The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Improved Algorithms for Maximum Satisfiability and Its Special Cases

Kirill Brilliantov[1], **Vasily Alferov**[2], **Ivan Bliznets**[3]

[1] Constructor University
[2] Independent Researcher
[3] Utrecht University
ki.br178@gmail.com, vasily.v.alferov@gmail.com, iabliznets@gmail.com/i.bliznets@uu.nl

Abstract

The Maximum Satisfiability (MAXSAT) problem is an optimization version of the Satisfiability problem (SAT) in which one is given a CNF formula with n variables and needs to find the maximum number of simultaneously satisfiable clauses. Recent works achieved significant progress in proving new upper bounds on the worst-case computational complexity of MAXSAT. All these works reduce general MAXSAT to a special case of MAXSAT where each variable appears a small number of times. So, it is important to design fast algorithms for (n, k)-MAXSAT to construct an efficient exact algorithm for MAXSAT. (n, k)-MAXSAT is a special case of MAXSAT where each variable appears at most k times in the input formula.

For the $(n, 3)$-MAXSAT problem, we design a $O^*(1.1749^n)$ algorithm improving on the previous record running time of $O^*(1.191^n)$. For the $(n, 4)$-MAXSAT problem, we construct a $O^*(1.3803^n)$ algorithm improving on the previous best running time of $O^*(1.4254^n)$. Using the results, we develop a $O^*(1.0911^L)$ algorithm for the MAXSAT where L is a length of the input formula which improves previous algorithm with $O^*(1.0927^L)$ running time.

Introduction

The Satisfiability problem (SAT) is one of the most influential problems with applications in Computer Science, Artificial Intelligence, and other fields. Its optimization version, Maximum Satisfiability problem (MAXSAT), also significantly impacts many areas. It was successfully used in planning, scheduling, configuration problems, AI and data analysis problems, combinatorial problems, verification and security, and bioinformatics (Bacchus, Järvisalo, and Martins 2021). For a detailed survey about SAT and MAXSAT problems, we refer to (Biere, Heule, and van Maaren 2021).

It is well-known that SAT and MAXSAT are NP-hard along with their many special cases (Garey and Johnson 1979). Many approaches were tested to cope with the NP-hardness of SAT and MAXSAT: randomized, approximation, exact, parameterized algorithms, and others. In recent years significant progress was achieved from the exact exponential algorithms point of view for both problems. New upper bounds were proved on the computational complexity in the worst-case scenario for SAT and MAXSAT. Moreover, the new upper bounds significantly improve previously known results, some of which stayed without progress for decades. For example, Chu et al. (Chu, Xiao, and Zhang 2021) presented an $O^*(1.234^m)$ algorithm, Peng and Xiao (Peng and Xiao 2021) constructed a $O^*(1.0646^L)$ for the Satisfiability problem where m is the total numbers of clauses and L is the total number of literals in the input formula. Similar results were obtained for Maximum Satisfiability in papers (Xu et al. 2019b; Xiao 2022; Alferov and Bliznets 2021) proving $O^*(1.2886^m)$ and $O^*(1.0927^L)$ upper bounds.

All three last-mentioned papers used the branch-and-bound method with sophisticated reduction and branching rules, including detailed reasoning. Roughly speaking, in such algorithms input instance is usually transformed into an equivalent instance where each variable appears at most five times. After that, we reduce to an instance where each variable occurs at most four times, and finally, an instance where each variable appears at most three times is solved. So, in the worst-case scenario, the algorithm's efficiency for MAXSAT highly depends on how quickly we can solve (n, k)-MAXSAT for $k \in \{3, 4, 5\}$. Recall that (n, k)-MAXSAT is a special case of MAXSAT in which the input CNF contains only variables that appear at most k times. It is known that (n, k)-MAXSAT is NP-hard for $k \geq 3$ (Raman, Ravikumar, and Rao 1998). We note that on its own $(n, 3)$-MAXSAT attracts much attention (see table 1). In the paper we sufficiently improve the running time for $(n, 3)$-MAXSAT and $(n, 4)$-MAXSAT getting new upper bounds $O^*(1.1749^n)$ and $O^*(1.3803^n)$ correspondingly. These improvements allow us to design a $O^*(1.0911^L)$ algorithm for the MAXSAT problem where L is the total number of literals in the input formula (or simply the length of the formula). Historical progress for MAXSAT in terms of L is shown in table 2.

Except for measures n, L, there are natural measures/parameters m (the total number of clauses in the input formula) and k (the number of clauses that must be satisfied). Generally, such measures take a lot of work to compare. However, we manage to show that our algorithms imply $O^*(1.1554^k)$ algorithm for $(n, 3)$-MAXSAT and $O^*(1.2989^k)$ algorithm for $(n, 4)$-MAXSAT. These obtained upper bounds are significantly better than one might

Running time	Reference
$O^*(1.732^n)$	(Raman, Ravikumar, and Rao 1998)
$O^*(1.3248^n)$	(Bansal and Raman 1999)
$O^*(1.27203^n)$	(Kulikov and Kutskov 2009)
$O^*(1.2600^n)$	(Bliznets 2013)
$O^*(1.237^n)$	(Xu, Chen, and Wang 2016)
$O^*(1.194^n)$	(Xu et al. 2019a)
$O^*(1.191^n)$	(Belova and Bliznets 2020)
$O^*(1.1749^n)$	**this paper**

Table 1: Known results for $(n, 3)$-MAXSAT. Here, n denotes the number of variables in the input formula.

Running time	References
$O^*(1.1279^L)$	(Niedermeier and Rossmanith 1999)
$O^*(1.1057^L)$	(Bansal and Raman 1999)
$O^*(1.1049^L)$	(Alferov and Bliznets 2021)
$O^*(1.0927^L)$	(Alferov and Bliznets 2021)
$O^*(1.0911^L)$	**this paper**

Table 2: Progress for MAXSAT in terms of L

derive from paper (Chen, Xu, and Wang 2017) that presents the best upper bound for the general case of MAXSAT. We note that $O^*(c^k)$ algorithm for MAXSAT immediately implies $O^*(c^m)$ algorithm since $k \leq m$. So, our algorithms produce competitive upper bounds for MAXSAT in terms of m since the best algorithm was $O^*(1.2989^m)$ in (Li et al. 2022). Moreover, the bottleneck for this algorithm was dealing with 4-variables. We warn that algorithm from (Li et al. 2022) gives a slightly better upper bound for $(n, 4)$-MAXSAT than ours in the 5-th digit after the dot and very recently were published an improvement by (Xiao 2022) with the running time $O^*(1.2886^m)$.

Preliminary

The paper assumes familiarity with concepts such as boolean variables, literals, and clauses. Interested readers may find details and further information in (Marek 2009).

By length of a formula or a clause, we mean the total number of literals used to write the formula or clause. An *assignment* is a function that assigns values 0 or 1 to each variable in the formula. An assignment satisfies a clause if some of its literals have value 1. An assignment is optimal if it satisfies the maximum number of clauses.

A variable x is an (i, j)-variable in the formula F if literal x and literal $\overline{x}$ appear i times and j times in F, respectively. A variable is called an h-variable if it is an (i, j)-variable such that $i + j = h$. In this case, we refer to h as a degree variable. A variable of degree at least h is called h^+-variable, and a variable of degree at most h is called h^--variable. Since replacing literals $x, \overline{x}$ with $\overline{x}, x$ respectively in the entire formula does not change the answer for the problem instance, we always assume $i \geq j$ for (i, j)-variables. A clause is called a unit-clause if it contains only one literal. If x is (i, j)-variable, $i = j$, and variable x appears in a unit clause, then we rename it so that there is a unit clause $\overline{x}$.

A subset of the initial formula's clauses is a sub-formula. If a variable with literals inside the sub-formula does not have any literals outside the sub-formula, the sub-formula is said to be closed.

Variable x appears in clause C if either literal x or literal $\overline{x}$ appears in C. Variables x and y *co-appear* in clause C if variables x and y appear in C. x and y are *neighbours* if there is a clause in F, where they co-appear. Similar notions we define for literals.

Our algorithm employs the standard branch-and-bound technique, with the time analysis augmented by the measure-and-conquer approach. Like other algorithms that use this technique, our algorithm consists of reduction and branching rules. A reduction rule, or R-Rule for short, is a polynomial-time algorithm that converts one MAXSAT instance into an equivalent instance with the same or a lower measure value. A polynomial-time algorithm known as a branching rule (B-Rule) divides a MAXSAT instance into several instances with lower measure values, on which the algorithm is then run recursively. If a branching rule transforms in polynomial time an instance with measure L to several instances with measures $L-a_1, L-a_2, \dots, L-a_k$ we call $(a_1, a_2, \dots, a_k)$ – the branching vector of this rule. The only positive root of the polynomial $x^L = x^{L-a_1} + x^{L-a_2} + \dots + x^{L-a_k}$ is called the branching number of the corresponding rule. If an algorithm uses only branching rules with branching numbers $c_1, c_2, \dots, c_t$, then the algorithm's running time is bounded by $(\max_i^t c_i)^L poly(L)$. More details about branch and bound technique can be found in book (Fomin and Kratsch 2010). We exhaustively apply R-rules and B-rules in the order they appear in the paper. Moreover, we repeat the process from the beginning after the rule application, as some previous rules might become applicable after branching. In most cases, our branching rules have two branches, and we can set a variable x in one branch $x = 0$ and $x = 1$ in the second branch. In this case, we denote a decrease of measure by Δ_i in branch $x = i$.

Branching on a variable x is a B-Rule that transforms a formula F into two formulas $F_{x=0}$ and $F_{x=1}$ (same formula assuming $x = 0$ and $x = 1$, respectively). Clearly, this rule is always correct.

We examine the complexity of our method using the so-called discounted length, which was first presented in the (Alferov and Bliznets 2021) article, rather than measuring it in terms of L. The discounted length of a formula F equals $L - n_3 = 2n_3 + 4n_4 + 5n_5 + \dots$, where L is the length of a formula F, and n_i is the number of i-variables. The measure is crucial for our proofs; without the measure, we do not know how to prove claimed results. Since $d \leq L$, if we achieve the $O(c^d)$ algorithm, we also obtain the $O(c^L)$ algorithm.

Proofs of R-Rules, B-Rules, lemmas, and theorems marked by (*) are deferred to the full version of the paper due to space constraints.

Reduction Rules

First of all, we list some reduction rules. Note that all these rules do not increase the discounted length d of the formula.

We write $(F,k) \to (F',k')$ if reduction rule transforms formula F into F' in polynomial time and it is possible to satisfy k clauses in F if and only if it is possible to satisfy k' clauses in F'. Most of the presented reduction rules are widely known and are given without proof.

R-Rule 1. *Let x be a (i,j)-variable such that its positive literal x appears $t \geqslant j$ times in unit clauses. Then we can set $x = 1$.*

R-Rule 2. *If $F = (x \vee \overline{x} \vee C) \wedge F'$, then $(F,k) \to (F', k-1)$.*

R-Rule 3. *If $F = (x \vee C) \wedge (\overline{x} \vee C) \wedge F'$, then $(F,k) \to (C \wedge F', k-1)$.*

R-Rule 4 ((Bansal and Raman 1999)). *Let x be a $(1,1)$-variable in formula $F = (x \vee C) \wedge (\overline{x} \vee D) \wedge F'$, then we can replace clauses $(x \vee C)$ and $(\overline{x} \vee D)$ with clause $(C \vee D)$, i.e. $(F,k) \to ((C \vee D) \wedge F', k-1)$.*

R-Rule 5 ((Xu et al. 2019b)). *Let x be a $(i,1)$-variable and $i \geqslant 2$ and $F = (x \vee l \vee C_1) \wedge ... \wedge (x \vee l \vee C_i) \wedge (\overline{x} \vee D) \wedge F'$, then we can remove l from all clauses containing x and add it to the clause containing $\overline{x}$, i.e. $(F,k) \to ((x \vee C_1) \wedge ... \wedge (x \vee C_i) \wedge (\overline{x} \vee l \vee D) \wedge F', k)$.*

R-Rule 6 (*). *Let x be a $(i,1)$-variable and $1 \leqslant j \leqslant i$ and $F = (x \vee l \vee C_1) \wedge ... \wedge (x \vee l \vee C_j) \wedge (x \vee C_{j+1}) \wedge ... \wedge (x \vee C_i) \wedge (\overline{x} \vee l \vee D) \wedge F'$ and $G = (x \vee C_1) \wedge ... \wedge (x \vee C_i) \wedge (\overline{x} \vee l \vee D) \wedge F'$.*

In this case, we can remove literal l from clauses with literal x. i.e. $(F,k) \to (G,k)$

R-Rule 7. *If formula $F = F_1 \wedge F_2$ and F_1, F_2 are closed sub-formulas, then solve F_1 and F_2 independently and combine results.*

R-Rule 8. *If there are at most 10 distinct variables in F, solve F in constant time.*

R-Rule 9 ((Xu, Chen, and Wang 2016)). *There is a polynomial-time algorithm that takes an instance (F,k) of the MAXSAT problem as an input and outputs a new instance (F',k') satisfying the following conditions: (i) $k' \leqslant k$; (ii) (F,k) is a Yes-instance if and only if (F',k') is a Yes-instance; (iii) there are no two 3-variables in F' that co-appear twice or more.*

Proof. In the original paper, authors formulate this statement for $(n,3)$-MAXSAT. However, their proof can be repeated for the general case of MAXSAT without changes. Indeed, they never use the fact that they have an instance of $(n,3)$-MAXSAT instead of the general MAXSAT. □

Additionally, let us introduce a few auxiliary lemmas.

Lemma 1 ((Gaspers and Sorkin 2012)). *Maximum Satisfiability problem restricted to formulas with clauses of length at most 2, MAX-2-SAT, can be solved in $2^{\frac{m}{6.321}}$ time where m is the number of clauses in the input formula.*

Corollary 1 (*). *MAXSAT on instances where each variable appears more than three times and each clause has a length exactly two can be solved in $2^{\frac{d}{12.642}}$ which essentially corresponds to $(12.642, 12.642)$-branching vector.*

Lemma 2 (*). *If there is a (p,q)-variable z, and $(i,1)$-variable y and formula has the following type: $F = (z \vee C_1) \wedge ... \wedge (z \vee C_p) \wedge (\overline{z} \vee D_1) \wedge ... \wedge (\overline{z} \vee D_q) \wedge F'$, then after application of R-Rules 5 and 6 y can appear at most $i-1$ times in $C_1, ..., C_p$. Similar result hold for $D_1, ..., D_q$.*

Rules for 6^+-Variables

This section aims to reduce general MAXSAT to $(n,5)$-MAXSAT. We achieve this result by branching in variables with a high degree. We start with the following important lemma.

Lemma 3 (Lemma 2 in (Alferov and Bliznets 2021)). *Let x be an (i,j)-variable $(i+j \geqslant 4)$ in the formula $F = (x \vee C_1) \wedge ... \wedge (x \vee C_i) \wedge (\overline{x} \vee D_1) \wedge ... \wedge (\overline{x} \vee D_j) \wedge F'$. The branching on x gives at least a $(i + j + \sum_{k=1}^{i} |C_k|, i + j + \sum_{k=1}^{j} |D_k|)$ branching vector in measure d.*

B-Rule 1 (*). *If x is a (i,j)-variable such that $i+j \geq 6$ and $(i,j) \neq (4,2)$, then branching on x gives at least $(9,7)$-branching.*

From now on, if we have a 6^+-variable in the formula, it must be a $(4,2)$-variable. So either our formula already does not contain 6^+-variables or has the following type:

$F = (x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (x \vee C_4) \wedge (\overline{x} \vee D_1) \wedge (\overline{x} \vee D_2) \wedge F'$.

B-Rule 2 (*). *Let F has the type described above, and at least one of the conditions below is met:*

- *$|D_1| + |D_2| > 0$;*
- *$|C_1| + |C_2| + |C_3| + |C_4| > 4$;*
- *there is a 3- or 4-variable in C_1, C_2, C_3, C_4;*
- *there is a 5-variable that appears in C_1, C_2, C_3, C_4 more than once.*

Branching on x gives at least $(11,6)$-branching.

Now we can assume that D_1, D_2 are empty and $|C_1| = |C_2| = |C_3| = |C_4| = 1$.

B-Rule 3. *If x, y are $(4,2)$-variables such that $F = (x \vee l_y) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (x \vee C_4) \wedge (\overline{x}) \wedge (\overline{x}) \wedge F'$, where l_y is a literal of y, then branching on y gives at least $(12,6)$ branching.*

Proof. Branching on y gives us at least $(10,6)$ by similar reasons as in the proof of B-Rule 1. However, in $l_y = 0$ branch after elimination $\overline{l}_y$ and its neighbors x still will be at least 3-variable and R-Rule 3 would be applicable. So in this branch measure additionally drops at least by 2. The measure's decrease will increase by at least two. Hence, depending on l_y the branching vector is at least $(10+2, 6) = (12,6)$ or $(10, 6+2) = (10,8)$. □

It is left to consider only the situation described below.

B-Rule 4. *Let x be a $(4,2)$-variable and formula has the following type: $F = (x \vee l_1) \wedge (x \vee l_2) \wedge (x \vee l_3) \wedge (x \vee l_4) \wedge (\overline{x}) \wedge (\overline{x}) \wedge F'$ where l_1, l_2, l_3, l_4 are literals of four distinct 5^+-variables. Consider the following branching:*

- $x = 1$;
- $x = 0, l_1 = 1$;
- $x = 0, l_1 = 0, l_2 = l_3 = l_4 = 1$.

It gives at least $(10, 11, 26)$*-branching.*

Proof. First of all, we prove the correctness of this rule. Let σ be an optimum assignment such that at most two literals from l_1, l_2, l_3, l_4 are set to true and $\sigma(x) = 0$. Then, if we flip the value of x, the new assignment σ' will satisfy at least the same number of clauses as σ. So, it is enough considering only assignments such that either x equals 1, or x equals 0, and at least three literals among l_1, l_2, l_3, l_4 are 1.

In the first branch, we eliminate variable x and literals l_1, l_2, l_3, l_4. Variable x and variable of literal l_1 are eliminated in the second branch. Since l_1 is a literal of a 5^+-variable, the measure drops at least by 11. Finally, in the third branch, variables of x, l_1, l_2, l_3, l_4 are eliminated. So the measure decreases at least by $6 + 5 \cdot 4 = 26$. □

Presented branching rules exhaust all possibilities of how 6^+-variable can appear in the input formula. Essentially we reduce MAXSAT problem to $(n, 5)$-MAXSAT. The worst branching is $(9, 7)$, so if we manage to solve $(n, 5)$-MAXSAT in $O^*(1.0911^d)$ time, we also solve MAXSAT within this running time. Hence, also in $O^*(1.0911^L)$.

Rules for 5-Variables

In this section, we present reduction and branching rules that allow us to handle 5-variables and produce at least $(7, 9)$-branching with respect to measure d.

Lemma 4 (* (Alferov and Bliznets 2021))**.** *It is enough to consider only input formulas F of the following type:*

$(x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (\overline{x} \vee D_1) \wedge (\overline{x} \vee D_2) \wedge F'$

where all 5*-variables are* $(3, 2)$*-variables,* C_1, C_2, C_3 *are not empty,* $|D_1| \geq |D_2|$ *and* D_1 *is not empty.*

Below we consider two separate cases depending on whether D_2 is empty.

D_2 is empty

R-Rule 10 (*)**.** *If there are two* $(3, 2)$*-variables* x *and* y *such that* $F = (x \vee \overline{y}) \wedge (x \vee y \vee C_1) \wedge (x \vee y \vee C_2) \wedge (\overline{x} \vee y) \wedge \overline{x} \wedge (\overline{y} \vee D) \wedge F'$*, then set* $x = y$.

B-Rule 5. *If there are two* $(3, 2)$*-variables* x *and* y *such that* $F = (x \vee l_y) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (\overline{x} \vee D_1) \wedge (\overline{x}) \wedge F'$ *and literal* $\neg l_y$ *appear in at most 2 clauses with variable* x*, then branching on* y *gives us at least* $(11, 6)$*-branching.*

Proof. Let s be the total number of literals co-appearing with literal $\neg l_y$. We consider two branches, $l_y = 0$ and $l_y = 1$. In branch $l_y = 0$, R-Rule 3 would be applicable for variable x.

Let us analyze the decrease of measure in branch $l_y = 0$:

- if literal $\neg l_y$ and variable x do not share any clauses, then the measure decreases at least by $\Delta \geqslant s + 3$ since x becomes a 3-variable after application of R-Rule 3.
- if literal $\neg l_y$ and variable x share one clause, then $\Delta \geqslant s + 4$, since x becomes a 4-variable after assignment of $l_y = 0$ and disappears after R-Rule 3.
- if literal $\neg l_y$ co-appears twice with variable x, then $\Delta \geqslant s + 1 + 2 = s + 3$, since x becomes 3-variable after assigning $l_y = 0$ and disappears after the application of R-Rule 3.

By Lemma 4, we know that y's positive literals have at least three neighbors, and its negative literals have at least one neighbor. Branching on y gives at least $(6, 8)$-vector. The above argument implies that the measure additionally drops by 3 in one of the branches. So we have at least $(6, 11)$ or $(9, 8)$-branching. □

R-Rule 11 (*)**.** *If* x *is a* $(3, 2)$*-variable,* y *is a* $(2, 2)$*-variable and* $F = (x \vee y) \wedge (x \vee y) \wedge (x \vee \overline{y} \vee C) \wedge F'$*, then we can set* $x = \overline{y}$.

B-Rule 6. *If* x *is a* $(3, 2)$*-variable and* $F = (x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (\overline{x} \vee D) \wedge \overline{x} \wedge F'$*, then branching on* x *gives at least* $(11, 6)$*-branching vector.*

Proof. We can claim, using lemma 2, that each 3-variable appears in C_1, C_2, C_3 at most once and each $(3, 1)$-variable appears in C_1, C_2, C_3 at most twice.

If C_i contains only one literal and it belongs to 5-variable, then one of the following is true: (i) B-Rule 5 is applicable; (ii) R-Rule 10 is applicable; (iii) $F = (x \vee \overline{y}) \wedge (x \vee y \vee C_1) \wedge (x \vee y \vee C_2) \wedge (\overline{x} \vee y \vee E) \wedge \overline{x} \wedge (\overline{y} \vee D) \wedge F'$ and $|E| > 0$.

In case (iii) branching on x gives:

$$\left.\begin{aligned} \Delta_0 &\geqslant \underbrace{5}_{x} + \underbrace{1}_{E} + \underbrace{1}_{y} &= 7 \\ \Delta_1 &\geqslant \underbrace{5}_{x} + \underbrace{5}_{y} &= 10 \end{aligned}\right\} \Rightarrow (10, 7)$$

If 4-variable appears three times in the union of C_1, C_2, C_3, then the formula has the following type $F = (x \vee y \vee C_1') \wedge (x \vee y \vee C_2') \wedge (x \vee \overline{y} \vee C_3') \wedge (\overline{x} \vee D) \wedge \overline{x} \wedge F'$.

If at least two clauses among C_1', C_2', C_3' are not empty, then branching on x leads to:

$$\left.\begin{aligned} \Delta_0 &\geqslant \underbrace{5}_{x} + \underbrace{1}_{D} &= 6 \\ \Delta_1 &\geqslant \underbrace{5}_{x} + \underbrace{4}_{y} + \underbrace{2}_{C_1', C_2', C_3'} &= 11 \end{aligned}\right\} \Rightarrow (11, 6)$$

If C_3' is empty, then C_1' and C_2' are not empty; otherwise, R-Rule 3 is applicable. So, if $C_3' = \emptyset$ we have $(11, 6)$-branching.

If $C_3' \neq \emptyset$, then either we have a $(11, 6)$-branching or $C_1' = C_2' = \emptyset$. In this case, R-Rule 11 is applicable.

At this point, we have that if $C_i = l$ for some i, l, then l is not a literal of a 5-variable, each 3-variable appears at most once in C_1, C_2, C_3, and each 4-variable appears at most twice in C_1, C_2, C_3. Hence, deletion of C_i for any $i = 1, 2, 3$

decreases measure at least by 2. Since C_1, C_2, C_3 are not empty, branching on x gives us:

$$\left.\begin{aligned}\Delta_0 &\geqslant \underbrace{5}_{x} + \underbrace{1}_{D} &&= 6\\ \Delta_1 &\geqslant \underbrace{5}_{x} + \underbrace{2+2+2}_{C_1,C_2,C_3} &&= 11\end{aligned}\right\} \Rightarrow (11, 6)$$

□

D_2 is not empty

B-Rule 7 (*). *If x is a $(3,2)$-variable and $F = (x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (\overline{x} \vee D_1) \wedge (\overline{x} \vee D_2) \wedge F'$ and $|D_1|, |D_2| > 0$, then branching on x gives at least $(7, 9)$-branching.*

Again, in the section the worst branching vector is $(9, 7)$ so it is left to solve $(n, 4)$-MAXSAT in $O^*(1.0911^d)$, or faster than 1.4172^n.

Rules for 4-Variables

This section presents an algorithm for branching on 4-variables in $(n, 4)$-MAXSAT. Before we proceed, we recall a Lemma from (Alferov and Bliznets 2021):

Lemma 5. *Let x be a 4-variable in $(n, 4)$-MAXSAT formula such that literal x has t neighbors. Let $F_{x=1}$ be the formula obtained by setting $x = 1$ in F and exhaustive application of R-Rules 1-7. Then $d(F) - d(F_{x=1}) \geq 4 + 2t$*

(2, 2)-variables

Our formula can be represented as:
$F = (x \vee C_1) \wedge (x \vee C_2) \wedge (\overline{x} \vee D_1) \wedge (\overline{x} \vee D_2) \wedge F'$.

R-Rule 12 (R-Rule 10 in (Alferov and Bliznets 2021)). *If x is (2, 2)-variable and $F = (x \vee l) \wedge (x \vee l) \wedge F'$ or $F = (\overline{x} \vee l) \wedge (\overline{x} \vee l) \wedge F'$, then we can set $l = \overline{x}$ or $l = x$ respectively.*

Claim 1. *Clauses C_1, C_2 contain at least two variables. Moreover, if $|D_1|, |D_2| > 0$, then clauses D_1, D_2 also contain at least two variables.*

Proof. If the statement is not true, R-Rule 3 or R-Rule 12 are applicable. □

Due to R-Rules 1, 3 at most one of the clauses C_1, C_2, D_1, D_2 is empty. Let it be D_2. Below we consider two cases: $|D_2| > 0$ and $|D_2| = 0$.

D_2 is empty First, we present some reduction rules.

R-Rule 13 (R-Rule 13 in (Alferov and Bliznets 2021)). *If x is $(2,2)$-variable, $F = (x \vee C_1) \wedge (x \vee C_2) \wedge (\overline{x} \vee D_1) \wedge (\overline{x}) \wedge F'$ and C_1, C_2 contain complementary literals, then set $x = 0$.*

R-Rule 14 (*). *If x is a $(2,2)$-variable x and $F = (x \vee l \vee C_1) \wedge (x \vee C_2) \wedge (\overline{x} \vee l) \wedge \overline{x} \wedge F'$, then set $x = 0$.*

R-Rule 15 (R-Rules 15-16 in (Alferov and Bliznets 2021)). *Suppose x is a $(2,2)$-variable, l_y, t_y are literals of 3-variable y and $F = (x \vee \overline{l}_y \vee C_1') \wedge (x \vee C_2) \wedge (\overline{x} \vee l_y) \wedge \overline{x} \wedge (t_y \vee E) \wedge F'$. Let $G = (x \vee C_1' \vee E) \wedge (x \vee C_2) \wedge \overline{x} \wedge F'$.*

- *If $\overline{l}_y = t_y$, then set $x = l$;*
- *If $l_y = t_y$, then: $(F, k) \to (G, k-2)$.*

R-Rule 16 (*). *If x is a $(2,2)$-variable and y is a 4-variable such that $F = (x \vee y \vee C_1') \wedge (x \vee y \vee C_2') \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x}) \wedge (l_y \vee E) \wedge F'$, then set $x = 0$.*

R-Rule 17 (*). *If x is a $(2,2)$-variable, l is a literal, $F = (x \vee \overline{l}) \wedge (x \vee C) \wedge (\overline{x} \vee l) \wedge \overline{x} \wedge F'$, then set $x = l$.*

B-Rule 8 (*). *Let x be a $(2,2)$-variable, $|D_2| = 0$, $|C_1|$, $|C_2|$, $|D_1| > 0$, then it is enough to consider branching into two cases: (i) $x = 0$; (ii) $x = 1$, $C_1 = C_2 = 0$, $D_1 \neq 0$. It gives at least $(12, 6)$-branching. We note that if $|D_1| = 1$, we set $D_1 = 1$; otherwise, we are not using information that $D_1 \neq 0$.*

D_2 is not empty To tackle the case, first, we present a reduction rule.

R-Rule 18 (*). *Let x be a (2, 2)-variable, y be a (2, 2) or (2, 1)-variable and $F = (x \vee l_y) \wedge (\overline{x} \vee \overline{l}_y) \wedge (x \vee C) \wedge (\overline{x} \vee D) \wedge F'$, where l_y is a literal of y and C, D do not contain y. We can set $x = \overline{l}_y$.*

B-Rule 9. *Let x be a $(2,2)$-variable and $|C_1|$, $|C_2|$, $|D_1|$, $|D_2| > 0$. In this situation branching on x produce at least $(10, 8)$-branching.*

Proof. To analyze this branching rule, we consider several cases. Note that clauses D_1, D_2 are symmetrical with C_1, C_2; this allows us to reduce the number of considered cases.

If C_1, C_2 contains at least three distinct variables, then:

$$\left.\begin{aligned}\Delta_0 &\geqslant \underbrace{4}_{x} + \underbrace{2+2}_{D_1 \cup D_2} &&= 8\\ \Delta_1 &\geqslant \underbrace{4}_{x} + \underbrace{2+2+2}_{C_1 \cup C_2} &&= 10\end{aligned}\right\} \Rightarrow (8, 10)$$

Recall that by Claim 1 C_1, C_2 contain at least two different variables, as well as D_1, D_2. So, now it is enough to consider the case when the union of C_1, C_2 contains exactly two variables and the union of D_1, D_2 also contains exactly two variables. Assume that the union of C_1, C_2 contains variables y, z.

Firstly, we consider the case when y appears in C_1 and C_2.

From Lemma 2 follows that y is a 4-variable. Note that $\Delta_0 \geq 8$ as in the previous case and $\Delta_1 \geqslant 4 + 4 + 2 = 10$ (variables x, y disappear and one literal). Hence, again we have a $(8, 10)$-branching.

It is left to consider the case when $|C_1| = |C_2| = |D_1| = |D_2| = 1$. Let $C_1 = l_y, C_2 = l_z$, here l_y is a literal of variable y, and l_z is a literal of variable z.

If variable y appears in D_1 or D_2, then the formula can be represented as:

$$F = (x \vee l_y) \wedge (x \vee l_z) \wedge (\overline{x} \vee s_y) \wedge (\overline{x} \vee D_2) \wedge F'$$

If $s_y = l_y$, then R-Rule 3 is applicable. Hence, $s_y = \overline{l_y}$. If y is a $(2,2)$ or $(2,1)$-variable, then R-Rule 18 is applicable. So y is a $(3,1)$-variable. It follows that F' either contains two clauses with l_y or two clauses with $\bar{l}_y$. In this case, R-Rule 1 would be applicable for y either in branch $x = 0$ or in branch $x = 1$. Consequently, the measure drops at least by $4 + 2 + 2 = 8$ in the first branch (we eliminate x and two literals of different variables) and at least by $4 + 4 + 2 = 10$ in the second (we eliminate x, y and one additional literal). Hence, we have a $(10, 8)$-branching.

The last case is when $|C_1| = |C_2| = |D_1| = |D_2| = 1$ and the union of these clauses contains four distinct variables.

If there is a $(k, 1)$-variable among C_1, C_2, D_1, D_2. Let it be variable y. Without loss of generality, variable y appears in C_1. There are two possibilities: either $C_1 = y$ or $C_1 = \overline{y}$. In the first case $F = (x \vee y) \wedge (y \vee E_2) \wedge ... \wedge (y \vee E_k) \wedge (\overline{y} \vee H) \wedge F'$ and R-Rule 1 would be applicable for y in branch $x = 0$. So we get:

$$\left.\begin{array}{ll} \Delta_0 \geqslant \underbrace{4}_{x} + \underbrace{2+2}_{D_1 \cup D_2} + \underbrace{2}_{y} & = 10 \\ \Delta_1 \geqslant \underbrace{4}_{x} + \underbrace{2+2}_{C_1 \cup C_2} & = 8 \end{array}\right\} \Rightarrow (10, 8)$$

In the second case formula F can be represented as:

$$F = (x \vee \overline{y}) \wedge (x \vee l_z) \wedge (y \vee E_1) \wedge ... \wedge (y \vee E_k) \wedge F'$$

In branch $x = 1$, we apply R-Rule 1 to eliminate y. If there is a variable w in $\bigcup_{i=1}^{k} E_i$ distinct from z, then in branch $x = 1$ we eliminate variables x, y as well as literals of variables z, w. So we have:

$$\left.\begin{array}{ll} \Delta_0 \geqslant \underbrace{4}_{x} + \underbrace{2+2}_{D_1 \cup D_2} & = 8 \\ \Delta_1 \geqslant \underbrace{4}_{x} + \underbrace{2+2}_{C_1 \cup C_2} + \underbrace{2}_{\bigcup_{i=1}^{k} E_i} & = 10 \end{array}\right\} \Rightarrow (8, 10)$$

If there is no such variable, then y and z co-appear at least twice. So they cannot be both 3-variables; otherwise, R-Rule 9 is applicable. It means that at least one of these variables is a 4-variable. Since we apply R-Rule 1 in branch $x = 1$, we set $y = 1$, and after simplification, variable z disappears. Δ_0 is the same as in the previous case. $\Delta_1 \geqslant 4 + 4 + 2 = 10$, since we eliminate x, y, z (two 4-variables and one 3^+-variable).

The only unconsidered case is when all of the variables in C_1, C_2, D_1, D_2 are (2, 2)-variables. Since B-Rule 8 and all previous cases are not applicable, there is a closed subformula on $(2, 2)$-variables with each clause consisting of exactly two variables. So we can apply R-Rule 7 and then Corollary 1. ☐

(3, 1)-variables

We know that all 4-variables in our formula are $(3, 1)$-variables. So either our formula contains only of 3-variables, or we can apply the following branching rule:

B-Rule 10. *Let x be a $(3, 1)$-variable and F has the following type: $F = (x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (\overline{x} \vee D) \wedge F'$, then branch into two cases:*

1. $x = 0, C_1 = C_2 = C_3 = 1, D = 0$*;*
2. $x = 1$

Proof. The soundness of this rule is guaranteed by Lemma 1 from (Alferov and Bliznets 2021).

First of all, we prove an auxiliary claim:

Claim 2. *In branch $x = 1$ our measure decrease by $4 + 2(|C_1| + |C_2| + |C_3|)$.*

Proof. In order to show the statement, it is enough to prove that each 3-variable can appear at most once in C_1, C_2, C_3 and each 4-variable appears at most twice in C_1, C_2, C_3.

This claim is a direct corollary from Lemma 2. Indeed, since 4-variables are actually $(3, 1)$-variables they can appear at most twice in C_1, C_2, C_3. The same may be said about 3-variables that are $(2, 1)$-variables. ☐

Equipped with the recently proved claim, we show at least $(4, 16)$-branching. To show this, we consider several cases depending on the values of $|C_1|, |C_2|, |C_3|$. Note that all $|C_i| > 0$, since R-Rule 1 is inapplicable.

1. If $|C_1| = |C_2| = |C_3| = 1$: we know that in branch $x = 0$ it is sufficient to consider $C_i \neq 0$, so literals from C_i's will be assigned. Moreover, there are at least two variables in the union of C_1, C_2, C_3, so we have $\Delta_0 \geqslant 4 + (2 + 2) = 8$ and $\Delta_1 \geqslant 4 + 2(1 + 1 + 1) = 10$ by the proved claim.
2. If there is i, j such that $|C_i| = 1$ and $|C_j| > 1$, then we have: $\Delta_0 \geqslant 4 + 2 = 6$ and $\Delta_1 \geqslant 4 + 2(1 + 1 + 2) = 12$
3. If $|C_1|, |C_2|, |C_3| > 1$, then we have: $\Delta_0 \geqslant 4$ and $\Delta_1 \geqslant 4 + 2(2 + 2 + 2) = 16$.

The worst branching vector in this section is $(4, 16)$, so we can solve $(n, 4)$-MAXSAT in $O^*(1.08391^d)$ (in $O^*(1.3803^n)$) if we show how to solve $(n, 3)$-MAXSAT in $O^*(1.08391^d)$. ☐

Rules for 3-Variables

In this section, we present an algorithm for $(n, 3)$-MAXSAT. As we can eliminate variables that appear once and twice, we may assume that all variables appear exactly three times. In (Belova and Bliznets 2020), it was shown that either we can apply some reduction rules on formula F or we can find variable x such that

$$F = (x \vee C_1) \wedge (x \vee C_2) \wedge (\overline{x} \vee C_3) \wedge F',$$

with $|C_1|, |C_2|, |C_3| > 0$.

Moreover, the algorithm from (Belova and Bliznets 2020) has the following structure: (i) it applies exhaustively reduction rules in polynomial time; (ii) it applies branching rules from (Xu et al. 2019a) for cases when $(C_1, C_2, C_3) \in \{(1, 2, 2), (2, 2, 1), (\geq 3, \geq 2, 1)\}$; (iii) branching rules for all others cases depending on values of $|C_1|, |C_2|, |C_3|$. It is important to note that step (ii) contains all bottle-neck cases of algorithm from (Belova and Bliznets 2020). As the branching vector for rules from step (iii) is at least $(6, 3)$ in terms of n and $(12, 6)$ in terms of measure d. So it is enough to replace branching rules from step (ii) with more efficient ones. We consider two cases: (i) $|C_1| = 1, |C_2| = 2, |C_3| = 2$ and (ii) $|C_1| \geq 2, |C_2| \geq 2, |C_3| = 1$. We present branching with vectors at least $(12, 6)$ and $(16, 4)$ correspondingly. The discussion above shows that it is enough to prove a $O^*(1.1749^n)$ upper bound for $(n, 3)$-MAXSAT.

Before we proceed, we recall that no 3-variables coappear more than once due to R-Rule 9. Besides, we use the following lemma.

Lemma 6. *If $F = (x \vee \overline{y}) \wedge (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge (y \vee D_1) \wedge (y \vee D_2) \wedge F'$ then we can set $x = y$.*

Proof. Indeed, after assigning x to 1, variable y appears as positive literal only, so we should assign $y = 1$.

It is left to show that for an optimum assignment with $x = 0, y = 1$, we can find another optimum assignment with $x = 1, y = 1$. Indeed, if we substitute $y = 1$ in the formula and simplify, the formula will contain clauses $(x), (x \vee C_1), (\overline{x} \vee C_2)$ and it is evident that if we flip the value of x from 0 to 1, we gain at least one satisfied clause and lose at most one clause. So we get an optimum assignment with $x = y = 1$. □

B-Rule 11. *(*) If $|C_1| = 1, |C_2| = |C_3| = 2$, then there is at least $(16, 4)$-branching.*

B-Rule 12. *If $|C_1| \geqslant 2$, $|C_2| \geqslant 2$, $|C_3| = 1$, then there is a $(16, 4)$-branching*

Proof. We consider two cases: $C_3 = \overline{y}$ or $C_3 = y$.

If $C_3 = \overline{y}$ then:

$$F = (x \vee C_1) \wedge (x \vee C_2) \wedge (\overline{x} \vee \overline{y}) \wedge (y \vee D_1) \wedge (y \vee D_2) \wedge F.'$$

Simple branching on $x = 1$ and $x = 0$ gives us:

$$\left.\begin{array}{l} \Delta_0 \geqslant \underbrace{2}_{x} + \underbrace{2}_{y} + \underbrace{4}_{D_1, D_2} \quad = 8 \\ \Delta_1 \geqslant \underbrace{2}_{x} + \underbrace{4}_{C_1} + \underbrace{4}_{C_2} \quad = 10 \end{array}\right\} \Rightarrow (8, 10),$$

since D_1 and D_2 are not empty and cannot share variables, as well as $|C_1|, |C_2| \geq 2$ and also do not share any variables due R-Rule 9.

If $C_3 = y$ then:

$$F = (x \vee C_1) \wedge (x \vee C_2) \wedge (\overline{x} \vee y) \wedge (y \vee E_1) \wedge (\overline{y} \vee E_2) \wedge F'.$$

Here, we can apply Lemma 6 with roles of x, y changed. So after simplification, we get a $(3, 1)$-variable, and we can apply branching on it having at least $(4, 16)$-branching. □

The worst branching vector in this section is $(4, 16)$. So we can solve $(n, 3)$-MAXSAT in $O^*(1.08391^d)$ running time which is $O^*(1.1749^n)$. And with previous sections this imply $O^*(1.08391^d)$, $O^*(1.08391^L)$, $O^*(1.3803^n)$ algorithms for $(n, 4)$-MAXSAT, as well as $O^*(1.0911^d)$, $O^*(1.0911^L)$ algorithms for MAXSAT.

Algorithms in Terms of k and m

Theorem 1. *(Belova and Bliznets 2020) Assume that MAXSAT parameterized above matching can be solved in $O^*(c_1^{k'})$ time, (n, s)-MAXSAT can be solved in $O^*(c_2^n)$ and $c = c_2^{\frac{\log c_1}{\log c_1 + \log c_2}}$. In this case, we can check if in the input CNF we can satisfy at least k clauses in $O^*(c^k)$.*

Proof. We note that in (Belova and Bliznets 2020) the theorem is formulated only for $s = 3$. However, the proof never uses the fact and can be repeated for any s. □

Theorem 2. *$(n, 3)$-MAXSAT can be solved in $O^*(1.1554^k)$ running time. $(n, 4)$-MAXSAT can be solved in $O^*(1.2989^k)$ running time.*

Proof. In Corollary 6 from (Basavaraju et al. 2016), it was shown that MAXSAT parameterized above matching can be solved in $O^*(4^{k'})$. In the paper, we showed that $(n, 4)$-MAXSAT and $(n, 3)$-MAXSAT admit $O^*(1.3803^n)$ and $O^*(1.1749^n)$ algorithms respectively. Hence, we proved claimed results by Theorem 1. □

Conclusion

In the paper, we improve an algorithm for MAXSAT from $O^*(1.0927^L)$ to $O^*(1.0911^L)$. Moreover, we significantly improve running times for $(n, 4)$-MAXSAT from $O^*(1.4254^n)$ to $O^*(1.3803^n)$ and for $(n, 3)$-MAXSAT from $O^*(1.191^n)$ to $O^*(1.1749^n)$. Besides the results imply upper bounds in terms of k for $(n, 3)$-MAXSAT and $(n, 4)$-MAXSAT. The new upper bounds in terms of k are (1.1554^k) and $O^*(1.2989^k)$ respectively. So, most probably, for future improvements of MAXSAT, one only has to focus on 5-variables.

We note that results for $(n, 4)$-MAXSAT in terms of n were achieved thanks to the discounted measure introduced in (Alferov and Bliznets 2021), so our research once again emphasizes the importance of using non-standard measures in evaluating the complexity of input formulas. Also, our paper shows the usefulness of the measure even in obtaining results in terms of k and m.

Some of our reduction rules might be used in preprocessing steps of solvers. However, their efficiency requires further research. We found our new reduction rule, R-Rule 6, especially useful in proving upper bounds and analyzing our algorithm. However, its practical usefulness requires more investigation. Newly designed branching rules also might positively influence the construction of branching-based solvers.

Acknowledgements

The authors thank Tatiana Belova for fruitful discussions and for her comments on the draft.

Ivan Bliznets is supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 853234).

References

Alferov, V.; and Bliznets, I. 2021. New Length Dependent Algorithm for Maximum Satisfiability Problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5): 3634–3641.

Bacchus, F.; Järvisalo, M.; and Martins, R. 2021. *Maximum Satisfiability*, 929 – 991. Frontiers in Artificial Intelligence and Applications. Netherlands: IOS PRESS, 2 edition. ISBN 978-1-64368-160-3.

Bansal, N.; and Raman, V. 1999. Upper Bounds for MaxSat: Further Improved. In *Algorithms and Computation*, 247–258. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-46632-1.

Basavaraju, M.; Francis, M. C.; Ramanujan, M.; and Saurabh, S. 2016. Partially polynomial kernels for set cover and test cover. *SIAM Journal on Discrete Mathematics*, 30(3): 1401–1423.

Belova, T.; and Bliznets, I. 2020. Algorithms for (n, 3)-MAXSAT and parameterization above the all-true assignment. *Theoretical Computer Science*, 803: 222–233.

Biere, A.; Heule, M.; and van Maaren, H. 2021. *Handbook of satisfiability*. IOS press.

Bliznets, I. A. 2013. A new upper bound for (n,3)-MAX-SAT. *Journal of Mathematical Sciences*, 188(1): 1–6.

Chen, J.; Xu, C.; and Wang, J. 2017. Dealing with 4-variables by resolution: An improved MaxSAT algorithm. *Theoretical Computer Science*, 670: 33–44.

Chu, H.; Xiao, M.; and Zhang, Z. 2021. An Improved Upper Bound for SAT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3707–3714.

Fomin, F. V.; and Kratsch, D. 2010. *Exact exponential algorithms*. Springer-Verlag Berlin Heidelberg.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and intractability*, volume 174. Miller Freeman.

Gaspers, S.; and Sorkin, G. B. 2012. A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between. *Journal of computer and system sciences*, 78(1): 305–335.

Kulikov, A. S.; and Kutskov, K. 2009. New upper bounds for the problem of maximal satisfiability. *Discrete Mathematics and Applications*, 19(2): 155–172.

Li, W.; Xu, C.; Yang, Y.; Chen, J.; and Wang, J. 2022. A Refined Branching Algorithm for the Maximum Satisfiability Problem. *Algorithmica*, 84(4): 982–1006.

Marek, V. W. 2009. *Introduction to mathematics of satisfiability*. CRC Press.

Niedermeier, R.; and Rossmanith, P. 1999. New upper bounds for MaxSat. In *International Colloquium on Automata, Languages, and Programming*, 575–584. Springer.

Peng, J.; and Xiao, M. 2021. A Fast Algorithm for SAT in Terms of Formula Length. In *International Conference on Theory and Applications of Satisfiability Testing*, 436–452. Springer.

Raman, V.; Ravikumar, B.; and Rao, S. S. 1998. A simplified NP-complete MAXSAT problem. *Information Processing Letters*, 65(1): 1–6.

Xiao, M. 2022. An Exact MaxSAT Algorithm: Further Observations and Further Improvements. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 1887–1893. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Xu, C.; Chen, J.; and Wang, J. 2016. Resolution and linear CNF formulas: Improved (n,3)-MaxSAT algorithms. In *Theoretical Computer Science*.

Xu, C.; Li, W.; Wang, J.; and Yang, Y. 2019a. An improved algorithm for the $(n, 3)$-MaxSAT problem: asking branchings to satisfy the clauses. *Journal of Combinatorial Optimization*, 1–19.

Xu, C.; Li, W.; Yang, Y.; Chen, J.; and Wang, J. 2019b. Resolution and domination: an improved exact MaxSAT algorithm. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 1191–1197. AAAI Press.