a unit clause x, then there is no unit clause $\overline{x}$ by R-Rule 3. There are at most two unit clauses $\overline{x}$, and if there is a unit clause $\overline{x}$, then there is no unit clause x by R-Rule 3. By Lemma 6, the branching vector is not worse than $(\frac{4}{2}, \frac{2-1}{2}) = (2, 0.5)$ or $(\frac{4-2}{2}, \frac{2}{2}) = (1, 1)$.

Case 3: x is an (a, b)-literal, where $a \geq b \geq 3$. Since R-Rule 4 and R-Rule 3 can not be applied, we know that the formula does not contain unit clauses x and $\overline{x}$ simultaneously and it contains at most $b-1$ unit clauses x or at most $\min\{a-1, b\}$ unit clauses $\overline{x}$. By Lemma 6, the branching vector is not worse than $(\frac{a}{2}, \frac{b-(b-1)}{2}) = (\frac{a}{2}, \frac{1}{2})$ for the former case and not worse than $(\frac{a-(a-1)}{2}, \frac{b}{2}) = (\frac{1}{2}, \frac{b}{2})$ for the latter case. Since $a, b \geq 3$, the branching vector is not worse than $(1.5, 0.5)$. □

4.2 Step 2: $(3, 2)$-Literals

In this step, we deal with $(3, 2)$-literals. Let x be a $(3, 2)$-literal with five clauses $xC_1, xC_2, xC_3, \overline{x}D_1$, and $\overline{x}D_2$.

B-Rule 2. *If $D_1 \cup D_2 \neq \emptyset$, branch with (B1) $x = 1$; (B2) $x = 0$.*

Lemma 10. *The branching vector of B-Rule 2 is not worse than $(1.5, 0.5)$.*

Proof. There is at most one unit clause $\overline{x}$ since $D_1 \cup D_2 \neq \emptyset$. If there is no unit clause $\overline{x}$, then there is at most one unit clause x by R-Rule 4. By Lemma 6, the branching vector is not worse than $(\frac{3}{2}, \frac{2-1}{2}) = (1.5, 0.5)$. If there is one unit clause $\overline{x}$, then there is no unit clause x by R-Rule 3 and the branching vector is not worse than $(\frac{3-1}{2}, \frac{2}{2}) = (1, 1)$ by Lemma 6, which is also not worse than $(1.5, 0.5)$. □

B-Rule 3. *If $D_1 = D_2 = \emptyset$, branch with (B1) $x = 0$; (B2) $x = 1$ and $C_1 = C_2 = C_3 = 0$ according to Lemma 8.*

Lemma 11. *The branching vector of B-Rule 3 is not worse than $(1, 2)$.*

Proof. There are two unit clauses $\overline{x}$ (i.e., $|\mathrm{UT}(\overline{x})| = 2$) by the condition of this rule and no unit clause x by R-Rule 3. By Lemma 5, the measure decreases by at least $\frac{2-0}{2} = 1$ in the first sub-branch and $\frac{|\mathrm{CLS}(x\overline{C_1C_2C_3})|-|\mathrm{UT}(\overline{x}C_1C_2C_3)|}{2}$ in the second sub-branch. Let $p = |\mathrm{CLS}(\overline{C_1C_2C_3}) \setminus \mathrm{CLS}(x \cup \overline{x})|$ and $q = |\mathrm{UT}(C_1C_2C_3) \setminus \mathrm{CLS}(x \cup \overline{x})|$. We have $p \geq q + 3$ since R-Rule 6 can not be applied. With $|\mathrm{CLS}(x)| = 3$ and $|\mathrm{UT}(\overline{x})| = 2$, it holds that $|\mathrm{CLS}(x\overline{C_1C_2C_3})| - |\mathrm{UT}(\overline{x}C_1C_2C_3)| = (|\mathrm{CLS}(x)| + p) - (\mathrm{UT}(\overline{x}) + q) = (3 + p) - (2 + q) = 1 + p - q \geq 4$. Thus, the branching vector is not worse than $(1, \frac{4}{2}) = (1, 2)$. □

4.3 Step 3: $(2^+, 1)$-Non-Singletons

B-Rule 4. *If there is an $(2^+, 1)$-non-singleton x with the unique clause containing $\overline{x}$ being $\overline{x}D$, then branch with (B1) $x = 1$; (B2) $x = 0$ and $D = 0$ according to Lemma 7.*

Lemma 12. *The branching vector of B-Rule 4 is not worse than $(1, 1)$.*

Proof. Let x be an $(a, 1)$-non-singleton such that $a \geq 2$. By Lemma 5, the measure decreases by at least $\frac{a-0}{2} = \frac{a}{2}$ in the first sub-branch since $D \neq \emptyset$, and at least $\frac{|\mathrm{CLS}(\overline{xD})|-|\mathrm{UT}(xD)|}{2}$ in the second sub-branch. Since that x is an $(a, 1)$-non-singleton and R-Rule 4 is not applicable, we know that $|\mathrm{CLS}(\overline{x})| = 1$ and $|\mathrm{UT}(x)| = 0$. Let $p = |\mathrm{CLS}(\overline{D}) \setminus \mathrm{CLS}(x \cup \overline{x})|$ and $q = |\mathrm{UT}(D) \setminus \mathrm{CLS}(x \cup \overline{x})|$. we have $p \geq q + 1$, other R-Rule 5 can be applied. Thus, it holds that $|\mathrm{CLS}(\overline{xD})|-|\mathrm{UT}(xD)| = (|\mathrm{CLS}(\overline{x})|+p)-(|\mathrm{UT}(x)|+q) = 1 + p - q \geq 2$. Thus, the branching vector is not worse than $(\frac{a}{2}, \frac{2}{2})$, and so not worse than $(1, 1)$ since $a \geq 2$. □

4.4 Step 4: $(2, 2)$-Literals and $(2, 1)$-Singletons

B-Rule 5. *If there is a $(2, 2)$-literal x such that there is no unit clause x or $\overline{x}$, then branch with (B1) $x = 1$; (B2) $x = 0$.*

Lemma 13. *The branching vector of B-Rule 5 is not worse than $(1, 1)$.*

Proof. By Lemma 6, the branching vector is not worse than $(\frac{2-0}{2}, \frac{2-0}{2}) = (1, 1)$. □

B-Rule 6. *Let x be a $(2, 2)$-literal or $(2, 1)$-literal such that there is exactly one unit clause $\overline{x}$. Let the two clauses containing x be xC_1 and xC_2. Branch with (B1) $x = 0$; (B2) $x = 1$ and $C_1 = C_2 = 0$ according to Lemma 8.*

Lemma 14. *The branching vector of B-Rule 6 is not worse than $(0.5, 1.5)$.*

Proof. There is no unit clause x, otherwise R-Rule 3 can be applied. Thus, we know $C_1, C_2 \neq \emptyset$. By Lemma 5, in the first sub-branch, the measure decreases by at least $\frac{2-0}{2} = 1$ if x is a $(2,2)$-literal and at least $\frac{1-0}{2} = 0.5$ if x is a $(2,1)$-literal.

By Lemma 5 again the measure decreases by at least $\frac{|\mathrm{CLS}(x\overline{C_1C_2})|-|\mathrm{UT}(\overline{x}C_1C_2)|}{2}$ in the second sub-branch. Let $p = |\mathrm{CLS}(x\overline{C_1C_2}) \setminus \mathrm{CLS}(x \cup \overline{x})|$ and $q = |\mathrm{UT}(\overline{x}C_1C_2) \setminus \mathrm{CLS}(x \cup \overline{x})|$. We have $p \geq q + 2$, otherwise R-Rule 6 can be applied. With $|\mathrm{CLS}(x)| = 2$ and $|\mathrm{UT}(\overline{x})| = 1$, we have $|\mathrm{CLS}(x\overline{C_1C_2})| - |\mathrm{UT}(\overline{x}C_1C_2)| = (|\mathrm{CLS}(x)| + p) - (|\mathrm{UT}(\overline{x})| + q) = (2 + p) - (1 + q) = 1 + p - q \geq 3$.

Thus, the branching vector is not worse than $(1, \frac{3}{2}) = (1, 1.5)$ if x is a $(2,2)$-literal and not worse than $(0.5, \frac{3}{2}) = (0.5, 1.5)$ if x is a $(2,1)$-singleton. □

Lemma 15. *After Step 4, the formula only contains $(3,1)$-singletons and their negations.*

Proof. By Lemma 4, all variables in a reduced formula are 3^+-variables. Step 1 deals with all 6^+-variables and $(4,1)$-literals. Step 2 deals with all $(3,2)$-literals. After Step 2, the formula only contains 4-variables and 3-variables. For a $(2,1)$-literal, it is handled by Step 3 if it is a $(2,1)$-non-singleton and handled by Step 4 otherwise. All $(2,2)$-literals are handled by Step 4. Moreover, all $(3,1)$-non-singletons are handled by Step 3. Consequently, the remaining literals in the formula are $(3,1)$-singletons and their negations. □

4.5 Step 5: $(3,1)$-Singletons in Short Clauses

In this step, we are going to deal with $(3,1)$-singletons contained in clauses with a length of 2 or 3.

Definition 2. *An instance is* simplified *if the formula only contains $(2^+,1)$-singletons and their negations.*

By Lemma 15, we know that the current instance is a simplified instance. We assume w.l.o.g. that all literals in unit clauses are negative literals in a simplified instance. By this assumption, we have that $|\mathrm{CLS}(x)| = 3$ for any positive literal x in the formula and $|\mathrm{UT}(\overline{x})| = 1$ for any negative literal $\overline{x}$ in the formula.

First, we introduce a structural property of simplified instances, which allows us to apply a better branching.

Lemma 16. *A simplified instance has an optimal assignment that satisfies all non-unit clauses.*

Proof. Let A be an optimal assignment that satisfies the maximum number of non-unit clauses. Assume that there is a non-unit clause C that is unsatisfied under A. Let x be some literal in C. Then assignment A sets $x = 0$. Since x is a $(2^+,1)$-singleton, there is only one unit clause $\overline{x}$ that contains $\overline{x}$ and it is satisfied by A. Let A' be the assignment obtained from A by replacing $x = 0$ with $x = 1$. Then, assignment A' satisfies the non-unit clause C and makes at most one unit clause $\overline{x}$ unsatisfied. This shows that the number of clauses satisfied by A' is not less than that by A. However, the number of non-unit clauses satisfied by A' is greater than that by A, a contradiction to the assumption on A. □

Based on Lemma 16, we apply the following two branching rules to eliminate 2-clauses and 3-clauses.

B-Rule 7. *If there is a 2-clause xy, then branch with (B1) $x = 1$; (B2) $x = 0$ and $y = 1$.*

Lemma 17. *The branching vector of B-Rule 7 is not worse than $(1, 1.5)$.*

Proof. By Lemma 5, we know that the measure decreases by at least $\frac{3-1}{2} = 1$ in the first sub-branch and at least $\frac{|\mathrm{CLS}(\overline{x}y)|-|\mathrm{UT}(x\overline{y})|}{2}$ in the second sub-branch.

Since that a negative literal is contained in a unique unit clause, and a unit clause only contains a negative literal, we have that $|\mathrm{CLS}(\overline{x}y)| = |\mathrm{CLS}(\overline{x})| + |\mathrm{CLS}(y)| = 1 + 3 = 4$ and $|\mathrm{UT}(x\overline{y})| = |\mathrm{UT}(x)| + |\mathrm{UT}(\overline{y})| = 0 + 1 = 1$. Thus, the branching vector is not worse than $(1, \frac{|\mathrm{CLS}(\overline{x}y)|-|\mathrm{UT}(x\overline{y})|}{2}) = (1, \frac{4-1}{2}) = (1, 1.5)$. □

B-Rule 8. *If there is a 3-clause xyz, then branch with (B1) $x = 1$; (B2) $x = 0$ and $y = 1$; (B3) $x = y = 0$ and $z = 1$.*

Lemma 18. *The branching vector of B-Rule 8 is not worse than $(1, 1.5, 2)$.*

Proof. By the same argument in the proof of Lemma 17, we can show that the measure decreases by at least 1 and 1.5 in the first and second sub-branches respectively.

Consider the third sub-branch. We have $|\mathrm{CLS}(\overline{x}\overline{y}z)| = |\mathrm{CLS}(\overline{x})| + |\mathrm{CLS}(\overline{y})| + |\mathrm{CLS}(z)| = 1 + 1 + 3 = 5$ since that a negative literal is only contained in a unit clause, and a unit clause only contains negative literal. Note that all 2-clauses have been handled by B-Rule 7, the length of a non-unit clause is at least 3 now. Thus, $|\mathrm{UT}(xy\overline{z})| = |\mathrm{UT}(xy)| + |\mathrm{UT}(\overline{z})| = 0 + 1 = 1$ holds. By Lemma 5, the measure decreases by at least $\frac{|\mathrm{CLS}(\overline{x}\overline{y}z)|-|\mathrm{UT}(xy\overline{z})|}{2} = \frac{5-1}{2} = 2$. Therefore, the branching vector is not worse than $(1, 1.5, 2)$. □

4.6 Step 6: Fast Solving the Remaining Part

After Step 5, all non-unit clauses in $\mathcal{F}$ are 4^+-clauses. Recall that we denote by n the number of variables and by m the number of clauses in $\mathcal{F}$, and there are n unit clauses and $m-n$ non-unit clauses in $\mathcal{F}$.

Lemma 19. *If $0.7m + 0.037n \geq k$ holds in a simplified instance $(\mathcal{F}, k)$ with all non-unit clauses being 4^+-clauses, then we can find an assignment satisfying at least k clauses in polynomial time.*

Proof. We set $p = 0.26$ and assign each variable with value 1 with probability p. Then, each unit clause is satisfied with a probability of $1 - p$. Since every non-unit clause contains at least 4 positive literals, a non-unit clause is satisfied with a probability of $1 - (1-p)^4$. Thus, by the linearity of expectation and setting $p = 0.26$, the expected number of satisfied clauses under this random assignment would be

$$\begin{aligned} & n(1-p) + (m-n)(1-(1-p)^4) \\ \geq\ & 0.74n + 0.7(m-n) \\ \geq\ & 0.7m + 0.04n \geq 0.7m + 0.037n \geq k. \end{aligned}$$

By using the method of conditional expectation, we can deterministically construct an assignment that satisfies at least k clauses in polynomial time. □

Next, we assume that $0.7m + 0.037n < k$. Thus, $0.6m + 0.11n < 3(k - 0.5m) = 3\mu$. For this case, we follow the approach proposed in [Bliznets and Golovnev, 2012] and reduce the problem to the MINIMUM SET COVER problem, which is formally defined as follows.

MINIMUM SET COVER (MSC)
Input: A universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$; A positive integer s.
Question: Does there exist a subfamily $\mathcal{C} \subseteq \mathcal{S}$ such that $|\mathcal{C}| \leq s$ and the union of $\mathcal{C}$ is $\mathcal{U}$?

An instance of MSC is denoted by $(\mathcal{U}, \mathcal{S}, s)$. A subfamily $\mathcal{C} \subseteq \mathcal{S}$ is a *set cover* for $\mathcal{S}$ if $\bigcup_{D \in \mathcal{C}} D = \mathcal{U}$. We use $|\mathcal{U}|$ to denote the number of elements in universe $\mathcal{U}$ and $|\mathcal{S}|$ to denote the number of subsets in family $\mathcal{S}$. For an element $e \in \mathcal{U}$, the frequency of e, denoted by $f(e)$, is defined as the number of subsets in $\mathcal{S}$ that contain e. It was shown in [van Rooij and Bodlaender, 2011] that MINIMUM SET COVER can be solved in time $O^*(1.28759^{k(\mathcal{U},\mathcal{S})})$, where $k(\mathcal{U}, \mathcal{S}) = \sum_{e \in \mathcal{U}} v(f(e)) + \sum_{S \in \mathcal{S}} w(|S|)$, and v and w are weight functions $\mathbb{N} \to \mathbb{R}_+$. The maximum value of v is 0.595723 and the maximum value of w is 1. We note that the maximum value of w for sets of size at most 3 is 0.706023 (refer to the end of section 3 in [van Rooij and Bodlaender, 2011]). Thus, the following result can be utilized.

Lemma 20 (van Rooij and Bodlaender 2011)**.** *If each subset in $\mathcal{S}$ has a size of at most 3, then* MINIMUM SET COVER *can be solved in time $O^*(1.28759^{0.595723|\mathcal{U}|+0.706023|\mathcal{S}|}) \subseteq O^*(1.29^{0.6|\mathcal{U}|+0.71|\mathcal{S}|})$ and polynomial space.*

Given a simplified instance $\mathcal{I} = (\mathcal{F}, k)$ of MaxSAT, we reduce it to an instance $\mathcal{I}' = (\mathcal{U}_\mathcal{F}, \mathcal{S}_\mathcal{F}, m - k)$ of MSC as follows. For each non-unit clause C_z in $\mathcal{F}$, there is an element e_{C_z} in the universe $\mathcal{U}_\mathcal{F}$. For each variable x_i in $\mathcal{F}$, there is a subset $S_{x_i} = \{e_{C_z} \mid \text{literal } x_i \text{ appears in clause } C_z\}$ in the family $\mathcal{S}_\mathcal{F}$. We have $|\mathcal{U}_\mathcal{F}| = m - n$ and $|\mathcal{S}_\mathcal{F}| = n$. Moreover, each subset in $\mathcal{S}_\mathcal{F}$ has a size of 3, since each positive literal x_i is a $(3, 1)$-singleton. This reduction has been used in several algorithms for MaxSAT. The proof of the correctness of this reduction (i.e., Lemma 21) is deferred to the full version of this paper.

Lemma 21 (♣). *The simplified instance $\mathcal{I}$ of MaxSAT is a YES-instance if and only if the instance $\mathcal{I}'$ of MSC is a YES-instance.*

Lemma 22. *Each execution of Step 6 can be done in time $O^*(2.1467^\mu)$ and polynomial space.*

Proof. By reducing a simplified instance to an instance of MCS with $|\mathcal{U}| = m - n$ and $|\mathcal{S}| = n$, we can solve the problem in time $O^*(1.29^{0.6n+0.71(m-n)})$ by Lemma 20. Note that $0.6(m - n) + 0.71n = 0.6m + 0.11n < 3\mu$, we have $1.29^{3k} < 2.1467^k$, which proves the lemma. □

4.7 The Final Result

Now we are prepared to present our main result. The algorithm will execute the above six branching steps in order. Before applying each branching step, reduction rules are exhaustively applied until the instance is reduced. It is important to note that, before each step, if the measure $\mu \leq 0$, then the algorithm will report that the input instance is a YES-instance and terminate. All reduction rules can be executed in polynomial time. As shown in Table 1, for the first five steps, the largest branching factor is 2.1479, corresponding to the branching vector $(1.5, 0.5)$ in Steps 1, 2, 4, and 5. The last step (Step 6) can be executed in time $O^*(2.1467^\mu)$ by Lemma 22. Thus, the problem can be solved in time $O^*(2.1479^\mu)$. Clearly, each step requires only polynomial space. In conclusion, we obtain

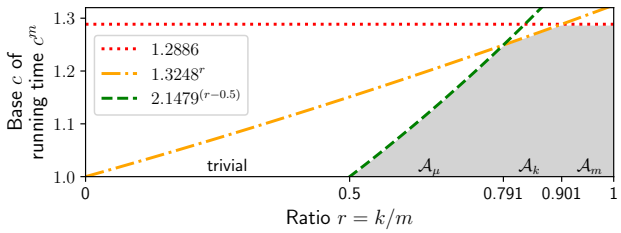

Figure 1: The bases of running-time bounds (which have been converted into bounds in terms of m) of algorithms $\mathcal{A}_\mu$, $\mathcal{A}_k$, and $\mathcal{A}_m$ under different values of $r = k/m$. The shaded area is unexplored.

Theorem 1. *MaxSAT can be solved in $O^*(2.1479^\mu)$ time and polynomial space, where $\mu = k - \lceil m/2 \rceil$.*

5 Discussion and Conclusion

In this paper, we have presented a new algorithm for MaxSAT with running-time bound $O^*(2.1479^\mu)$, where $\mu = k - \lceil m/2 \rceil$. We compare our algorithm with the algorithms parameterized by k and m. Denote by $\mathcal{A}_\mu$ our algorithm, by $\mathcal{A}_k$ the $O^*(1.3248^k)$-time algorithm [Chen *et al.*, 2017], and by $\mathcal{A}_m$ the $O^*(1.2886^m)$-time algorithm [Xiao, 2022]. Let $r = k/m$ be the ratio of clauses in a formula that one wants to satisfy. The bounds $O^*(1.3248^k)$ and $O^*(2.1479^\mu)$ can be expressed in terms of m, i.e., $O^*(1.3248^{rm})$ and $O^*(2.1479^{(r-0.5)m})$, respectively. Figure 1 shows the front line of these algorithms, which indicates that our algorithm $\mathcal{A}_\mu$ is faster when $0.5 < r \leq 0.791$, $\mathcal{A}_k$ is superior for $0.791 < r \leq 0.901$, and $\mathcal{A}_m$ excels for $0.901 < r \leq 1$.

In our algorithm, the two newly introduced reductions R-Rule 5 and R-Rule 6 are crucial in our worst-case-based analysis. Whether they also have a good performance and application scenarios in practical solving deserves further research.

Another direction for further study is to consider other "above guarantee" parameterizations of MaxSAT. Given a formula $\mathcal{F}$ and a parameter μ', an "above guarantee" parameterization of MaxSAT asks whether $g(\mathcal{F}) + \mu'$ clauses can be satisfied. Here, $g(\mathcal{F})$ is a lower bound on the maximum number of clauses that can be satisfied. The lower bound considered in this paper is $\lceil m/2 \rceil$. Different lower bounds can be obtained if some constraints are imposed on the input CNF-formula. For MaxSAT above other lower bounds, most existing algorithms are based on reducing to other problems [Kim and Williams, 2011; Crowston *et al.*, 2012; Gutin *et al.*, 2013]. It would be interesting to investigate whether other paradigms (e.g., the branch-and-bound paradigm) are applicable to design better algorithms for these "above guarantee" parameterizations of MaxSAT.

Acknowledgements

This work is supported by National Natural Science Foundation of China under grants 62372095 and 62172077, and Natural Science Foundation of Sichuan Province of China under grant 2023NSFSC0059.

References

[Al-Yahya *et al.*, 2023] Tasniem Nasser Al-Yahya, Mohamed El Bachir Menai, and Hassan Mathkour. On the structure of the boolean satisfiability problem: A survey. *ACM Comput. Surv.*, 55(3):46:1–46:34, 2023.

[Bansal and Raman, 1999] Nikhil Bansal and Venkatesh Raman. Upper bounds for maxsat: Further improved. In *ISAAC 1999*, pages 247–258, 1999.

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

[Bliznets and Golovnev, 2012] Ivan Bliznets and Alexander Golovnev. A new algorithm for parameterized max-sat. In *Parameterized and Exact Computation, IPEC 2012*, pages 37–48. Springer, 2012.

[Brilliantov *et al.*, 2023] Kirill Brilliantov, Vasily Alferov, and Ivan Bliznets. Improved algorithms for maximum satisfiability and its special cases. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2023*, volume 37, pages 3898–3905, 2023.

[Chen and Kanj, 2004] Jianer Chen and Iyad A Kanj. Improved exact algorithms for max-sat. *Discret. Appl. Math.*, 142(1):17–27, 2004.

[Chen *et al.*, 2017] Jianer Chen, Chao Xu, and Jianxin Wang. Dealing with 4-variables by resolution: An improved maxsat algorithm. *Theoretical Computer Science*, 670:33–44, 2017.

[Chu *et al.*, 2021] Huairui Chu, Mingyu Xiao, and Zhe Zhang. An improved upper bound for SAT. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 3707–3714. AAAI Press, 2021.

[Chu *et al.*, 2023] Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, pages 3915–3923. AAAI Press, 2023.

[Cook, 1971] 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. ACM, 1971.

[Crowston *et al.*, 2012] Robert Crowston, Gregory Z. Gutin, Mark Jones, and Anders Yeo. A new lower bound on the maximum number of satisfied clauses in max-sat and its algorithmic applications. *Algorithmica*, 64(1):56–68, 2012.

[Cygan *et al.*, 2013] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Trans. Comput. Theory*, 5(1):3:1–3:11, 2013.

[Dantsin and Hirsch, 2021] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability - Second Edition*, pages 669–692. IOS Press, 2021.

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

[Fomin and Kratsch, 2010] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Garg and Philip, 2016] Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above A higher guarantee. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 1152–1166. SIAM, 2016.

[Gutin and Mnich, 2022] Gregory Z. Gutin and Matthias Mnich. A survey on graph problems parameterized above and below guaranteed values. *CoRR*, abs/2207.12278, 2022.

[Gutin and Yeo, 2017] Gregory Z. Gutin and Anders Yeo. Parameterized constraint satisfaction problems: a survey. In *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7, pages 179–203. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[Gutin *et al.*, 2013] Gregory Z. Gutin, Mark Jones, Dominik Scheder, and Anders Yeo. A new bound for 3-satisfiable maxsat and its algorithmic application. *Inf. Comput.*, 231:117–124, 2013.

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

[Kim and Williams, 2011] Eun Jung Kim and Ryan Williams. Improved parameterized algorithms for above average constraint satisfaction. In *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011*, volume 7112, pages 118–131. Springer, 2011.

[Li *et al.*, 2023] Shuolin Li, Chu-Min Li, Mao Luo, Jordi Coll, Djamal Habet, and Felip Manyà. A new variable ordering for in-processing bounded variable elimination in SAT solvers. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023*, pages 1979–1987. ijcai.org, 2023.

[Lokshtanov *et al.*, 2014] Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014.

[Mahajan and Raman, 1999] Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.

[Mahajan *et al.*, 2009] Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *J. Comput. Syst. Sci.*, 75(2):137–153, 2009.

[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 2023*, pages 2004–2012. ijcai.org, 2023.

[Samer and Szeider, 2021] Marko Samer and Stefan Szeider. Fixed-parameter tractability. In *Handbook of Satisfiability - Second Edition*, pages 693–736. IOS Press, 2021.

[Schaefer, 1978] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978.

[van Rooij and Bodlaender, 2011] Johan M. M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discret. Appl. Math.*, 159(17):2147–2164, 2011.

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