Resolution and Domination: An Improved Exact MaxSAT Algorithm

Chao Xu[1] , **Wenjun Li**[2] , **Yongjie Yang**[1,3] , **Jianer Chen**[4] and **Jianxin Wang**[1*]
[1]School of Computer Science and Engineering, Central South University, Changsha, China
[2]School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha, China
[3]Chair of Economic Theory, Saarland University, Saarbrücken, Germany
[4]Department of Computer Science and Engineering, Texas A&M University, College Station, USA
{xuchaofay, liwenjun, jxwang}@csu.edu.cn, yyongjiecs@gmail.com, chen@cse.tamu.edu

Abstract

We study the MAXIMUM SATISFIABILITY problem (MAXSAT). Particularly, we derive a branching algorithm of running time $O^*(1.2989^m)$ for the MAXSAT problem, where m denotes the number of clauses in the given CNF formula. Our algorithm considerably improves the previous best result $O^*(1.3248^m)$ by Chen and Kanj [2004] published 15 years ago. For our purpose, we derive improved branching strategies for variables of degrees 3, 4, and 5. The worst case of our branching algorithm is at variables of degree 4 which occur twice both positively and negatively in the given CNF formula. To serve the branching rules and shrink the size of the CNF formula, we also propose a variety of reduction rules which can be exhaustively applied in polynomial time and, moreover, some of them solve a bottleneck of the previous best algorithm.

1 Introduction

The SATISFIABILITY problem and many of its variants are arguably the most influential problems due to their significant applications in a wide range of areas. Though most of these problems turn out to be computationally hard [Garey and Johnson, 1979; Garey *et al.*, 1976; Schaefer, 1978], their significance still drives both theorists and practitioners to derive approaches to solve the problems. Prominent approaches include deriving approximation, randomized, heuristic algorithms, etc. [Poloczek *et al.*, 2017; Bliznets, 2013; Goemans and Williamson, 1994; Luo *et al.*, 2017; Poloczek *et al.*, 2017; Cai *et al.*, 2014]. In addition, many powerful solvers for these problems have been developed [Hutter *et al.*, 2017; Sohanghpurwala *et al.*, 2017]. However, given that these problems are intrinsically hard, none of these heuristic-based solvers is able to solve all instances. Another important and complementary line of research is developing exact algorithms. First, with the rapid development of computer techniques, many exponential-time algorithms (especially single-exponential algorithms) can be also implemented in an acceptable time. This motivates researchers to improve the running times of exact algorithms for many NP-hard problems (see, e.g., [Chen *et al.*, 2010; Gupta *et al.*, 2018]). Second, principles behind exact algorithms might be useful for practitioners to improve their heuristic algorithms. For instance, combined with refined pruning techniques, branching strategies in exact branching algorithms can be employed in heuristic or machine learning algorithms [Sturtevant, 2003; Marques-Silva, 1999], and preprocessing procedures in many exact algorithms can be even directly used in heuristic algorithms [Wang *et al.*, 2018].

running times	references
$O^*(1.3803^m)$	[Niedermeier and Rossmanith, 1999]
$O^*(1.3413^m)$	[Bansal and Raman, 1999]
$O^*(1.3248^m)$	[Chen and Kanj, 2004]
$O^*(1.2989^m)$	**this paper**

Table 1: Important progress of exact MAXSAT algorithms.

In this paper, we study an exact algorithm for the (decision version) of the MAXIMUM SATISFIABILITY (MAXSAT) problem, which has significant applications in many areas (see [Morgado *et al.*, 2013] for a survey). Particularly, in this problem, we are given a Conjunctive Normal Form (CNF) formula F with m clauses and n Boolean variables, together with an integer k, and the question is whether there is an assignment satisfying at least k clauses in F. It has long been known that the MaxSAT problem is NP-hard, even in several special cases such as when every clause consists of only two literals and each variable occurs in at most three clauses [Raman *et al.*, 1998]. Our main result is an exact branching algorithm of running time $O^*(1.2989^m)$, which significantly improves the previous best result $O^*(1.3248^m)$ by Chen and Kanj [2004] published 15 years ago. To this end, we propose several new reduction rules to shrink the size of the input instances. Moreover, based on the reduction rules and by refined observations, we put forward novel branching strategies for variables of degrees 3, 4, and 5. Table 1 summarizes recent progress of exact algorithms for the MAXSAT problem.

Finally, it should be pointed out that with respect to the parameter n, the MAXSAT problem cannot be solved in $O^*(2^{cn})$ time for any constant $c < 1$ in the worst case, assuming the Strong Exponential Time Hypothesis

*Contact Author

(SETH) [Impagliazzo and Paturi, 2001]. In addition, Chen and Kanj [2004] also presented an algorithm of running time $O^*(1.3695^k)$ which has been improved to $O^*(1.325^k)$ very recently [Chen *et al.*, 2017]. Our algorithm does not improve this bound.

2 Preliminary

We assume that the reader is familiar with the basics of propositional logic. A Boolean *variable* is a variable that can be assigned either the value 1 (TRUE) or 0 (FALSE). Throughout this paper, we simply use variables for Boolean variables. A *literal* is either a variable x or its negation $\bar{x}$. Note that the negation of $\bar{x}$ is x, i.e., $\bar{\bar{x}} = x$. A *clause* is a disjunction of several literals. For instance, $C = x_1 \vee x_2 \vee x_3$ is a clause of three literals x_1, x_2, and x_3. For simplicity, we omit the symbol $\vee$. Note that the order of the literals in a clause is irrelevant to the clause's identity. Hence, $x_1x_2x_3 = x_2x_1x_3$. For two clauses C_1 and C_2, we use C_1C_2 to denote the clause consisting of all literals included in at least one of C_1 and C_2. For a clause C and a literal x, $C \setminus x$ is the clause C without the literal x. A *conjunctive norm form* (CNF) formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ is a conjunction of clauses. An *assignment* f is a function which assigns to each variable x of F a value $f(x) \in \{0, 1\}$. A clause C is *satisfied* by f if at least one literal in C gets the value 1. An *optimal assignment* refers to one that satisfies the maximum number of clauses. For simplicity, for a clause C, we use $f(C) = 0$ (or $C = 0$) for that every literal in C has the value 0. In this case, C is unsatisfied by f. However, by $f(C) = 1$ (or $C = 1$) we mean that at least one literal of C obtains the value 1 from f, to indicate that the clause C is satisfied by f.

A literal x is an *i-literal* if x occurs i times in F. In addition, we say that x is an *(i, j)-literal* if x and $\bar{x}$ appear respectively i and j times in F. The *degree* of the variable x is then defined as $i + j$. A variable x of degree h (resp. at least h) is also called an *h-variable* (resp. *h^+-variable*). The *size* of a clause C, denoted $|C|$, is the number of literals in C. A clause is an *h-clause* if its size is h, and an *h^+-clause* if its size is at least h. In particular, a 1-clause is also called a *unit* clause. An *$(i, 1)$-literal* x is called a *singleton* (resp. *non-singleton*) if the literal $\bar{x}$ appears in a unit clause $\bar{x}$ (resp. a 2^+-clause).

Resolution on a variable x in F is the operation that for every pair of clauses xC and $\bar{x}D$ in F, first creates a clause CD, and then removes from F all clauses including the variable x.

3 Reduction Rules

In this section, we present numerous reduction rules, some of which have been studied in the literature but many are new. These reduction rules are designed for eliminating some specific combinatorial structures in order for the applications of branching rules studied in the next section and shrinking the size of the given formula. A CNF formula is *reduced* by some reduction rules if none of the reduction rules applies to this formula. Let F be a given CNF formula. In the following, before introducing the j-th reduction rule, $j > 1$, we assume that F is reduced by all the reduction rules introduced before. For two instances (F, k) and (F', k'), $(F, k) \to (F', k')$ means that a reduction rule transforms (F, k) into (F', k'). Moreover, by $F_{x=1}$ we mean a CNF formula obtained from F by assigning $x = 1$. More precisely, $F_{x=1}$ is obtained from F by removing all clauses including the literal x and removing the literal $\bar{x}$ from all clauses including the literal $\bar{x}$. A reduction rule is *sound* if for every MAXSAT instance I, it holds that I is a YES-instance if and only if the instance obtained from I by applying this rule is a YES-instance. The first six reduction rules have been studied in the literature.

R-Rule 1 ([Bansal and Raman, 1999])**.**
$(F \wedge x\bar{x}C, k) \to (F, k - 1)$.

R-Rule 2 ([Bansal and Raman, 1999])**.**
$(F \wedge xC \wedge \bar{x}C, k) \to (F \wedge C, k - 1)$.

R-Rule 3 ([Chen and Kanj, 2004])**.** *If there is an (i, j)-literal x in the CNF formula F which contains at least j unit clauses x, then* $(F, k) \to (F_{x=1}, k - i)$.

For instance, if there is a $(4, 1)$-literal x and the five clauses including the variable x are x, xC_1, xC_2, xC_3, and $\bar{x}D$, then we can directly assign x the value 1 due to R-Rule 3. Note that a special case of R-Rule 3 is when $j = 0$, i.e., a variable occurs in all clauses in the same form (positive or negative).

R-Rule 4 ([Bansal and Raman, 1999])**.** *If there is a $(1, 1)$-literal x, then* $(F \wedge xC_1 \wedge \bar{x}C_2, k) \to (F \wedge C_1C_2, k - 1)$.

R-Rule 5 ([Bliznets and Golovnev, 2012])**.** *If there is a 3-variable x and a 2-clause xy, then*
$(F \wedge xy \wedge xC \wedge \bar{x}D, k) \to (F \wedge yD \wedge \bar{y}CD, k - 1)$.

R-Rule 6 ([Li *et al.*, 2007])**.** *If there are clauses xy, $\bar{x}$, and $\bar{y}$, then* $(F \wedge xy \wedge \bar{x} \wedge \bar{y}, k) \to (F \wedge \bar{x}\bar{y}, k - 1)$.

Inspired by R-Rule 6, we propose a new rule as follows.

R-Rule 7. *If there are clauses xyz, $\bar{x}$, $\bar{y}$, and $\bar{z}$, then*
$(F \wedge xyz \wedge \bar{x} \wedge \bar{y} \wedge \bar{z}, k) \to (F \wedge \bar{x}\bar{y}z \wedge \bar{x}\bar{z} \wedge \bar{y}\bar{z}, k)$.

Lemma 1. *R-Rule 7 is sound.*

Proof. Let ρ be an assignment. Let $F_1 = xyz \wedge \bar{x} \wedge \bar{y} \wedge \bar{z}$ and $F_2 = \bar{x}\bar{y}z \wedge \bar{x}\bar{z} \wedge \bar{y}\bar{z}$. If $\rho(x) = \rho(y) = \rho(z) = 1$, only one clause in both F_1 and F_2 is satisfied. If exactly one of $\rho(x)$, $\rho(y)$, and $\rho(z)$ is 0, then exactly two clauses in F_1 and F_2 are satisfied. Otherwise, exactly three clauses in both F_1 and F_2 are satisfied. □

Notice that in R-Rule 6 we decrease k by one, but in R-Rule 7 the value of k is unchanged. Now we introduce numerous new reduction rules to eliminate $(i, 1)$-literals with some specific structures. The worst branching case of the previous best algorithm studied in [Chen and Kanj, 2004] is the $(5, 1)$-branching at $(i, 1)$-literals. The new reduction rules help us circumvent such branchings.

R-Rule 8. *If there are two clauses xyC_1 and $\bar{x}yC_2$ where x is an $(i, 1)$-literal, $i \geq 1$, then replace xyC_1 with xC_1.*

We show that our new rule is sound.

Lemma 2. *R-Rule 8 is sound.*

Proof. Let F_1 be a formula including two clauses xyC_1 and $\bar{x}yC_2$ as stipulated in R-Rule 8. Moreover, let F_2 be the formula obtained from F_1 by replacing xyC_1 with xC_1. Clearly, if an assignment satisfies at least k clauses in F_2, it also satisfies at least k clauses in F_1. For the other direction,

because F_1 and F_2 differ only at xyC_1 and xC_1, if an assignment ρ satisfies k clauses in F_1, it satisfies at least $k-1$ clauses in F_2. Moreover, if it satisfies exactly $k-1$ clauses in F_2, it must be that xC_1 is unsatisfied but yxC_1 is satisfied by ρ, implying that $\rho(x)=0$ and $\rho(y)=1$. Then, by resetting $x=1$ in ρ we obtain an assignment satisfying k clauses in F_2 (and also in F_1). □

R-Rule 9. *If there is an $(i,1)$-literal x such that $i \geq 2$ and all clauses containing the literal x contains a literal y, then remove the literal y from all clauses including the literal x, and add the literal y in the clause including the literal $\bar{x}$, i.e.,*
$(F_1 := F \wedge xyC_1 \wedge xyC_2 \wedge \cdots \wedge xyC_i \wedge \bar{x}D, k) \rightarrow$
$(F_2 := F \wedge xC_1 \wedge xC_2 \wedge \cdots \wedge xC_i \wedge \bar{x}yD, k)$.

Lemma 3. *R-Rule 9 is sound.*

Proof. Assume that there is an assignment ρ_1 satisfying k clauses in F_1. Clearly, ρ_1 satisfies at least $k-i-1$ clauses in F. If ρ_1 satisfies at least $k-i$ clauses in F, then resetting $x=1$ in ρ_1 (if $\rho(x)=1$ in advance we do nothing) makes at least k clauses in F_2 be satisfied; we are done. Otherwise, if ρ_1 satisfies exactly $k-i-1$ clauses in F, then xyC_1, $xyC_2, \ldots, xyC_i$, and $\bar{x}D$ are all satisfied by ρ_1. There can be only 3 cases (1) $\rho_1(D)=1$; (2) $\rho_1(y)=1$; (3) none of (1) and (2) and $\rho_1(C_1)=\rho_1(C_2)=\cdots=\rho_1(C_i)=1$. In the first and second cases, we reset $x=1$ in ρ_1, and in the third case we reset $x=0$ in ρ_1. After the resetting, all the $i+1$ clauses including the variable x in F_2 are satisfied. The proof for the other direction is analogous. □

The following two reduction rules studied in [Chen *et al.*, 2017] are also needed in our algorithm.

R-Rule 10 ([Chen *et al.*, 2017])**.** *If there is an $(i,1)$-literal x such that the clause $\bar{x}yD$ including the literal $\bar{x}$ contains a $(j,1)$-literal y, then resolute on x, i.e.,*
$(F \wedge xC_1 \wedge \cdots \wedge xC_i \wedge \bar{x}yD, k) \rightarrow$
$(F \wedge yC_1D \wedge \cdots \wedge yC_iD, k-1)$.

R-Rule 11 ([Chen *et al.*, 2017])**.** *Let x be a $(2,1)$-literal, and xC_1, xC_2, and $\bar{x}D$ the three clauses including the variable x. If C_1C_2D contains a positive literal and a negative literal of the same variable, then resolute on x, i.e.,*
$(F \wedge xC_1 \wedge xC_2 \wedge \bar{x}D, k) \rightarrow (F \wedge C_1D \wedge C_2D, k-1)$.

We continue to propose new reduction rules to deal with $(i,1)$-literals which are now based on the domination relations between literals and clauses. Let F be a CNF formula. A literal x in F is *dominated* by another literal y (resp. a clause C not necessarily in F) if each clause in F containing the literal x also contains the literal y (resp. some literal in C). In addition, a literal x is *almost dominated* by a clause D if, except at most one, all clauses in F including the literal x includes some literal in D. For instance, if $F = xya \wedge xy\bar{b} \wedge yc \wedge \bar{x}d$, then the literal x is dominated by the literal y and the clause $a\bar{b}$, and is almost dominated by the clause ad. For a clause $A = x_1 \cdots x_i$, let $!A$ be the clause consists of all negations of literals in A, i.e., $!A = \overline{x_1} \cdots \overline{x_i}$.

R-Rule 12. *Let x be a $(2,1)$-literal, and xC_1, xC_2, and $\bar{x}D$ be the three clauses including the variable x. If there is some literal y in C_1C_2D such that the literal $\bar{y}$ is dominated by $!(C_1C_2D)$, then resolute on x, i.e.,*
$(F_1 := F \wedge xC_1 \wedge xC_2 \wedge \bar{x}D, k) \rightarrow$
$(F_2 := F \wedge C_1D \wedge C_2D, k-1)$.

Lemma 4. *R-Rule 12 is sound.*

Proof. Let ρ be an optimal assignment satisfying at least k clauses in F_1. If $D=1$ under ρ, then ρ satisfies at least $k-1$ clauses in F_2 since C_1D and C_2D are satisfied by ρ. Otherwise, if $D=0$ but at least one of C_1 and C_2, say, C_1, is true under ρ, then xC_1 in F_1 and C_1D in F_2 are satisfied. Then, the correctness of R-Rule 4 suggests that we can replace xC_2 and $\bar{x}D$ with C_2D. Finally, if C_1, C_2, and D are all false under ρ, we claim that the assignment obtained from ρ by resetting $y=1$ is another optimal assignment. As a matter of fact, as $\bar{y}$ is dominated by $!(C_1C_2D)$, any clause C' including the literal $\bar{y}$ contains at least one negation of some literal from C_1C_2D which makes $C'=1$ under ρ. So, the claim follows and we come to the previous cases. The proof for the other direction is analogous. □

R-Rule 13. *Let x be a $(2,1)$-literal in the clauses xC_1, xC_2, and $\bar{x}zD$. If the literal $\bar{z}$ is almost dominated by $!(C_1C_2D)$, then resolute on x and decrease k by one.*

R-Rule 14. *For a 1-literal $\bar{x}$ in a clause $\bar{x}zD$, if the literal $\bar{z}$ is almost dominated by $!D$, then resolute on x and decrease k by one.*

R-Rule 15. *Let x be a $(2,1)$-literal in the clauses xC_1, xC_2, and $\bar{x}D$. If there are two literals y_1 and y_2 in C_1C_2 such that the literal $\overline{y_1}$ is dominated by some literal z and the literal $\overline{y_2}$ is dominated by the literal $\bar{z}$, then resolute on x and decrease k by one.*

Similar to the proof of Lemma 4, we can show the soundness of R-Rules 13–15.

Lemma 5. *R-Rules 13–15 are sound.*

Finally, our algorithm needs the following reduction rule studied in [Chen *et al.*, 2017].

R-Rule 16 ([Chen *et al.*, 2017])**.** *Let x be a $(2,2)$-literal in the clauses xC_1, xC_2, $\bar{x}C_3$, $\bar{x}C_4$. If each C_i, $1 \leq i \leq 4$, contains a singleton literal, then resolute on x.*

It should be pointed out that several of the above reductions rules may significantly increase the size of the formula. Nevertheless, thanks to a kernelization algorithm for the MaxSAT problem (see, e.g., [Chen and Kanj, 2004]), this does not diminish the applicability of these rules. In particular, due to this kernelization algorithm, once the size of the formula in an instance gets too large, we can always transform the instance into an equivalent instance whose size is bounded by $O(k^2)$. Then, given that all our reduction rules never increase the value of k, the above reduction rules can be exhaustively applied in polynomial time.

4 Branching Rules

In this section, we study branching rules for instances induced by all reduction rules studied in the previous section. Let the number of clauses in a CNF formula F be m. Each branching splits an instance I into several subinstances

sections	variable types	B-Rules (vectors)	roots
Sec. 4.1	6^+-variable &	B-Rule 1 $(6,1)$	1.2853
	$(i \geq 3, 1)$-nsin	B-Rule 2 $(3,3)$	1.2600
Sec. 4.2	3-variable	B-Rule 3 $(8,1)$	1.2321
Sec. 4.3	$(3,2)$-literal	B-Rules 4, 7 $(4,2)$	1.2721
		B-Rule 5 $(10,3,3)$	1.2937
		B-Rule 6 $(6,1)$	1.2853
		B-Rule 8 $(2,4,12)$	1.2937
Sec. 4.4	$(2,2)$-literal	B-Rule 9 $(6,1)$	1.2853
		B-Rules 10, 11, 13, 14	
		$(10,3,3)$	1.2937
		B-Rule 12 $(15,11,3,3)$	1.2950
		B-Rule 15	
		$(14,14,12,12,7,7,5,5)$	1.2989
Sec. 4.5	$(3,1)$-sin	Lemma 22	1.2576
	$(4,1)$-sin		

Table 2: An overview of branchings in Section 4. Here, "sin" stands for "singleton" and "nsin" for "non-singleton".

which can be recursively solved. We say that the branching is *sound* if it holds that I is a YES-instance if and only if at least one of the subinstances is a YES-instance. If branching on F results in r many CNF formulas which include respectively $m - d_1, m - d_2, \ldots, m - d_r$ clauses, we call such a branching a $(d_1, \ldots, d_r)$-*branching*, and call $(d_1, \ldots, d_r)$ its *branching vector*. It is known that the polynomial $p(x) = x^m - x^{m-d_1} - \cdots - x^{m-d_r}$ has a unique positive root not smaller than 1, which is called the *branching root* of this branching. A branching is *inferior* to another branching if the branching root of the former is greater than that of the latter. If every branching step in a branching algorithm has its root bounded by a constant c, the running time of the algorithm can be bounded by $O^*(c^m)$.

Now, we are ready to present the *branching rules* (B-Rules). We assume that B-Rule j is applied only when none of B-Rules i with $i < j$ is applicable. Let F be a CNF formula reduced by the reduction rules studied in the previous section. By saying branching on a variable x, we mean the branching with $x = 1$ and $x = 0$. Clearly, any branching on a variable is sound. In each branching case, the parameter k decreases by the number of clauses satisfied in this branching. We won't point this out in the following branching rules as we analyze the running time with respect to the number of clauses. This section is divided into several subsections each of which is concerned with branching rules for variables of some specific type. We refer to Table 2 for an overview. Observe that any induced instance by R-Rules 1–3 contains only variables of degree at least three.

4.1 Branching on 6^+-Variables

In this section, we present two branching rules to deal with variables of degree at least 6. A direct branching on a 7^+-variable produces a $(6,1)$-branching with branching root $1.2853 < 1.2989$ in the worst case. However, a direct branching on a 6-variable only guaranties a $(5,1)$-branching with branching root 1.3248. Hence, we need refined branching rules for 6-variables in order to improve the $O^*(1.3248^m)$-algorithm studied by Chen and Kanj [2004].

Lemma 6 (B-Rule 1). *Branching on a 7^+-variable, a $(3,3)$- or $(4,2)$-literal, or on a $(5,1)$-singleton is not inferior to the $(6,1)$-branching.*

Notice that in the branching $x = 1$ for a $(5,1)$-singleton x in B-Rule 1, in addition to the five satisfied clauses which include the literal x, the unite clause $\bar{x}$ is also reduced, leading to the total decrease of m by 6.

Lemma 7 (B-Rule 2). *Let x be an $(i,1)$-literal, $i \geq 3$, such that the clause $\bar{x}D$ including the literal $\bar{x}$ is not unit (i.e., $|D| \geq 1$). Then, the branching with $x = 1$ and $x = D = 0$ is sound and not inferior to the $(i, 1 + 2|D|)$-branching, which is not inferior to the $(3,3)$-branching.*

Proof. To prove the soundness of this rule, it suffices to show that if there is an optimal assignment under which some literal y in D is assigned the value 1, there is another optimal assignment under which $x = 1$. Assume that σ is an optimal assignment such that that $\sigma(y) = 1$ and $\sigma(x) = 0$. Then, due to that $\bar{x}D$ is the only clause containing $\bar{x}$, and $y = 1$ already satisfies this clause, resetting $x = 1$ does not decrease the number of satisfied clauses.

Now we analyze the branching vector. Let $D = y_1 y_2 \ldots y_h$ where $h \geq 1$. Branching with $x = 1$ satisfies at least i clauses. On the other hand, because the instance is reduced by R-Rule 10, each $\overline{y_i}$, $1 \leq i \leq h$, cannot be a 1-literal. Moreover, as R-Rule 14 is not applicable, branching with $x = D = 0$ satisfies at least $2|D|$ clauses. Finally, since R-Rule 1 is not applicable, $x = 0$ satisfies one more clause $\bar{x}y_1 \cdots y_h$. Therefore, the branching is not inferior to the $(i, 1 + 2|D|)$-branching where $i \geq 3$ and $1 + 2|D| \geq 3$. □

4.2 Branching on 3-Variables

A direct branching on a 3-variable results in a $(2,1)$-branching in the worst case. Bliznets and Golovnev [2012] proposed a smart resolution-involved $(6,1)$-branching for 3-variables. We further improve this branching significantly to a $(8,1)$-branching (B-Rule 3). This improved branching is used as sub-branchings in other branching cases and hence plays an important role in improving the whole algorithm. Our improvement relies on a specific assignment for domination-related literals.

Lemma 8 ([Kulikov and Kutzkov, 2007]). *If a literal y dominates another literal x, then the branching with (B1) $x = 0$, and (B2) $x = 1$ and $y = 0$ is sound.*

Proof. From an optimal assignment ρ with $\rho(x) = \rho(y) = 1$, resetting $\rho(x) = 0$ achieves another optimal assignment since every clause including the literal x is satisfied by y which dominates x. This is covered by the branching $x = 0$. □

Lemma 9 (B-Rule 3). *Let x be a 3-variable, and xC_1, xC_2, and $\bar{x}D$ the three clauses including the variable x. Then the following branching is sound and leads to a branching not inferior to the $(1, 3 + |C_1| + |C_2| + 2|D|)$-branching, which is not inferior to the $(8,1)$-branching: (B1) $C_1 = C_2 = D = 0$; (B2) resolute on x and decrease k by one.*

Proof. To see the soundness of the branching rule, observe that it is safe to resolute on x if at least one of C_1, C_2, and D

is true. The case where C_1, C_2, and D are all false is covered in the branch B1.

In the branch B2, the number of clauses decreases by one. In the branch B1, we can set $x = 1$ and hence the three clauses including the variable x are reduced (two of them are satisfied). Moreover, due to R-Rules 3, 8–13, there are at least $|C_1| + |C_2| + 2|D|$ extra clauses, each of which contains at least a negation of some literal in C_1C_2D, which are satisfied under this branching. Therefore, the number of clauses decreases in total by at least $|C_1| + |C_2| + 2|D| + 3$. Now we show that the branching is not inferior to the $(8, 1)$-branching. Since R-Rule 5 is not applicable, it holds that $|C_1|, |C_2| \geq 2$ and hence $|C_1| + |C_2| + 2|D| + 3 \geq 7$. However, we show that when $|C_1| + |C_2| + 2|D| + 3 = 7$, we can further reduce at least one more clause. First, the equality holds only when $D = \emptyset$, $|C_1| = |C_2| = 2$ and, moreover, every literal in C_1C_2 is a $(j, 1)$-literal for some $j \geq 1$. Let $C_1 = y_1y_2$ and $C_2 = y_3y_4$ where every $\overline{y_i}$, $1 \leq i \leq 4$, is a 1-literal. As the instance is reduced by R-Rules 6–7, at least one of the literals $\overline{y_1}$ and $\overline{y_2}$ is contained in a 2^+-clause, say $\overline{y_1}zD'$. As R-Rule 3 is not applicable, there exists at least one clause including the literal $\bar{z}$. As $D = \emptyset$, the literal z cannot be any literal in $!(C_1C_2)$ due to R-Rule 12; it cannot be any literal in C_1C_2 due to R-Rule 10. As $\overline{y_1}$ is a 1-literal, it holds that z dominates $\overline{y_1}$. Then, due to Lemma 8, we can set $z = 0$. Due to R-Rule 15, $\bar{z}$ cannot be in any clause containing some literal in $\overline{y_2}\overline{y_3}\overline{y_4}$. This implies that after setting $z = 0$ at least one more clause is reduced. □

4.3 Branching on $(3, 2)$-Literals

Now we are left with variables of degrees 4 and 5. This section is devoted to branching rules for $(3, 2)$-literals.

Lemma 10 (B-Rule 4). *For a $(3, 2)$-literal x in a reduced formula F, if there is a unit clause x or $\bar{x}$, then branching on x is not inferior to the $(4, 2)$-branching.*

After B-Rule 4, there is no unit clause containing a $(3, 2)$- or $(2, 3)$-literal.

Lemma 11 (B-Rule 5). *Let $\bar{x}C$ be a clause where x is a $(3, 2)$-literal and C contains a literal occurring at most thrice in the clauses without the literal $\bar{x}$. Then, branching on x yields a branching not inferior to the $(10, 3, 3)$-branching.*

Lemma 12 (B-Rule 6). *Let x be a $(3, 2)$-literal. If the two clauses with the literal $\bar{x}$ contain a $(4, 1)$-singleton y_1 and a 5-variable y_2, respectively, then branching on y_2 leads to a branching not inferior to the $(6, 1)$-branching.*

The following branching rule is for $(2, 2)$-literals but is used as a sub-branching in the next branching rule dealing with $(3, 2)$-literals.

Lemma 13 (B-Rule 7). *Let x be a $(2, 2)$-literal such that there is a unit clause $\bar{x}$. Let xC_1 and xC_2 be the two clauses including the literal x. Then the following branching is sound and leads to a branching not inferior to the $(4, 2)$-branching: (B1) $x = 1$ and $C_1 = C_2 = 0$; (B2) $x = 0$.*

Lemma 14 (B-Rule 8). *Let $\bar{x}y_1D_1$ and $\bar{x}y_2D_2$ be two clauses where x is a $(3, 2)$-literal, and each of y_1 and y_2 is either a $(3, 2)$- or a $(2, 3)$-literal. Then, branching on y_1 leads to a branching not inferior to the $(2, 4, 12)$-branching.*

Proof. W.l.o.g. we assume that $|D_1| \leq |D_2|$. We distinguish between two cases: (1) $|D_2| \geq 1$ and (2) $|D_2| = |D_1| = 0$.

In the first case, in the branching $y_1 = 0$, at least two clauses including the literal $\overline{y_1}$ are reduced. In the branching $y_1 = 1$, all clauses including the literal y_1 is reduced, and x becomes an $(i, 1)$-non-singleton for some $i \in \{1, 2, 3\}$. If $i = 1$, y_1 is a $(3, 2)$-literal. Moreover, R-Rule 2 applies, leading to one more clause being reduced. In this case, we have a $(2, 4)$-branching. If $i = 2$, then B-Rule 3 is invoked (in the worst case), leading to at least a $(1, 9)$-subbranching (recall that $|D_2| \geq 1$ and due to B-Rule 4 any clause including the literal x is of size at least two). In this case, we have a $(2, 4, 12)$-branching if y_1 is a $(3, 2)$-literal and a $(3, 3, 11)$-branching if y_1 is a $(2, 3)$-literal. Finally, if $i = 3$, B-Rule 2 applies, generating at least a $(3, 5)$-subbranching. In this case, we have a $(2, 6, 8)$-branching if y_1 is a $(3, 2)$-literal and a $(3, 5, 7)$-branching if y_1 is a $(2, 3)$-branching. All the above branchings are not inferior to the $(2, 4, 12)$-branching.

Now we consider the second case. If $y_1 = 0$, $\bar{x}y_1$ becomes a unit clause and $\bar{x}$ becomes a $(2, 0)$-, $(2, 1)$-, $(2, 2)$-, or $(2, 3)$-literal. In the first and second cases, R-Rule 3 is applicable; in the third case, B-Rule 7 is applicable; and in the last case, B-Rule 4 is applicable. All of these cases lead to at least a $(4, 2)$-subbranching. In the branching $y_1 = 1$, x becomes a $(i, 1)$-non-singleton where $i \in \{1, 2, 3\}$. If $i = 1$, R-Rule 3 applies. If $i = 2$, either some reduction rules apply or B-Rule 3 applies and a $(1, 9)$-subbranching is generated. Finally, if $i = 3$, then either some reduction rules apply or B-Rule 2 applies and a $(3, 5)$-subbranchings is generated. In the worst case, we have a $(4, 6, 4, 12)$-branching.

Since the $(4, 6, 4, 12)$-branching is not inferior to the $(2, 4, 12)$-branching, the lemma follows. □

4.4 Branching on $(2, 2)$-Literals: the Worst Case

Now, assume that only $(4, 1)$- and $(3, 1)$-singletons, and $(2, 2)$-literals exist in the reduced formula. We present numerous branching rules to eliminate $(2, 2)$-literals.

Lemma 15 (B-Rule 9). *Let x be a $(2, 2)$-literal. If there is a clause xy_1C such that y_1 is a $(4, 1)$-singleton, and the other clause containing the literal x contains another singleton, then branching on y_1 leads to a branching not inferior to the $(6, 1)$-branching.*

Proof. In the branching $y_1 = 0$, the clause including $\overline{y_1}$ is satisfied. In the branching $y_1 = 1$, four clauses including the literal y_1 are satisfied and the unsatisfied unit clause $\overline{y_1}$ is reduced. Moreover, when $y_1 = 1$, x becomes a 1- or 0-literal, invoking the application of at least one of R-Rules 2, 3, and 10, further decreasing the number of clauses by one. Therefore, the number of clauses decreases by 6 in total. □

Lemma 16 (B-Rule 10). *Let x be a $(2, 2)$-literal. If the two clauses including the literal x contain a singleton y_1 and a $(2, 2)$-literal y_2, respectively, then, branching on y_2 leads to a branching not inferior to the $(10, 3, 3)$-branching.*

Observe that after the above branching rules, there are no clauses including both a $(2, 2)$-literal and a $(4, 1)$-singleton.

In other words, any clause including a $(2,2)$-literal can contain only other $(2,2)$-literals or $(3,1)$-singletons. Moreover, due to R-Rule 16 and B-Rule 7, for each $(2,2)$-literal x in clauses xC_1, xC_2, $\bar{x}D_1$, $\bar{x}D_2$, it holds that (1) $|C_1|, |C_2|, |D_1|, |D_2| \geq 1$, and (2) at least one of C_1, C_2, D_1, and D_2 contains only $(2,2)$-literals. Therefore, all $(2,2)$-literals x as denoted above fall into one of the following categories: either $C_1C_2D_1D_2$ contains only $(2,2)$-literals, or C_1C_2 (resp. D_1D_2) contains only $(2,2)$-literals and D_1D_2 (resp. C_1C_2) contains only $(3,1)$-singletons. The next two branching rules are to eliminate literals of the second type.

Lemma 17 (B-Rule 11). *Let xC be a clause such that x is a $(2,2)$-literal, $1 \leq |C| \leq 3$, and C contains only singletons. Then, branching on x leads to a branching not inferior to the $(10,3,3)$-branching.*

After B-Rule 11, if a clause xC contains a $(2,2)$-literal x and C contains only singletons, it must be that $|C| \geq 4$.

Lemma 18 (B-Rule 12). *Let x be a $(2,2)$-literal, and xC_1, xC_2, $\bar{x}yD_1$, $\bar{x}D_2$ the clauses including the variable x. If C_1C_2 contains only singletons, then branching on y leads to a branching not inferior to the $(15,11,3,3)$-branching.*

Now, all $(2,2)$-literals are of the first type, i.e., every clause including a $(2,2)$-literal contains only $(2,2)$-literals.

Lemma 19 (B-Rule 13). *Let x be a $(2,2)$-literal, and xy_1C_1 and xy_2C_2 the two clauses including the literal x. If y_1 and y_2 are the literals of the same variable (i.e., $y_1 = y_2$ or $y_1 = \overline{y_2}$), or there is another clause $y_1\overline{y_2}C_3$, then branching on x leads to a branching not inferior to the $(10,3,3)$-branching.*

Lemma 20 (B-Rule 14). *Let xy and $\bar{x}zC_1$ be two clauses such that x is a $(2,2)$-literal. Then, branching on z leads to a branching not inferior to the $(10,3,3)$-branching.*

The last branching rule can be used to eliminate all the remaining $(2,2)$-literals.

Lemma 21 (B-Rule 15). *For clauses xy_1C_1, $\bar{x}y_2C_2$, y_1D_1, and y_2D_2 where x is a $(2,2)$-literal, the following branching is sound and leads to a branching not inferior to the $(14,14,12,12,7,7,5,5)$-branching:*

B1. $x = 1$ *and* $y_1 = 0$*;*

B2. $x = y_1 = 1$ *and* $D_1 = 0$*;*

B3. $x = 0$ *and* $y_2 = 0$*;*

B4. $x = D_2 = 0$ *and* $y_2 = 1$.

4.5 Dealing with Singletons and Our Main Result

If none of the reduction rules and branching rules is applicable, then we are left with only $(3,1)$- and $(4,1)$-singletons. A branching algorithm handling this special case has been studied in the literature, as shown in the following lemma.

Lemma 22 ([Bliznets and Golovnev, 2012])**.** *The* MAXSAT *problem on formulas with only $(3,1)$- and $(4,1)$-singletons can be solved in time $O^*(1.2576^m)$.*

Now we are able to show our main result, i.e., an exact algorithm of running time $O^*(1.2989^m)$ for the MAXSAT problem. The algorithm applies the branching rules in the order in accordance with their occurrences, and importantly, before each branching we exhaustively apply all reduction rules presented in the previous section. As pointed out in the end of Section 3, it takes polynomial time to exhaustively apply all reduction rules. If none of the branching rules applies, we are left with $(3,1)$- and $(4,1)$-singletons, and in this case we solve the instance in $O^*(1.2576^m)$ time (Lemma 22). The running time of the whole algorithm is dominated by the branching described in Lemma 21, which has the worst branching root 1.2989 (see Table 2 for a reference).

Theorem 1. *The* MAXSAT *problem can be solved in time $O^*(1.2989^m)$.*

5 Concluding Remarks

In this paper, we have proposed an exact branch-and-bound algorithm for the MAXSAT problem whose running time is bounded by $O^*(1.2989^m)$, where m is the number of clauses. Our algorithm considerably improves the previous best result $O^*(1.3248^m)$ published 15 years ago [Chen and Kanj, 2004]. To this end, we put forward many new reduction rules to shrink the size of the given CNF formula in polynomial time. In addition, based on these reduction rules and refined observations, we derived improved branching strategies for variables of degrees 3–5.

It is important to point out that our analysis is purely worst-case based. The performance of our reduction and branching rules in solving practical instances deserves to be further investigated. In fact, the cores of many competitive exact MAXSAT solvers are branch-and-bound procedures (see, e.g., [Davis *et al.*, 1962; Argelich *et al.*, 2018; Abramé and Habet, 2015; Lin *et al.*, 2008]). Some research even reveals that in some cases branch-and-bound algorithms are the best option [Morgado *et al.*, 2013]. In addition, it makes much sense to study how to efficiently integrate our reduction and branching rules into heuristic algorithms. To employ our branching rules, one may need to resort to powerful pruning approaches, such as approaches based on the phase transition [Coppersmith *et al.*, 2004], or using machine learning algorithms to determine the pruning conditions, which have proved to be useful in solving a variety of problems (see, e.g., [He *et al.*, 2014]). We refer to [Morgado *et al.*, 2013] and references therein for discussions on many other nice ideas of designing (exact or heuristic) branch-and-bound based MaxSAT algorithms.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Nos. 61420106009, 61872097, 61672536, 61802441, 61872048, 61872450, 61828205, 61702557), 111 Project (No. B18059), Hunan Provincial Science and Technology Program (No. 2018wk4001), and the China Postdoctoral Science Foundation (No. 2017M612584).

References

[Abramé and Habet, 2015] André Abramé and Djamal Habet. On the resiliency of unit propagation to Max-resolution. In *IJCAI*, pages 268–274, 2015.

[Argelich *et al.*, 2018] Josep Argelich, Chu Min Li, Felip Manyà, and Joan Ramon Soler. Clause branching in MaxSAT and MinSAT. In *CCIA*, pages 17–26, 2018.

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

[Bliznets and Golovnev, 2012] Ivan Bliznets and Alexander Golovnev. A new algorithm for parameterized MAX-SAT. In *IPEC*, pages 37–48, 2012.

[Bliznets, 2013] Ivan Bliznets. A new upper bound for $(n, 3)$-MAX-SAT. *J. Math. Sci.*, 188(1):1–6, 2013.

[Cai *et al.*, 2014] Shaowei Cai, Chuan Luo, and Kaile Su. Scoring functions based on second level score for k-SAT with long clauses. *J. Artif. Intell. Res.*, 51:413–441, 2014.

[Chen and Kanj, 2004] Jianer Chen and Iyad Kanj. Improved exact algorithms for MAX-SAT. *Discrete Appl. Math.*, 142(1–3):17–27, 2004.

[Chen *et al.*, 2010] Jianer Chen, Iyad Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40–42):3736–3756, 2010.

[Chen *et al.*, 2017] Jianer Chen, Chao Xu, and Jianxin Wang. Dealing with 4-variables by resolution: An improved MaxSAT algorithm. *Theor. Comput. Sci.*, 670:33–44, 2017.

[Coppersmith *et al.*, 2004] Don Coppersmith, David Gamarnik, MohammadTaghi Hajiaghayi, and Gregory Sorkin. Random MAX SAT, random MAX CUT, and their phase transitions. *Random Struct. Algorithms*, 24(4):502–545, 2004.

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

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

[Garey *et al.*, 1976] Michael Garey, David Johnson, and Larry Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[Goemans and Williamson, 1994] Michel Goemans and David Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM J. Discrete Math.*, 7(4):656–666, 1994.

[Gupta *et al.*, 2018] Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. When rigging a tournament, let greediness blind you. In *IJCAI*, pages 275–281, 2018.

[He *et al.*, 2014] He He, Hal Daumé III, and Jason Eisner. Learning to search in branch and bound algorithms. In *NIPS*, pages 3293–3301, 2014.

[Hutter *et al.*, 2017] Frank Hutter, Marius Lindauer, Adrian Balint, Sam Bayless, Holger Hoos, and Kevin Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artif. Intell.*, 243:1–25, 2017.

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

[Kulikov and Kutzkov, 2007] Alexander Kulikov and Konstantin Kutzkov. New bounds for MAX-SAT by clause learning. In *CSR*, pages 194–204, 2007.

[Li *et al.*, 2007] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *J. Artif. Intell. Res.*, 30:321–359, 2007.

[Lin *et al.*, 2008] Han Lin, Kaile Su, and Chu Min Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *AAAI*, pages 351–356, 2008.

[Luo *et al.*, 2017] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artif. Intell.*, 243:26–44, 2017.

[Marques-Silva, 1999] João Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *EPIA*, pages 62–74, 1999.

[Morgado *et al.*, 2013] Antonio Morgado, Federico Heras, Mark Liffiton, Jordi Planes, and Joao Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[Niedermeier and Rossmanith, 1999] Rolf Niedermeier and Peter Rossmanith. New upper bounds for MaxSat. In *ICALP*, pages 575–584, 1999.

[Poloczek *et al.*, 2017] Matthias Poloczek, Georg Schnitger, David Williamson, and Anke van Zuylen. Greedy algorithms for the maximum satisfiability problem: Simple algorithms and inapproximability bounds. *SIAM J. Comput.*, 46(3):1029–1061, 2017.

[Raman *et al.*, 1998] Venkatesh Raman, Bala Ravikumar, and Srinivasa Rao Satti. A simplified NP-complete MAXSAT problem. *Inform. Process. Lett.*, 65(1):1–6, 1998.

[Schaefer, 1978] Thomas Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[Sohanghpurwala *et al.*, 2017] Ali Asgar Sohanghpurwala, Mohamed Hassan, and Peter Athanas. Hardware accelerated SAT solvers — A survey. *J. Parallel Distrib. Comput.*, 106:170–184, 2017.

[Sturtevant, 2003] Nathan Sturtevant. Last-branch and speculative pruning algorithms for Max$^{\mathrm{n}}$. In *IJCAI*, pages 669–675, 2003.

[Wang *et al.*, 2018] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *IJCAI*, pages 1514–1522, 2018.