New Length Dependent Algorithm for Maximum Satisfiability Problem

Vasily Alferov,[3] **Ivan Bliznets** [1 2]
[1] HSE University
[2] St. Petersburg Department of Steklov Mathematical Institute of Russian Academy of Sciences
[3] JetBrains Research
vasily.v.alferov@gmail.com, iabliznets@gmail.com

Abstract

In this paper, we study the computational complexity of the MAXIMUM SATISFIABILITY problem in terms of the length L of a given formula. We present an algorithm with running time $O(1.0927^L)$, hence, improving the previously known best upper bound $O(1.1058^L)$ developed more than 20 years ago by Bansal and Raman. Theoretically speaking, our algorithm increases the length of solvable formulas by 13.3% (compare this to the recent breakthrough result for MAXIMUM SATISFIABILITY problem with respect to the number of clauses by Xu et al. in 2019 giving a 7.5% improvement). Besides, we propose a significantly simpler algorithm with running time $O(1.1049^L)$. The algorithm outperforms Bansal's and Raman's algorithm in simplicity and running time.

Introduction

The SATISFIABILITY problem is a well-known problem that plays a tremendous role in Computer Science, Artificial Intelligence, and has a lot of applications. In this paper we consider its optimisation version called MAXIMUM SATISFIABILITY, MAXSAT for short. This problem also has lots of applications: bioinformatics, hardware debugging, software debugging, scheduling, probabilistic reasoning, electronic markets, to name a few. More details about applications of MAXSAT can be found in the survey (Morgado et al. 2013).

In the MAXIMUM SATISFIABILITY problem one is given a boolean formula in Conjunctive Normal Form (CNF) and the goal is to satisfy the maximum number of clauses simultaneously. It is one of the first problems that was shown to be NP-hard. So the existence of an efficient exact algorithm for this problem is unlikely. That is why almost all possible approaches and methods were used to cope with the computational hardness of the problem. Researchers tried to study special cases of the problem (Williams 2005; Belova and Bliznets 2020), designing randomized and approximation algorithms (Goemans and Williamson 1994; Poloczek et al. 2017), constructing exact (Bansal and Raman 1999; Xu et al. 2019) and parameterized algorithms (Crowston et al. 2014), as well as developing different heuristics (Berg, Saikko, and Järvisalo 2015; Ignatiev, Morgado, and Marques-Silva 2019). There is even an annual competition among MAXSAT solvers, called MAXSAT Evaluation (maxsat-evaluations.github.io).

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Running time	References
$O^*(1.3803^m)$	(Niedermeier and Rossmanith 1999)
$O^*(1.3412^m)$	(Bansal and Raman 1999)
$O^*(1.3248^m)$	(Chen and Kanj 2004)
$O^*(1.2989^m)$	(Xu et al. 2019)

Table 1: Progress for MAXSAT in terms of m

Running time	References
$O^*(1.618^k)$	(Mahajan and Raman 1999)
$O^*(1.400^k)$	(Niedermeier and Rossmanith 1999)
$O^*(1.381^k)$	(Bansal and Raman 1999)
$O^*(1.370^k)$	(Chen and Kanj 2002)
$O^*(1.358^k)$	(Bliznets and Golovnev 2012)
$O^*(1.325^k)$	(Chen, Xu, and Wang 2015)

Table 2: Progress for MAXSAT in terms of k

The main goal of the paper is to improve upper bound on the worst-case computational complexity of the MAXIMUM SATISFIABILITY problem. So we are interested in an exact algorithm for MAXSAT. Generally, the complexity of an input instance of some problem is measured in its input size. For the MAXIMUM SATISFIABILITY problem there are four natural measures which describe the complexity of an instance of MAXSAT: n – the number of different variables, m – the number of clauses, k – the number of clauses that one wants to satisfy, L – the overall number of literals in the instance. The last measure is the closest to the bit-size measure. We note that even solving SATISFIABILITY problem in time $O^*((2-\epsilon)^n)$ is a big open problem and many conjectured that actually, this is impossible to do (Strong Exponential time Hypothesis (Impagliazzo and Paturi 2001)). The situation is different for other measures and there is a significant line of research that attempts to solve MAXSAT in terms of m, k and L, see tables 1, 2, 3.

First of all, we compare our results with the result of Bansal and Raman in terms of L (previously best known upper bounds, see table 3). Our first algorithm is just slightly faster. However, its main advantage is simplicity, as it con-

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

Table 3: Progress for MAXSAT in terms of L

tains 7 branching rules while Bansal's algorithm consists of 11 branching rules and some of these branching rules have complicated structure with subcases. Our second result significantly improves Bansal's algorithm. Theoretically, it allows us to handle inputs 13.3% longer than before. That is, having the same computational power and ignoring polynomial factors (as they are not important for sufficiently large inputs, this is theoretical assumption) new input can be $\frac{\log(1.1057)}{\log(1.0927)} - 1$ times longer and answer will be computed within the same time.

It is hard to compare our results with the results from (Chen, Xu, and Wang 2015; Xu et al. 2019) since they are given with regard to different measures. We note that these recent improvements theoretically make it possible to increase possible values of k and m respectively by 8.7% and 7.5% compared to the previous best upper bounds, while our improvement allows to increase the length by 13.3%. However, we cannot say that our algorithm is superior to those presented in (Chen, Xu, and Wang 2015; Xu et al. 2019). So, Xu et al. algorithm is preferable when clauses have relatively long length. Our algorithm is preferable when the input formula has a noticeable number of short clauses of length 1. For example, if we are given a formula F where each clause has length at most 3 and at least 5% of clauses have length 1, then our upper bound is better than one given in (Xu et al. 2019).

Moreover, our improvement is achieved using a completely different approach from the approach used in previous works dedicated to the study of the general case of MAXIMUM SATISFIABILITY. We achieve our improvement mainly because of using a new measure called *discounted length* equal to $L - n_3$ where n_3 is the number of variables appearing exactly three times. So our main power is a measure-and-conquer approach, while the rest of the results presented in tables 1, 2, 3 were achieved by designing novel branching and reduction rules. It is worth mentioning that in (Xu et al. 2019) there are 16 reduction rules, half of which are new. However, almost all of the new reduction rules are inapplicable in our case, since they might increase the total length of a formula which is unacceptable for a reduction rule in our case. We note that measure-and-conquer was applicable before only for special cases of MAXIMUM SATISFIABILITY like MAX-2-SAT (each clause has length at most 2) (Kojevnikov and Kulikov 2006) or SATISFIABILITY problem (Chen and Liu 2009). We note that even though SAT and MAXSAT are related problems they might require different approaches, see (Liu and De Melo 2017). And in our case, a significant difference lies in clauses of length one. Clauses of length one are trivially resolved in case of the SATISFIABILITY problem while in case of MAXIMUM SATISFIABILITY they cause major problems as a satisfaction of such clause decrease the length measure only by one.

Preliminary

In the paper we assume familiarity with such notions like boolean variable, literal, clause. We refer interested reader to book (Marek 2009) for details.

The length of a clause is defined as the number of literals in the clause. A clause of length one is called a unit-clause. In this work we denote clauses by capital letters C, D, E. A formula is in Conjunctive Normal Form (CNF for short) if it is a conjunction of clauses. For example, the formula $\overline{x} \wedge \overline{y} \wedge \overline{z} \wedge (x \vee y) \wedge (y \vee z) \wedge (x \vee z)$ is in CNF. The length of a formula is defined as the total number of literals in all of its clauses.

An assignment is a function that assigns to each variable from the formula value 0 or 1. A clause is satisfied by an assignment if some of its literals gets the value 1. An assignment is optimal if it satisfies the maximum number of clauses.

A variable is called k-variable if it appears exactly k times in a formula. If a variable appears at least k times, it is called a k^+-variable, and if it appears at most k times, it is called a k^--variable. Similarly, we define k-literals, k^+-literals and k^--literals. If for a variable x the literal x appears k times in the formula and the literal $\overline{x}$ appears l times in the formula, x is called a (k, l)-variable. Similarly, we define (k, l)-literals. Since replacing variable x with $\overline{x}$ in the entire formula does not change the answer for the problem instance, we always assume $k \geq l$ for (k, l)-variables.

A subformula is a subset of clauses of the initial formula. A subformula is called closed if no literal of a variable with literals inside the subformula appear outside the subformula.

Variables x and y are called neighbors if there is a clause in the formula with literals of both x and y. Similarly, literals l and m are called neighbors if there's a clause in the formula containing both l and m.

Our algorithms follow standard branch-and-bound technique enhanced with measure-and-conquer approach. As other algorithms with such technique our algorithms consist of reduction and branching rules. A reduction rule (R-Rule for short) is a polynomial-time algorithm that transforms an instance of MAXSAT into an equivalent instance with the same or smaller measure value (*discounted length* in case of this paper). A branching rule (B-Rule for short) is a polynomial-time algorithm that transforms an instance of MAXSAT into several instances with smaller measure value, on which the algorithm is launched recursively. A branching rule is correct if the initial instance is a YES-instance if and only if one of the transformed instances is a YES-instances. If a branching rule transforms in polynomial time an instance with measure L to several instances with measures $L - a_1, L - a_2, \ldots, L - a_k$ we call $(a_1, a_2, \ldots, a_k)$ – the branching vector of this rule. The only positive root of the polynomial $x^L = x^{L-a_1} + x^{L-a_2} + \cdots + x^{L-a_k}$ is called the branching number of the corresponding rule. If an algorithm uses only branching rules with branching numbers $c_1, c_2, \ldots, c_t$ then the running time of the algorithm is

bounded by $(\max_i^t c_i)^L poly(L)$. More detailes about branch and bound technique can be found in book (Fomin and Kratsch 2010). R-Rules and B-Rules are applied exhaustively to an instance in the order they appear in the description of the algorithm.

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.

Instead of measuring the complexity of our algorithm in terms of L we analyze it in terms of so called discounted length. The discounted length $d(F)$ (sometimes me simply write d when F is clear from the context) for formula F is equal to $L - n_3 = 2n_3 + 4n_4 + 5n_5 + \ldots$, where L is the length of a formula F and n_i is the number of variables that appear exactly i times in the F. We note that the measure plays a significant role in the proof of the worst case analysis, without its usage we cannot prove the bound. It is obvious that if we obtain $O(c^d)$ algorithm we also obtain $O(c^L)$ algorithm since $d \leq L$.

Simple Algorithm

First of all we list some known reduction rules. Note that all these rules do not increase the discounted length d of the formula. We write $(F, k) \rightarrow (F', k')$ if reduction rule transforms formula F into F' and it is possible to satisfy k clauses in F if and only if it is possible to satisfy k' clauses in F'.

Known Reduction and Branching Rules

R-Rule 1. Let x be a variable such that both literals x and $\overline{x}$ are contained in the same clause $x \vee \overline{x} \vee C$. Then we can remove this clause, i.e. $((x \vee \overline{x} \vee C) \wedge F', k) \rightarrow (F', k-1)$.

R-Rule 2. Let l be a literal such that $\overline{l}$ does not appear in the formula. Then we can set $l = 1$.

R-Rule 3. Let C be a clause and x be a variable such that both clauses $x \vee C$ and $\overline{x} \vee C$ appear in the formula. Then we can replace both clauses with one clause C, i.e. $((x \vee C) \wedge (\overline{x} \vee C) \wedge F', k) \rightarrow (C \wedge F', k-1)$.

In particular, if R-Rule 3 is not applicable, for any variable x at most one of the literals x and $\overline{x}$ can appear in unit clauses.

R-Rule 4. 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 with x with clause $C \vee D$, i.e. $(F, k) \rightarrow ((C \vee D) \wedge F', k-1)$.

R-Rule 5. Let l be an (i, j)-literal that appears in t unit clauses, and $t \geq j$. Then we can set $l = 1$, i.e. $(F, k) \rightarrow (F_{[l=1]}, k-i)$.

R-Rule 6 ((Xu et al. 2019))**.** Let x be a $(i, 1)$-variable ($i \geq 2$) such that every clause containing x contains the same literal l. Then we can remove l from all clauses containing x and add it to the clause containing $\overline{x}$, i.e. $((x \vee l \vee C_1) \wedge (x \vee l \vee C_2) \wedge \cdots \wedge (x \vee l \vee C_i) \wedge (\overline{x} \vee D) \wedge F', k) \rightarrow ((x \vee C_1) \wedge (x \vee C_2) \wedge \cdots \wedge (x \vee C_i) \wedge (\overline{x} \vee l \vee D) \wedge F', k)$.

R-Rule 7. If in F there is a closed subformula F_1 on five or fewer variables then find the answer for F_1 in constant time.

Lemma 1. *Let x be a $(i, 1)$-variable in formula $F = (x \vee C_1) \wedge \cdots \wedge (x \vee C_i) \wedge (\overline{x} \vee D)$. Then branching on the following two cases is correct:*

1. $x = 1$
2. $x = 0$, $D = 0$, *and for each j such that $|C_j| = 1$, $C_j = 1$.*

Proof. Consider an optimal assignment with $x = 0$. If $D = 1$ in this assignment, setting $x = 1$ would satisfy at least the same number of clauses (thereby being an optimal assignment). Similarly, if $C_j = 0$ for some j, setting $x = 1$ would also satisfy at least the same number of clauses. Hence, there's always an optimal assignment satisfying one of these cases. □

Discounted Length

In the subsection we provide some intuition why we are using discounted length as well as we prove some useful properties of this measure.

(n, i)-MAXSAT is a special case of MAXSAT where each variable occurs at most i times. There is an $O(1.191^n)$ algorithm for $(n, 3)$-MAXSAT. That is equivalent to $O(1.06^L)$ running time since in this case $L = 3n$. So, instances with 3 variables have efficient algorithm and it is reasonable to allocate some "bonus" if instance become closer to instance of MAXSAT where each variable appears at most three times. Such bonus scheme is realised by decrease of the measure when a 4-variable becomes a 3-variable.

Lemma 2. *Let x be an (i, j)-variable ($i + j \geq 4$) in formula $F = (x \vee C_1) \wedge \ldots \wedge (x \vee C_i) \wedge (\overline{x} \vee D_1) \wedge \ldots \wedge (\overline{x} \vee D_j) \wedge F'$. Let $F_{x=1}$ be the formula obtained by setting $x = 1$ in F and exhaustive application of R-Rules 1-7. Then $d(F) - d(F_{x=1}) \geq i + j + \sum_{k=1}^{i} |C_k|$.*

Proof. As neither of R-Rules 1-7 create a new variable, we consider the measure decrease as the sum of decreases of weights of variables affected by the branching.

As variable x is eliminated, it contributes $i + j$ to the total decrease of d.

Now, consider a neighbor y of the literal x. If y is a 4^+-variable in both F and $F_{x=1}$, its weight decreases exactly by the number of its occurrences in all of Cs. If it is a 3-variable in $F_{x=1}$, its weight decreases even more. If it is a 2^--variable in $F_{x=1}$, it is instantly eliminated by R-Rules 2 or 4. Finally, if it was a 3-variable in F, it must have been a $(2, 1)$-variable, and hence couldn't appear all three times in Cs (otherwise R-Rule 6 would be applicable). Hence, it had at most two occurrences in all of Cs, and as it is instantly eliminated in $F_{x=1}$, its weight is decreased by two.

Thus, since every neighbor of x loses at least number of its occurrences in Cs in weight, the measure decreases at least by $i + j + \sum_{k=1}^{i} |C_k|$. □

Lemma 3. *Let x be an (i, j)-variable ($i + j \geq 4$) in the formula $(x \vee C_1) \wedge \ldots \wedge (x \vee C_i) \wedge (\overline{x} \vee D_1) \wedge \ldots \wedge (\overline{x} \vee D_j) \wedge F'$.*

Then 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 on measure d.

Proof. Apply lemma 2 to both sides of the branch. □

The last lemma allows us to directly introduce a branching rule for 6^+-variables.

B-Rule 1. If there's a 6^+-variable x, branch on it.

This gives at least a $(7,7)$-branching, i.e. in all cases branching numbers are not bigger than branching number of branching vector $(7,7)$.

Proof. If x is a 7^+-variable, this gives at least a $(7,7)$-branching.

If x is a $(5,1)$-variable, after R-Rules 3 and 5 it can only appear negatively in unit clauses. Hence, Lemma 3 guarantees at least a $(6,11)$-branching.

If x is a $(4,2)$-variable, after R-Rules 3 and 5 it can either appear only positively in unit clauses (at most once), or only negatively. Hence, Lemma 3 guarantees at least a $(8,9)$- or a $(6,10)$-branching.

If x is a $(3,3)$-variable, after R-Rules 3 and 5 it can appear in at most two unit clauses, either only positively or only negatively. Hence, Lemma 3 guarantees at least a $(7,9)$-branching. □

Rules for 5-variables

It is obvious that a 5-variable is either a $(4,1)$-variable or a $(3,2)$-variable. At first we consider the case when the formula contains $(4,1)$-variable. WLOG formula has the following type:

$$(x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (x \vee C_4) \wedge (\overline{x} \vee D) \wedge F'$$

R-Rule 8. Let x be a $(4,1)$-variable. Suppose there's such j that $|C_j| = 1$ and the literal from C_j appears in D. Then we can set $x = 1$.

Proof. Denote the literal from C_j as l. Consider an optimal assignment. If $l = 1$ in this assignment, then setting $x = 1$ in such assignment would satisfy all clauses with x. If $l = 0$ and $x = 0$, then setting $x = 1$ would satisfy at least the same amount of clauses, thereby being an optimal assignment. □

B-Rule 2. Let x be a $(4,1)$-variable such that R-Rule 8 is not applicable. Branch on two cases:

1. $x = 1$
2. $x = 0$, $D = 0$, and for each j such that $|C_j| = 1$, $C_j = 1$.

This gives at least a $(7,9)$-branching.

Proof. Correctness of this rule is guaranteed by lemma 1.

Note that after R-Rule 5 all of Cs are not empty.

If $|D| > 0$, this is at least a $(7,9)$-branching: at least 9 literals are eliminated in the first branch, giving decrease of at least 9 in the first case by lemma 3, and in the second case a 5-variable x and at least one 3^+-variable are eliminated, thus giving decrease of at least 7.

If $|D| = 0$, and $|C_j| = 1$ for some j, this is at least a $(7,9)$-branching. Similarly, in the first case d decreases by at least 9, and in the second case a 5-variable x and a 3^+-variable C_j are eliminated.

Finally, if $|D| = 0$, and $|C_j| > 1$ for each j, lemma 3 guarantees at least a $(5,13)$-branching. □

From now on, we can assume that every 5-variable is a $(3,2)$-variable. If F has 5-variable then it has the following type:

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

Now, we consider cases when $D_1 = D_2 = \emptyset$.

R-Rule 9. If x is a $(3,2)$-variable such that $D_1 = D_2 = \emptyset$, and the union of Cs contains complementary literals then set $x = 0$.

Proof. In this case at least one of Cs is satisfied in any assignment. Hence, setting $x = 0$ would satisfy at least three clauses with the variable x, while setting $x = 1$ would satisfy exactly three of them. □

B-Rule 3. If x is a $(3,2)$-variable such that $D_1 = D_2 = \emptyset$, branch on two cases:

1. $x = 0$
2. $x = 1$ and $C_1 = C_2 = C_3 = 0$

This gives at least a $(5,10)$-branching vector.

Proof. First, we prove the correctness of the rule.

Consider an optimal assignment with $x = 1$. If any of Cs is satisfied in this assignment, then setting $x = 0$ would satisfy at least three clauses with x, while in the initial assignment exactly three of them were satisfied. Hence, there exists an optimal assignment with either $x = 0$ or with $x = 1$ and all Cs unsatisfied.

Now we prove that the branching gives at least $(5,10)$ branching vector.

If the union of Cs contains literals of at least three distinct variables, it is at least a $(5,11)$-branching: in both cases a 5-variable x is eliminated, and in the second case three 3^+-variables are also eliminated.

If the union of Cs contains literals of exactly two distinct variables, they are not both 3-variables. Indeed, after R-Rules 9 and 6 are inapplicable, a 3-variable can appear in the union of C_j at most once, and there are at least three literals in this union. Hence, one of those two variables is a 4^+-variable and it is similarly at least a $(5,11)$-branching.

Finally, if the union of Cs contains literals of exactly one variable, it must be a 5-variable. Indeed, if it were a 4^--variable, after R-Rule 9 is inapplicable, this variable must have three identical literals in the union of C_i, which is only possible for $(3,1)$-variables. But then R-Rule 6 would be applicable. Hence, we have at least a $(5,10)$-branching vector. □

From now on we can assume that $D_1 \cup D_2 \neq \emptyset$

B-Rule 4. If x is a $(3,2)$-variable such that $|D_1|+|D_2| > 0$, branch on x.

This gives at least a $(6,8)$-branching vector.

Proof. If $|C_i| > 0$ for all i then we have at least a $(6, 8)$-branching by lemma 3.

If $|C_1| = 0$, then all of C_2, C_3, D_1, D_2 are non-empty and hence we have at lest a $(7, 7)$-branching. □

Rules for 4-variables

In this section all formulas contain only variables that appear at most 4 times. Hence, our measure $d = 2n_3 + 4n_4$.

Lemma 4. *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 $d(F) - d(F_{x=1}) \geq 4 + 2t$*

Proof. As in lemma 2, we count the total measure decrease as the sum of weight decreases for all affected variables.

For x, which is a 4-variable, the weight decreases by 4.

If y is a neighbor of the literal x, and y is a 4-variable, it becomes a 3^--variable, and its weight decreases by at least 2.

If y is a neighbor of the literal x, and y is a 3-variable, it becomes a 2^--variable, which is instantly eliminated, and its weight decreases by 2.

Hence, the measure decreases by at least $4 + 2t$. □

Lemma 5. *Let x be a 4-variable in $(n, 4)$-MAXSAT formula such that the literal x has a neighbors, and literal $\overline{x}$ has b neighbors. Then branching on x gives at least a $(4 + 2a, 4 + 2b)$-branching.*

Proof. Apply lemma 4 to both sides of the branching. □

We proceed with the rules for 4-variables. A 4-variable is either a $(2, 2)$-variable or a $(3, 1)$-variable. In the case of $(2, 2)$-variables the formula has the following type:

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

Recall that only one clause of the C_1, C_2, D_1, D_2 can be empty. WLOG if one of them is empty, then it is D_2.

First of all we deal with the case when $C_1 \cup C_2$ contains only literals of one variable. Note that these literals cannot be complementary by R-Rule 3. If they are identical, the following rule is applicable:

R-Rule 10. If x is a $(2, 2)$-variable, and y is such literal that $C_1 = C_2 = y$, set $y = \overline{x}$.

Proof. First, we prove that $y = 0$ implies $x = 1$.

Indeed, if we set $y = 0$, then x is left a 4^--variable with two positive occurrences in unit clauses. By R-Rule 5 it is instantly assigned with value 1.

Now, we prove that $y = 1$ implies $x = 0$. Indeed, if we set $y = 1$, by R-Rule 2 x is instantly assigned with value 0. □

Note that after this rule is applied, the formula remains an $(n, 4)$-MAXSAT instance. Indeed, at least four literals of a new 8-variable are instantly eliminated by R-Rule 1.

After application of this rule, the union of C_j contains literals of at least two distinct variables. Hence, the next rule follows from Lemma 5 and gives at least a $(6, 8)$-branching.

B-Rule 5. If x is a $(2, 2)$-variable, branch on x.

Now, we are left with $(3, 1)$-variables and 3-variables.

If the formula contains $(3, 1)$-variables then it has the following type:

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

Note that the union of C_i contains literals of at least two distinct variables. Indeed, complementary literals of exactly one variable are prohibited by R-Rule 3, and same literals of exactly one variable are prohibited by R-Rule 6.

For this case we provide two similar branching rules:

B-Rule 6. If x is a $(3, 1)$-variable, and $|D| > 0$, branch on two cases:

1. $x = 1$
2. $x = 0$, $D = 0$, and for each j such that $|C_j| = 1$, $C_j = 1$.

This gives at least an $(8, 6)$-branching.

Proof. Correctness of this rule is guaranteed by lemma 1.

In the first case d decreases by at least 8 by Lemma 5, and in second case we eliminate a 4-variable x and at least one 3^+-variable from D, hence, d decreases by at least 6. □

B-Rule 7. If x is a $(3, 1)$-variable, branch on two cases:

1. $x = 1$
2. $x = 0$, and for each j such that $|C_j| = 1$, $C_j = 1$

This gives at least a $(6, 8)$-branching.

Proof. Correctness of this rule is guaranteed by lemma 1.

If $|C_j| = 1$ for some j, it is at least a $(6, 8)$-branching: in the first case d decreases by at least 8 by Lemma 5, and in second case we eliminate a 4-variable x and a 3^+-variable, and hence d decreases by at least 6.

Otherwise, $\sum |C_i| \geq 6$.

First, note that no variable can appear in the union of C_j three times. Indeed, for $(2, 1)$-variables it is prohibited by R-Rule 6. Three positive occurrences of a $(3, 1)$-variable are also prohibited by R-Rule 6. And since B-Rule 6 is inapplicable, any negative occurrence of a $(3, 1)$-variable is prohibited.

If there are literals of at least four distinct variables in the union of C_j, by lemma 5 it is at least a $(4, 12)$-branching.

Otherwise, there are exactly three distinct variables in the union of C_j, each appearing exactly twice.

If there is a 4-variable among those variables, it is also at least a $(12, 4)$-branching, as it is instantly eliminated in the first branch and hence its weight decreases by at least 4, while weights of other variables decrease by at least two each.

Otherwise, all three variables are 3-variables. Each of those variables have exactly one literal out of the union of C_j. If any of them has new neighbor (i.e different from x and any of those three variables), in the first case number of occurrences of this neighbor decreases, and hence it is at least a $(12, 4)$-branching. Otherwise, x and these three variables form a closed subformula on 4 variables that is solved by R-Rule 7. □

Note that the two rules above are essentially the same. They are split in two in order to simplify the branching vector analysis.

Time Analysis

Theorem 1. MAXSAT *can be solved in* $O(1.1049^d)$ *time. Hence,* MAXSAT *is also solvable in* $O(1.1049^L)$ *time.*

Proof. After exhaustive application of all above rules we are left with an instance of $(n, 3)$-MAXSAT. We run algorithm from (Belova and Bliznets 2020) on it. This algorithms solves $(n, 3)$-MAXSAT in $O(1.191^n) = O(1.191^{n_3})$ time (Belova and Bliznets 2020). Since for $(n, 3)$-MAXSAT $d = 2n_3$, the time bound with respect to d is $O^*(1.0912^d)$.

During the whole algorithm we perform branchings with at least the following branching vectors $(5, 10), (7, 7), (6, 8)$. These branching vectors have the following branching numbers $1.1011, 1.1041, 1.1049$ correspondingly. So, since $(n, 3)$-MAXSAT is solvable in $O^*(1.0912^d)$ the total running time of our algorithm is at most $O(1.1049^d)$. □

This result provides a slight improvement for the previous bound of $O^*(1.1057^L)$ (Bansal and Raman 1999).

Main Algorithm

In this section we provide a more involved algorithm and detailed analysis for the measure d. This allows us to construct a significantly faster algorithm for MAXSAT. We note that here we reuse all R-rules described before as well as B-Rules 1-3,6 with a more detailed analysis. Due to space constraints all proofs in this section are omitted.

Rules for 5^+-variables

The first observation is that B-Rule 1 in fact guarantees a better branching vector.

Lemma 6. *B-Rule 1 gives at least a* $(6, 10)$ *branching vector.*

For $(4, 1)$-variables R-Rule 8 and B-Rule 2 already give a $(7, 9)$ branching vector. So we can assume that if F contains a 5-variable then the formula can be represented as: $(x \vee C_1) \wedge (x \vee C_2) \wedge (x \vee C_3) \wedge (\overline{x} \vee D_1) \wedge (\overline{x} \vee D_2) \wedge F'$

At first we consider case $|D_1| = |D_2| = 0$. For this case, we provide a tight analysis of B-Rule 3

As long as R-Rule 9 is inapplicable, the union of C_i does not contain a pair of complementary literals. Another special case is $C_1 = C_2 = C_3 = l$ for some literal l.

R-Rule 11. If x is a $(3, 2)$-variable such that $|D_1| = |D_2| = 0$, and all literals in the union of Cs are literals of the same variable (which means, after R-Rule 9 is inapplicable, that $C_1 = C_2 = C_3 = l$ for some literal l), set $x = 0$ and $l = 1$.

This rule allows us to obtain better branching vector for B-Rule 3.

Lemma 7. *If R-Rule 11 is inapplicable, B-Rule 3 gives at least a* $(5, 12)$*-branching.*

Another special case for $(3, 2)$-variables is when one of Cs is empty. In this case, both D_1 and D_2 are not empty.

R-Rule 12. If x is a $(3, 2)$-variable, one of Cs is empty, and D_1 and D_2 contain a pair of complementary literals, set $x = 1$.

B-Rule 8. If x is a $(3, 2)$-variable, one of Cs is empty, and the above rule is inapplicable, branch on two cases: (i) $x = 1$; (ii) $x = 0$, $D_1 = D_2 = 0$. This gives at least a $(7, 9)$-branching.

At this point, all Cs and at least one of Ds are non-empty. For this case, we start with the correctness of the following two generic branching rules.

Lemma 8. *If* x *is a* $(3, 2)$*-variable, the above rules are inapplicable,* $|C_1| = 1$*, and there exists an assignment that satisfies* C_2 *and* C_3 *and does not satisfy* C_1*,* D_1 *and* D_2*, then branching on the following three cases: (i)* $x = 1$*, (ii)* $x = 0$*,* $C_1 = 1$*, (iii)* $x = 0$*,* $C_1 = 0$*, if for* $j = 2$ *or* $j = 3$ *holds* $|C_j| = 1$*,* $C_j = 1$*, and if for some* j *holds* $|D_j| > 0$*,* $D_j = 0$*, is correct.*

Lemma 9. *If* x *is a* $(3, 2)$*-variable, the above rules are inapplicable,* $|C_1| = 1$*, and there is no assignment that satisfies* C_2 *and* C_3 *and does not satisfy* C_1*,* D_1 *and* D_2*, then branching on the following two cases: (i)* $x = 1$*, (ii)* $x = 0$*,* $C_1 = 1$*, is correct.*

Now consider several cases on lengths of Cs and Ds

B-Rule 9. Let x be a $(3, 2)$-variable such that the above rules are inapplicable. If $\sum |C_i| + \sum |D_i| \geq 6$, branch on x. This gives at least a $(6, 10)$-branching.

In particular, this rule applies if $\sum |C_i| \geq 5$. At this point, $3 \leq \sum |C_i| \leq 4$.

Now we provide several rules for the case of x having small number of neighbors.

B-Rule 10. Let x be a $(3, 2)$-variable such that the above rules are inapplicable. If there exists such j that $|C_j| = 1$ and the literal from C_j occurs in Ds, branch on two cases: (i) $x = 1$, (ii) $x = 0$, $C_k = 1$, where $k \neq j$ and $|C_k| = 1$. This gives at least a $(8, 8)$-branching.

B-Rule 11. Let x be a $(3, 2)$-variable such that the above rules are inapplicable, and let j be such an index that $|C_j| = 1$. Denote $C_j = l$. If $\bar{l}$ appears in Ds, branch on two cases: (i)$x = 1$, (ii) $x = 0$, $l = 1$. This gives at least a $(7, 9)$-branching.

B-Rule 12. Let x be a $(3, 2)$-variable. If there exist i and j such that $|C_i| = |C_j| = 1$ and $C_i = C_j$, then branch on two cases: (i) $x = 1$, (ii) $x = 0$, $C_i = 1$. This gives at least a $(8, 9)$-branching.

Note that the same case for complementary clauses C_i and C_j is prohibited by R-Rule 3.

After B-Rule 9 is inapplicable, $\sum |C_i|$ can be either equal to 3 or 4. In the next few rules we consider the case $\sum |C_i| = 4$. Note that this implies $\sum |D_i| = 1$. For simplicity we will assume $|D_1| = 1$ and $|D_2| = 0$.

B-Rule 13. Let x be a $(3, 2)$-variable such that $\sum |C_i| = 4$ and $\sum |D_i| = 1$. If the above rules are inapplicable, and the union of Cs consists of literals of exactly two 3-variables (two literals from each of them), branch on any of those 3-variables. This gives at least a $(9, 9)$-branching.

B-Rule 14. Let x be a $(3,2)$-variable such that $\sum |C_i| = 4$ and $\sum |D_i| = 1$. If the above rules are inapplicable, and there is a variable in either union of Cs or Ds that is not a 5-variable appearing in this union once, branch on x. This gives at least a $(6,10)$-branching.

B-Rule 15. Let x be a $(3,2)$-variable such that $\sum |C_i| = 4$ and $\sum |D_i| = 1$ and the above rules are inapplicable. Denote $|C_1| = 2$. Then branch on three cases: (i) $x = 1$, (ii) $x = 0$, $C_2 = 1$, (iii) $x = 0$, $C_2 = 0$, $C_3 = 1$, $D_1 = 0$. This gives at least a $(9,11,20)$-branching.

At this moment, $\sum |C_i| = 3$ and $1 \leq \sum |D_i| \leq 2$.

B-Rule 16. Let x be a $(3,2)$-variable such that $\sum |C_i| = 3$, and one of the following conditions hold:

1. $\sum |D_i| = 2$, and there's a variable in the union of Cs and Ds that is not a 5-variable appearing in this union once.
2. $\sum |D_i| = 1$, and in the union of Cs and Ds there's more than one 4^--variable.

Branching on x in this situation gives at least a $(6,10)$-branching.

B-Rule 17. Let x be a $(3,2)$-variable such that $\sum |C_i| = 3$ and the above rules are inapplicable. Let C_1 contain a literal of 5-variable. Then branch on three cases: (i) $x = 1$, (ii) $x = 0$, $C_1 = 1$, (iii) $x = 0$, $C_1 = 0$, $C_2 = C_3 = 1$, $D_1 = D_2 = 0$. This gives at least a $(8,11,24)$-branching.

Rules for 4-variables and Time Analysis

For $(2,2)$-variables we start with a rule similar to B-Rule 5.

B-Rule 18. Let x be a $(2,2)$-variable such that the above rules are inapplicable and x does not appear in unit clauses. Then branch on x. This gives at least a $(8,8)$-branching.

At this point, if there's a $(2,2)$-variable left in the formula, it necessarily appears in a unit clause. Without loss of generality, we consider $|D_2| = 0$. The next few reduction rules deal with some special cases of such variables.

R-Rule 13. Let x be a $(2,2)$-variable appearing in unit clauses. If C_1 and C_2 contain complementary literals, set $x = 0$.

In particular, a 3-variable can occur at most once in Cs: same appearances are prohibited by R-Rule 6 and complementary appearances are prohibited by R-Rule 13.

R-Rule 14. Let x be a $(2,2)$-variable appearing in a unit clause. Suppose $|D_1| = 1$. Denote $D_1 = l$. If l is a literal of a 3-variable, and l appears in either C_1 or C_2, set $x = 0$.

R-Rule 15. Let x be a $(2,2)$-variable appearing in a unit clause. Suppose $|D_1| = 1$. Denote $D_1 = l$. If l is a literal of a 3-variable, $\bar{l}$ appears in either C_1 or C_2, and $\bar{l}$ appears in F', set $x = l$.

R-Rule 16. Let x be a $(2,2)$-variable appearing in a unit clause. Suppose $|D_1| = 1$. Denote $D_1 = l$. If l is a literal of a 3-variable, $\bar{l}$ appears in either C_1 or C_2, and l appears in F', apply the following rule:

$$((l \vee E) \wedge (x \vee \bar{l} \vee C_1') \wedge (x \vee C_2) \wedge (\overline{x} \vee l) \wedge \overline{x} \wedge F'', k) \\ \rightarrow ((x \vee C_1' \vee E) \wedge (x \vee C_2) \wedge \overline{x} \wedge F'', k-2)$$

After R-Rules 13 – 15, if D_1 contains a single literal of a 3-variable, this variable cannot appear in Cs.

B-Rule 19. Let x be a $(2,2)$-variable appearing in a unit clause. If the above rules are inapplicable, branch on two cases: (i) $x = 0$, (ii) $x = 1$, and, if $|D_1| = 1$, $D_1 = 1$. This gives at least a $(6,10)$-branching.

After this rule, any 4-variable left is a $(3,1)$-variable.

For $(3,1)$-variables we start with an accurate analysis of B-Rule 6.

Lemma 10. *B-Rule 6 gives at least a $(6,10)$-branching.*

The last is the case of $(3,1)$-variables with empty D. We start with another use of lemma 1.

B-Rule 20. Let x be a $(3,1)$-variable such that $|D| = 0$ and there exists such index i that $|C_i| = 1$. Then branch on two cases: (i) $x = 1$, (ii) $x = 0$, and, for each j such that $|C_j| = 1$, $C_j = 1$. This gives at least a $(6,10)$-branching.

Now, $|D| = 0$, and $|C_i| \geq 2$ for each i. For this case, we provide several rules that work when x has few neighbors.

Note that no variable can appear three times in Cs. Indeed, for 3-variables it is prohibited by R-Rule 6, three positive appearances of a $(3,1)$-variable are also prohibited by R-Rule 6, and negative appearances of $(3,1)$-variables are prohibited by B-Rule 6 enhanced by Lemma 10.

R-Rule 17. If x is a $(3,1)$-variable such that the above rules are inapplicable, and y is a 3-variable appearing in Cs twice, remove x from the clause with the positive appearance of y, i,e. apply the following rule: $((y \vee E) \wedge (x \vee y \vee C_1') \wedge (x \vee \overline{y} \vee C_2') \wedge (x \vee C_3) \wedge \overline{x} \vee F'', k) \rightarrow ((y \vee E) \wedge (y \vee C_1') \wedge (x \vee \overline{y} \vee C_2') \wedge (x \vee C_3) \wedge \overline{x} \vee F'', k)$

B-Rule 21. If x is a $(3,1)$-variable such that the above rules are inapplicable, branch on x. This gives at least a $(4,14)$-branching.

Theorem 2. MAXSAT *can be solved in $O^*(1.0927^d)$ time. Hence,* MAXSAT *can be solved in $O^*(1.0927^L)$ time.*

Conclusion

In the paper we have presented two algorithms for the MAXIMUM SATISFIABILITY problem beating the previous result of Bansal and Raman (Bansal and Raman 1999) both in terms of simplicity and running time. The improvement is achieved by using a new carefully chosen measure and usage of the measure-and-conquer approach. This novelty allows us to handle inputs 13.3% longer than before which is a significant improvement for the worst-case analysis. Our theoretical finding can bring new insights into designing practical MAXSAT solvers. For example, some solvers contain branch-and-bound techniques at their core. Our research justifies that it is meaningful to use more involved measures that assess instance difficulty than simple standard measures like the number of clauses or input length. That is the decision on which variable to branch is taken based on the decrease of some new discounted measures and not straightforward measure input length. Newly developed reduction and branching rules also might be useful in practice. Applicability of our theoretical findings needs further research.

Acknowledgements

Research presented in Section "Main Algorithm" is supported by RSCF grant 18-71-10042. Research presented in Section "Simple Algorithm" is supported by HSE University and JetBrains Research.

References

Bansal, N.; and Raman, V. 1999. Upper bounds for MaxSat: Further improved. In *International symposium on algorithms and computation*, 247–258. Springer.

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

Berg, J.; Saikko, P.; and Järvisalo, M. 2015. Improving the effectiveness of SAT-based preprocessing for MaxSAT. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*. Citeseer.

Bliznets, I.; and Golovnev, A. 2012. A new algorithm for parameterized MAX-SAT. In *International Symposium on Parameterized and Exact Computation*, 37–48. Springer.

Chen, J.; and Kanj, I. A. 2002. Improved exact algorithms for MAX-SAT. In *Latin American Symposium on Theoretical Informatics*, 341–355. Springer.

Chen, J.; and Kanj, I. A. 2004. Improved exact algorithms for Max-Sat. *Discrete Applied Mathematics* 142(1-3): 17–27.

Chen, J.; and Liu, Y. 2009. An Improved SAT Algorithm in Terms of Formula Length. In *Workshop on Algorithms and Data Structures*, 144–155. Springer.

Chen, J.; Xu, C.; and Wang, J. 2015. Dealing with 4-variables by resolution: an improved MaxSAT algorithm. In *Workshop on Algorithms and Data Structures*, 178–188. Springer.

Crowston, R.; Gutin, G.; Jones, M.; Raman, V.; Saurabh, S.; and Yeo, A. 2014. Fixed-parameter tractability of satisfying beyond the number of variables. *Algorithmica* 68(3): 739–757.

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

Goemans, M. X.; and Williamson, D. P. 1994. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics* 7(4): 656–666.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2019. RC2: An efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation* 11(1): 53–64.

Impagliazzo, R.; and Paturi, R. 2001. On the complexity of k-SAT. *Journal of Computer and System Sciences* 62(2): 367–375.

Kojevnikov, A.; and Kulikov, A. S. 2006. A new approach to proving upper bounds for MAX-2-SAT. In *SODA*, volume 6, 11–17.

Liu, S.; and De Melo, G. 2017. Should algorithms for random SAT and Max-SAT be different? In *31st AAAI Conference on Artificial Intelligence, AAAI 2017*.

Mahajan, M.; and Raman, V. 1999. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms* 31(2): 335–354.

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

Morgado, A.; Heras, F.; Liffiton, M.; Planes, J.; and Marques-Silva, J. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints* 18(4): 478–534.

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

Poloczek, M.; Schnitger, G.; Williamson, D. P.; and Van Zuylen, A. 2017. Greedy algorithms for the maximum satisfiability problem: Simple algorithms and inapproximability bounds. *SIAM Journal on Computing* 46(3): 1029–1061.

Williams, R. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348(2-3): 357–365.

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