Fast Algorithms for SAT with Bounded Occurrences of Variables

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

Abstract

We present fast algorithms for the general CNF satisfiability problem (SAT) with running-time bound $O^*({c_d}^n)$, where c_d is a function of the maximum occurrence d of variables (d can also be the average occurrence when each variable appears at least twice), and n is the number of variables in the input formula. Similar to SAT with bounded clause lengths, SAT with bounded occurrences of variables has also been extensively studied in the literature. Especially, the running-time bounds for small values of d, such as $d = 3$ and $d = 4$, have become bottlenecks for algorithms evaluated by the formula length L and other algorithms. In this paper, we show that SAT can be solved in time $O^*(1.1238^n)$ for $d = 3$ and $O^*(1.2628^n)$ for $d = 4$, improving the previous results $O^*(1.1279^n)$ and $O^*(1.2721^n)$ obtained by Wahlström (SAT 2005) nearly 20 years ago. For $d \geq 5$, we obtain a running time bound of $O^*(1.0641^{\overline{d}n})$, implying a bound of $O^*(1.0641^L)$ with respect to the formula length L, which is also a slight improvement over the previous bound.

1 Introduction

The Boolean Satisfiability Problem (SAT) is the problem of testing propositional satisfiability for formulas in conjunctive normal form (CNF). As the first proved NP-complete problem [Cook, 1971], it has become one of the most foundational problems in computer science, AI, and math. SAT also has great applications in engineering. For these reasons, SAT and its variants are studied in a wide range of fields, such as heuristic algorithms, randomized algorithms, approximation algorithms, exact and parameterized algorithms, etc.

In exact algorithms, we are devoted to finding fast algorithms in which the correctness and the worst-case running-time bound are theoretically guaranteed. These running-time bounds are beneficial to understand the computational complexity of NP-complete problems and the limitation of solving these problems. Also, the methodology behind exact algorithms can make an impact on practical solvers.

In this paper, we study exact algorithms for SAT, where one is asked to decide the satisfiability of a given CNF with n variables. For the general case where there are no restrictions on the input CNF, the trivial algorithm to enumerate all possible assignments runs in time $O^*(2^n)$. The first non-trivial bound $O^*(2^{n(1-1/\alpha)})$ where $\alpha = \sqrt{n \log n}/2$ was obtained in [Dantsin *et al.*, 2004]. Later better bounds were achieved in [Schuler, 2005] and [Dantsin *et al.*, 2006]. Despite decades of research, no algorithm in time $O^*(c^n)$ for some constant $c < 2$ was found. The Strong Exponential Time Hypothesis (SETH) [Impagliazzo and Paturi, 2001] conjectures that such an algorithm does not exist. There has also been considerable research on two other natural measures: the number of clauses in the formula m and the length of the formula L. There is a series of improvements in the running-time bounds measured by m and L, respectively. Recently, the results were improved to $O^*(1.2226^m)$ [Chu *et al.*, 2021] and $O^*(1.0646^L)$ [Peng and Xiao, 2021].

Better bounds of $O^*(c^n)$ with $c < 2$ can be achieved for various restricted versions of SAT. One notable example is k-SAT, where every clause in the CNF is restricted to contain at most k literals. This problem is polynomial-time solvable for $k = 2$ and NP-complete for $k \geq 3$. Regarding k-SAT algorithms, two prevailing paradigms are local search algorithms, exemplified by Schöning's algorithm [Schöning, 1999], and random restriction algorithms, such as PPZ [Paturi *et al.*, 1999] and PPSZ [Paturi *et al.*, 2005]. Both paradigms have a running time of $2^{n(1-c/k+o(1/k))}$, where c is specific to the algorithm. The study of 3-SAT is of independent interest and has a rich history of improvements. Currently, the fastest known algorithms for 3-SAT include a deterministic one with time complexity $O^*(1.32793^n)$ [Liu, 2018] and a probabilistic one with time complexity $O^*(1.30698^n)$ [Scheder, 2021].

Another restriction is limiting the occurrence of each variable in the formula to at most d, which is the main focus of this paper. This problem can be solved in linear time when $d = 2$ and becomes NP-complete when $d \geq 3$ [Tovey, 1984]. Oliver and Luckhardt [1997] obtained a non-trivial bound $O^*(3^{n/9}) \subseteq O^*(1.1299^n)$ for the case $d = 3$. Subsequently, Wahlström [2005b] gave an improved $O^*(1.1279^{(d-2)n})$-time algorithm for $d \geq 3$. For $d = 3$ and 4, the algorithm runs in $O^*(1.1279^n)$ and $O^*(1.2721^n)$ time, respectively. These results are the best known for $d = 3$ and 4, and they have been utilized as sub-algorithms in several algorithms measured by the formula length L [Wahlström, 2005a; Chen and Liu, 2009; Peng and Xiao, 2021]. Specifically, they invoked Wahlström's algorithm when the maximum occur-

$d=3$	$d=4$	References
$O^*(1.1299^n)$	-	[Oliver and Luckhardt, 1997]
$O^*(1.1279^n)$	$O^*(1.2721^n)$	[Wahlström, 2005b]
$O^*(\mathbf{1.1238}^n)$	$O^*(\mathbf{1.2628}^n)$	**This paper**

Table 1: A summary of known results for SAT with $d = 3$ or 4.

rence of variables is 3 or 4. Consequently, their focus narrows down to scenarios where there is a variable with at least 5 occurrences. We also remark that the cases $d = 3$ and 4 have become the bottleneck cases in these previous algorithms.

Our Contribution. In this paper, we show that SAT with each variable appearing at least twice in the formula can be solved in time $O^*(1.1238^{(d-2)n})$ or $O^*(1.0641^{dn})$, where d can be the average occurrences of variables in the formula. The first running-time bound implies the results of $O^*(1.1238^n)$ for $d = 3$ and $O^*(1.2628^n)$ for $d = 4$, which are the first improvements from Wahlström's results. Note that $dn \geq L$. Our second running-time bound implies that SAT can be solved in time $O^*(1.0641^L)$, slightly improving the previous result $O^*(1.0646^L)$ [Peng and Xiao, 2021]. We consider our results for the cases $d = 3$ and 4 as the major contribution since our improvements for $d \geq 5$ are also based on those for the cases $d = 3$ and 4. We summarize the previous and our results for $d = 3$ and 4 in Table 1.

Our algorithms follow the branch-and-search approach, similar to previous algorithms. In these algorithms, variables are typically dealt in descending order of their degrees, where the *degree* of a variable refers to the number of its occurrences in the formula. When the degree of a variable is large enough, we can get a good branch directly. On the other hand, a variable with degree at most 2 can be easily handled by reduction rules. The most challenging cases usually involve variables with degrees 3 and 4. In the previous algorithm for $d = 3$ and 4 [Wahlström, 2005b], the worst branch will generate a branching vector of $(4, 8)$ (branching into two sub-branches: one reduces 4 degree-3 variables and one reduces 8 degree-3 variables), the corresponding complexity factor is 1.1279. In this paper, we improve the worst branching vector to $(5, 7)$ (the corresponding complexity factor is 1.1238) through several techniques: **(1)** introducing new reduction rules, such as R-Rule 8 and R-Rules 12-14; **(2)** replacing some degree-4 variables with degree-3 variables, simplifying the handling of degree-4 case and avoiding one previous bottleneck; **(3)** using refined analyses to avoid previous bottlenecks in dealing with degree-3 variables and the remaining degree-4 variables.

For $d \geq 5$, we adopt a general analytical framework to simplify the analysis, where we assign varying weights for variables based on their degrees to capture their contributions to the complexity. A key technique in this framework is the introduction of a *tunable parameter* in the weights, which allows us to derive different running-time bounds for the same algorithm by adjusting its value. Specifically, our dervied bound takes the form of $O^*(c_\lambda^{(d-\lambda)n})$, where λ is the tunable parameter, and c_λ is a constant related to λ. For $d = 3$ and 4, the bound will be better by setting $\lambda = 2$, and for $d \geq 5$, the bound will be better by setting $\lambda = 0$.

Other Related Work. We also mention the maximum satisfiability problem (MaxSAT), which is closely related to SAT. In MaxSAT, the objective is to satisfy the maximum number of clauses in a given CNF formula. The three parameters n, m, and L mentioned above were also considered for MaxSAT. For the number of variables n, the trivial bound $O^*(2^n)$ is also unlikely to break under SETH. For the number of clauses m and the length of the formula L, the bounds were recently improved to $O^*(1.2886^m)$ [Xiao, 2022] and $O^*(1.0926^L)$ [Alferov and Bliznets, 2021], respectively. There is also a series of results on the restricted version that the occurrence of each variable in the input CNF is at most $d = 3$. Since the first algorithm with running-time bound $O^*(1.732^n)$ [Raman *et al.*, 1998] was proposed, the results were frequently improved: $O^*(1.3248^n)$ [Bansal and Raman, 1999], $O^*(1.27203^n)$ [Kulikov and Kutskov, 2009], $O^*(1.2600^n)$ [Bliznets, 2013], $O^*(1.237^n)$ [Xu *et al.*, 2019], $O^*(1.194^n)$ [Xu *et al.*, 2021], and $O^*(1.191^n)$ [Belova and Bliznets, 2020]. Very recently, Brilliantov *et al.* [2023] improved the result to $O^*(1.1749^n)$. Based on this result, they can improve the result evaluated by L to $O^*(1.0911^L)$.

The proofs of lemmas and theorems marked with ♣ can be found in the full version of this paper.

2 Preliminaries

2.1 Notations

For a boolean variable x, it has two corresponding *literals*: the positive literal x and the negative literal $\overline{x}$. We use $\overline{x}$ to denote the negation of a literal x and thus $\overline{\overline{x}} = x$. Given a set of variables V, a *clause* on V is a subset of literals on V, and a *formula* is set of clauses. An assignment for V is a map $A : V \to \{0, 1\}$. A clause is *satisfied* by an assignment if at least one literal in it gets value 1, and a formula is *satisfied* by an assignment if all clauses in it are satisfied. A formula is *satisfiable* if it can be satisfied by at least one assignment.

A clause containing a single literal x may be simply written as x. We use C_1C_2 to denote the clause containing all literals in clauses C_1 and C_2. For a clause C, we use $\overline{C}$ to denote the set of negations of literals in C, and use $C = 1$ (resp., $C = 0$) to indicate that we assign 1 (resp., 0) to all literals in C. For a formula $\mathcal{F}$, we denote $\mathcal{F}_{C=1}$ as the resulting formula obtained from $\mathcal{F}$ by removing all clauses containing some literal in C and removing all literals in $\overline{C}$ from all clauses in $\mathcal{F}$, and let $\mathcal{F}_{C=0} = \mathcal{F}_{\overline{C}=1}$.

In a formula $\mathcal{F}$, a literal z is called an (i, j)-*literal* (resp., an (i^+, j)-*literal* or (i^-, j)-*literal*) if z appears i (resp., at least i or at most i) times and $\overline{z}$ appears j times in $\mathcal{F}$. Similarly, we can define (i, j^+)-literal, (i, j^-)-literal, etc. A literal z is *pure* if z is an $(1^+, 0)$-literal. A variable x *appears* in a formula or a set of literals C if x or $\overline{x}$ is contained in C. For a variable or a literal x in formula $\mathcal{F}$, the *degree* of it, denoted by $deg(x)$, is the occurrences of x and $\overline{x}$ in $\mathcal{F}$, i.e., $deg(x) = i + j$ for an (i, j)-literal x or $\overline{x}$. A variable x is a d-*variable* (resp., d^+-*variable*) if $deg(x) = d$ (resp., $deg(x) \geq d$). The degree of a formula $\mathcal{F}$, denoted by $deg(\mathcal{F})$, is the maximum degree of all variables in $\mathcal{F}$. For a clause or a formula C, the set of variables appearing in C is denoted by $var(C)$. A formula $\mathcal{F}$ is *d-regular* if all variables in $\mathcal{F}$ are d-variables.

The *length* of a clause C, denoted by $|C|$, is the number of literals in C. A clause is a *k-clause* (resp., *k^+-clause*) if the length of it is k (resp., at least k). A formula $\mathcal{F}$ is *k-CNF* if each clause in $\mathcal{F}$ has a length of at most k. A clause is a *P-monotone* (resp., *N-monotone*) clause if it contains only positive (resp., negative) literals. A clause is a *monotone* clause if it is P-monotone or N-monotone. Otherwise, the clause is *non-monotone*. A monotone formula is a formula where each clause in it is monotone.

In a formula $\mathcal{F}$, a literal x is called a *neighbor* of a literal z if there is a clause containing both z and x. The set of neighbors of a literal z in a formula $\mathcal{F}$ is denoted by $N(z, \mathcal{F})$. We also use $N^{(k)}(x, \mathcal{F})$ (resp., $N^{(k^+)}(z, \mathcal{F})$) to denote the neighbors of z in k-clauses (resp., k^+-clauses) in $\mathcal{F}$, i.e., for any $z' \in N^{(k)}(z, \mathcal{F})$ (resp., $z' \in N^{(k^+)}(z, \mathcal{F})$), there exists a k-clause (resp., k^+-clause) containing both z and z'.

Resolution [Davis and Putnam, 1960] is a frequently used technique for SAT. For two clauses xC and $\overline{x}D$ in formula $\mathcal{F}$, the *resolvent* of C and D by variable x is the clause CD. The resolvent CD is *trivial* if it contains both v and $\overline{v}$ for some literal v, or there is another clause E in $\mathcal{F}$ such that $E \subseteq CD$. We assume that duplicate literals in a resolvent are removed. A *resolution* on x in formula $\mathcal{F}$ is to construct a new formula $DP_x(\mathcal{F})$ by adding all non-trivial resolvents by x to $\mathcal{F}$ and removing all clauses containing variable x.

We assume that in the initial formula, each variable appears at least twice, although 1-variables may be generated during the algorithm. Note that 1-variables can be easily reduced but they may affect the average degree of the formula.

2.2 Branch-and-Search Algorithms

Our algorithm is a classical branch-and-search algorithm, which first applies reduction rules to reduce the instance and then searches for a solution by branching. We need to use a measure to evaluate the size of the search tree generated in the algorithm. Let μ be the measure and $T(\mu)$ be an upper bound on the size of the search tree generated by the algorithm on any instance with the measure of at most μ. A branching operation, which branches on the instance into l branches with the measure decreasing by at least a_i in the i-th branch, is usually represented by a recurrence relation

$$T(\mu) \leq T(\mu - a_1) + T(\mu - a_2) + \cdots + T(\mu - a_l),$$

or simply by a *branching vector* $[a_1, a_2, \ldots, a_l]$. The largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-a_i}$ is called the *branching factor* of the recurrence. If the maximum branching factor for all branching operations in the algorithm is at most γ, then $T(\mu) = O(\gamma^{\mu})$. More details about analyzing branching algorithms can be found in [Fomin and Kratsch, 2010]. We say that one branching vector is *not worse* than another if its corresponding branching factor is not greater than that of the latter. We need to find the worst branching vector with the largest branching factor in the algorithm. We will frequently use the following property, which can be obtained from Lemmas 2.2 and 2.3 in [Fomin and Kratsch, 2010].

Lemma 1. *The branching vector $[\Delta_0, \Delta_1]$ is not worse than $[q, p-q]$ if $\Delta_0 + \Delta_1 \geq p$ and $\min(\Delta_0, \Delta_1) \geq q$.*

2.3 The Measure

We use the measure-and-conquer method [Fomin *et al.*, 2009] to analyze the running-time bound. Specifically, we introduce a weight to each variable according to its degree. Let $w : \mathbb{Z}^+ \to \mathbb{R}^+$. We adopt the following measure of a formula $\mathcal{F}$:

$$\mu(\mathcal{F}) = \sum_{v \in var(\mathcal{F})} w_{deg(v)}.$$

Let n_i be the number of variables with a degree of i in the formula, L be the length of the formula, and $d = L/n$ be the average occurrence of variables in the formula. We also introduce a *tunable parameter* $\lambda \geq 0$ and restrict that $w_i \leq i - \lambda$ for $i > 1$. The tunable parameter is important. For the same algorithm, we will get different running-time bounds under different values of the tunable parameter. Recalling our assumption that there are no 1-variables in the initial formula, thus $n_1 = 0$ holds. For the initial formula $\mathcal{F}$, we have

$$\begin{aligned}\mu(\mathcal{F}) = \sum_i w_i n_i \leq \sum_{i>1} (i - \lambda) \cdot n_i &= \sum_{i>1} i \cdot n_i - \lambda \sum_{i>1} n_i \\ &= L - \lambda n = (d - \lambda) n.\end{aligned}$$

If we get a running-time bound of $O^*(c^{\mu(\mathcal{F})})$, then we also get a bound of $O^*(c^{(d-\lambda)n})$. When $\lambda = 0$, the running-time bound will be $O^*(c^{dn}) = O^*(c^L)$.

Let $\delta_i = w_i - w_{i-1}$ for $i > 1$. To guarantee that the measure does not increase during the algorithm and to simplify certain arguments, we impose some constraints on w_i and λ:

$$\begin{aligned} w_{i-1} \leq w_i \leq i - \lambda \text{ for } i > 1, \delta_{i-1} \geq \delta_i \text{ for } i > 3, \\ w_1 = w_2 = 0, w_3 < 2, w_4 = 2w_3, \\ w_i = i - \lambda \text{ for } i \geq 5, \text{ and } \lambda \leq 2. \end{aligned} \tag{1}$$

Based on the above constraints, we have

$$2\delta_5 > w_3 = \delta_3 = \delta_4 \geq \delta_5 \geq 1 = \delta_i \text{ for } i \geq 6. \tag{2}$$

With these constraints, we can see that once we determine the value of λ and w_3, then we can determine all weights.

3 Reduction Rules

We have 15 reduction rules. When we introduce one we assume that all previous rules can not be applied.

R-Rule 1. $\mathcal{F}' \wedge xxC \to \mathcal{F}' \wedge xC$.

R-Rule 2. $\mathcal{F}' \wedge C \wedge CD \to \mathcal{F}' \wedge C$.

R-Rule 3. $\mathcal{F}' \wedge x\overline{x}C \to \mathcal{F}'$.

R-Rule 4. *If there is a 1-clause x or a $(1^+, 0)$-literal x, then $\mathcal{F} \to \mathcal{F}_{x=1}$.*

R-Rule 5. *If there is a variable x with $deg(x) \leq 4$ such that the degree of each variable in $DP_x(\mathcal{F})$ is not greater than that in $\mathcal{F}$, then $\mathcal{F} \to DP_x(\mathcal{F})$.*

When none of the above rules is applicable, all variables in the formula are 3^+-variables. R-Rules 6 to 9 are going to deal with some pairs of clauses containing two common variables.

R-Rule 6 ([Chen and Liu, 2009])**.** *If there is a 2-clause xy and another clause $x\overline{y}C$, then $\mathcal{F}' \wedge xy \wedge x\overline{y}C \to \mathcal{F}' \wedge xy \wedge xC$.*

R-Rule 7 ([Peng and Xiao, 2021]). *Let x be a $(2^+,1)$-literal. If there are clauses xyC_1 and $\overline{x}yC_2$, then $\mathcal{F}' \wedge xyC_1 \wedge \overline{x}yC_2 \to \mathcal{F}' \wedge xC_1 \wedge \overline{x}yC_2$.*

R-Rule 8 (♣). *Let x be a $(2^+,1)$-literal. If there are clauses xyC_1 and $\overline{xy}C_2$, then $\mathcal{F}' \wedge xyC_1 \wedge \overline{xy}C_2 \to \mathcal{F}' \wedge \overline{xy}C_2$.*

R-Rule 9 ([Chen and Liu, 2009]). *If there are two clauses CD_1 and CD_2 such that $|C| \geq 2$ and $|D_1|, |D_2| \geq 1$, then $\mathcal{F}' \wedge CD_1 \wedge CD_2 \to \mathcal{F}' \wedge xC \wedge \overline{x}D_1 \wedge \overline{x}D_2$, where x is a new 3-variable.*

R-Rule 10 ([Chen and Liu, 2009]). *If there is a 2-clause xy such that x and y are not both $(1,2)$-literals and x is an $(1,2^+)$-literal, then replace x with $\overline{y}$ and apply R-Rule 3.*

R-Rule 11. *If there is a $(2,2)$-literal x and four clauses $xC_1, xC_2, \overline{x}D_1$, and $\overline{x}D_2$, then $\mathcal{F}' \wedge xC_1 \wedge xC_2 \wedge \overline{x}D_1 \wedge \overline{x}D_2 \to \mathcal{F}' \wedge y_1C_1 \wedge y_1C_2 \wedge y_2D_1 \wedge y_2D_2 \wedge \overline{y_1y_2}$, where y_1 and y_2 are two new $(2,1)$-literals.*

R-Rule 11 is like the backward version of R-Rule 10. Note that our setting can avoid endless loops between them.

R-Rule 12 (♣). *Let x, y_1, $\cdots$, y_k ($k \geq 2$) be $(2^+,1)$-literals. If there are clauses xy_1C_1, $\overline{y_1}y_2D_1$, $\cdots$, $\overline{y_i}y_{i+1}D_i$, $\cdots$, $\overline{y_{k-1}}y_kD_{k-1}$, and $\overline{xy_k}C_2$, then remove clause xy_1C_1.*

R-Rule 13 (♣). *Let x be a $(2,1)$-literal. If there are clauses $\overline{xy}C_1, xaC_2, xC_3$, and D such that $|C_1| \leq 1$, and $D = ya$ or $D = \overline{ya}C_4$ and literal a is a $(2,1)$-literal, then $\mathcal{F}' \wedge \overline{xy}C_1 \wedge xaC_2 \wedge D \wedge xC_3 \to \mathcal{F}' \wedge aC_1C_2 \wedge \overline{y}C_1C_3 \wedge D$.*

R-Rule 14 (♣). *Let x, y, z, and a be $(2,1)$-literals. If there are clauses $\overline{xy}, xa, y\overline{z}b, zaC$, and zD, then $\mathcal{F}' \wedge \overline{xy} \wedge xa \wedge y\overline{z}b \wedge zaC \wedge zD \to \mathcal{F}' \wedge abC \wedge \overline{x}bD \wedge xa$.*

R-Rule 15. *If there is a negative $(2,1)$-literal x, then replace x with $\overline{x}$ to make it a positive $(2,1)$-literal.*

Definition 1. *A formula is* reduced *if none of R-Rules 1–15 is applicable. We use $R(\mathcal{F})$ to denote the reduced formula obtained by iteratively applying R-Rules 1–15 on $\mathcal{F}$.*

Lemma 2 (♣). *For any formula $\mathcal{F}$, applying any reduction rule on it will not increase the measure μ. Moreover, it takes polynomial time to obtain $R(\mathcal{F})$ from $\mathcal{F}$.*

Lemma 3 (♣). *In a reduced formula, there are no $(i,0)$-literals and all variables are 3^+-variables.*

Lemma 4 (♣). *In a reduced formula, if there is a 2-clause xy, then no other clause contains xy, $\overline{x}y$, or $x\overline{y}$. Moreover, if x is a $(2^+,1)$-literal, then no other clause contains $\overline{xy}$.*

Lemma 5 (♣). *In a reduced formula, any two clauses contain at most one common 3-variable.*

Lemma 6 (♣). *In a reduced 3-regular formula, the two literals in any 2-clause are both negative $(1,2)$-literals or both positive $(2,1)$-literals.*

Lemma 7 (♣). *Let $C = \overline{xy}$ be a 2-clause that contains two negative $(1,2)$-literals in a reduced 3-regular formula. It holds that $var(N(x,\mathcal{F})) \cap var(N^{(2)}(y,\mathcal{F})) = \emptyset$.*

4 Main Algorithm

After obtaining a reduced formula by applying reduction rules, we apply branching rules. We have six branching steps.

Steps	Branching vectors	Factors $\lambda = 2$, $w_3 = 1$	Factors $\lambda = 0$, $w_3 = 1.87884$
1	(w_6+1, w_6+11)	1.0795	1.0636
2.1	$(w_5+2w_3, w_5+w_3+7\delta_5)$	1.0816	1.0604
2.2	$(w_5+2\delta_5, w_5+2w_3+6\delta_5)$	1.0956	1.0635
	$(w_5+4\delta_5, w_5+6\delta_5)$	1.0911	1.0641
2.3	$(5w_3, 7w_3)$	1.1238	1.0641
3	$(5w_3, 7w_3)$	1.1238	1.0641
4	$(6w_3, 6w_3)$	1.1224	1.0635
5.1	$(6w_3, 6w_3)$	1.1224	1.0635
5.2	$(5w_3, 7w_3)$	1.1238	1.0641
6	$1.1092^{\frac{1}{w_3}}$	1.1092	1.0568

Table 2: The branching vector of each step in the algorithm and the corresponding branching factors under $\lambda = 0$ & $w_3 = 1$ and $\lambda = 2$ & $w_3 = 1.87884$.

The worst cases of the branching vectors in each step are listed in Table 2. Recall that we have a tunable parameter λ in the weights. We will use the results for $\lambda = 2$ and 0. The responding branching factors are also given in Table 2.

Before going into the detailed branching steps, we first introduce some lower bounds on the decrease of measure in two frequently-used branchings. We use the following notations.

Definition 2. *For a literal x in a formula $\mathcal{F}$, we define*

- $p_i(x) = |\{y : y \in N(x,\mathcal{F}) \text{ and } deg(y) = i\}|$ *and*
- $q_i(x) = |\{y : y \in N^{(2)}(x,\mathcal{F}) \text{ and } deg(y) = i\}|$.

In other words, $p_i(x)$ is the number of literals with a degree of i in $N(x,\mathcal{F})$, and $q_i(x)$ is the number of literals with a degree of i in $N^{(2)}(x,\mathcal{F})$.

The following two lemmas can be easily obtained from Lemmas 6 and 9 in [Peng and Xiao, 2021]. We remark that the conditions of the two lemmas also hold in our algorithm.

Lemma 8 (♣). *Let x be a literal of a d-variable in a reduced formula $\mathcal{F}$ where $deg(\mathcal{F}) = d$. It holds that $\Delta = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1})) \geq w_d + \sum_{i=3}^{d} p_i(x)\delta_d$.*

Lemma 9 (♣). *Let x be a literal of a d-variable in a reduced formula $\mathcal{F}$ such that $deg(\mathcal{F}) = d$, and $\Delta_1 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1}))$ and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0}))$. It holds that $\Delta_1 + \Delta_0 \geq 2w_d + 2d\delta_d + \sum_{i=3}^{d}(q_i(x) + q_i(\overline{x}))(2w_3 - 2\delta_d)$.*

The following lemma will be frequently used.

Lemma 10 (♣). *Let x be a $(d-1,1)$-literal in a reduced formula $\mathcal{F}$ with $d = deg(\mathcal{F}) \geq 4$. Assume that the clauses containing x or $\overline{x}$ are $xC_1, \cdots, xC_{d-1}$ and $\overline{x}D$. It is safe to branch with (1) $x = 1$; (2) $x = 0$ and $D = 0$. Let $\Delta_1 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1}))$ and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0\&D=0}))$. It holds that $\Delta_1 + \Delta_0 \geq 2w_d + (2d-3)\delta_d + 3w_3$ and $\min(\Delta_1, \Delta_0) \geq w_d + \min((d-1)\delta_d, 2w_3 + \delta_d)$.*

4.1 Step 1: 6^+-variables

If there is a 6^+-variable in the formula, we pick up a variable x of maximum degree and branch with (1) $x = 1$; (2) $x = 0$.

We show that the branching vector is not worse than $(w_6 + 1, w_6+11)$. Let $d = deg(x) \geq 6$, $\Delta_1 = \mu(\mathcal{F})-\mu(R(\mathcal{F}_{x=1}))$, and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0}))$. Note that $\delta_d = \delta_6 = 1$ since $d \geq 6$. By Lemma 9, we have $\Delta_1 + \Delta_0 \geq 2w_d + 2d\delta_d \geq 2w_6 + 12$. By Lemma 8, we have $\Delta_1 \geq w_d + \sum_{i=3}^{d} p_i(x)\delta_d \geq w_6+1$ since $\sum_{i=3}^{d} p_i(x) \geq 1$. Similarly, we have $\Delta_0 \geq w_6 + 1$. Thus, $\min(\Delta_1, \Delta_0) \geq w_6 + 1$. Together with $\Delta_1+\Delta_0 \geq 2w_6+12$, we know that the branching vector is not worse than $(w_6 + 1, w_6 + 11)$ by Lemma 1.

4.2 Step 2: 4-variables and 5-variables

In this step, we deal with 4-variables and 5-variables. Note that all $(2, 2)$-literals are reduced by R-Rule 11.

Step 2.1: If there is a $(4, 1)$-literal x, where $\overline{x}D$ is assumed to be the unique clause containing $\overline{x}$, then branch with (1) $x = 0$ and $D = 0$; (2) $x = 1$. Let $\Delta_1 = \mu(\mathcal{F})-\mu(R(\mathcal{F}_{x=1}))$ and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0\&D=0}))$. By Lemma 10, we know that $\Delta_1+\Delta_0 \geq 2w_5+7\delta_5+3w_3$ and $\min(\Delta_1, \Delta_0) \geq w_5+\min(4\delta_5, 2w_3+\delta_5) \geq w_5+2w_3$ since $w_3 < 2\delta_5$. Thus, the branching vector is not worse than $(w_5+2w_3, w_5+w_3+7\delta_5)$ by Lemma 1.

Step 2.2: If there is a $(2, 3)$-literal x, then branch with (1) $x = 1$; (2) $x = 0$. Let $\Delta_1 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1}))$ and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0}))$. We consider two cases.

Case 1: There is at least one 2-clause containing x or $\overline{x}$, i.e., $\sum_{i=3}^{5}(q_i(x) + q_i(\overline{x})) \geq 1$. By Lemma 9, we have $\Delta_1+\Delta_0 \geq 2w_5+10\delta_5+\sum_{i=3}^{5}(q_i(x)+q_i(\overline{x}))(2w_3-2\delta_5) \geq 2w_5 + 2w_3 + 8\delta_5$. By Lemma 8, we have $\Delta_1 \geq w_5 + \sum_{i=3}^{5} p_i(x)\delta_5 \geq w_5 + 2\delta_5$ and $\Delta_0 \geq w_5 + \sum_{i=3}^{5} p_i(\overline{x})\delta_5 \geq w_5 + 3\delta_5$ since x is a $(2, 3)$-literal. Thus $\min(\Delta_1, \Delta_0) \geq w_5 + 2\delta_5$. Together with $\Delta_0 + \Delta_1 \geq 2w_5 + 2w_3 + 8\delta_5$, we know that the branching vector is not worse than $(w_5 + 2\delta_5, w_5 + 2w_3 + 6\delta_5)$ by Lemma 1.

Case 2: All clauses containing x or $\overline{x}$ are 3^+-clauses. Then we have $\sum_{i=3}^{5} p_i(x) \geq 4$ and $\sum_{i=3}^{5} p_i(\overline{x}) \geq 6$. It follows that $\Delta_1 \geq w_5 + 4\delta_5$ and $\Delta_0 \geq w_5 + 6\delta_5$ by Lemma 8. Thus, the branching vector is not worse than $(w_5 + 4\delta_5, w_5 + 6\delta_5)$.

Step 2.3: If there is a $(3, 1)$-literal x, where $\overline{x}D$ is assumed to be the unique clause containing $\overline{x}$, then we branch with (1) $x = 0$ and $D = 0$; (2) $x = 1$. The branching vector is not worse than $(5w_3, 7w_3)$. Let $\Delta_1 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1}))$ and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0\&D=0}))$. By Lemma 10 we know that $\Delta_1 + \Delta_0 \geq 2w_4 + 5\delta_4 + 3w_3 = 12w_3$ and $\min(\Delta_1, \Delta_0) \geq w_4 + \min(3\delta_4, 2w_3 + \delta_4) = 5w_3$ since $w_4 = 2w_3$ and $\delta_4 = w_3$. Thus, the branching vector is not worse than $(5w_3, 7w_3)$ by Lemma 1.

4.3 3-regular Formula (Steps 3–6)

Now the formula is a reduced 3-regular formula. Moreover, all positive literals appear twice and all negative literals appear once in the formula since R-Rule 15 is not applicable. The removal of any literal of a 3-variable will make it a 2-variable, and reduction rules will reduce the 2-variable. So when a literal of a 3-variable is removed, we will regard this variable as reduced. This property will be frequently used. We have four steps to deal with different cases.

Step 3: Good N-monotone 2-clauses

Assume that there is an N-monotone 2-clause $\overline{xy}$. There is no other clause that contains both x and y by Lemma 5. Let the four clauses containing x and y be xC_1, xC_2, yC_3, and yC_4, where we also assume, w.l.o.g., that $|C_1| \leq |C_2|$, $|C_3| \leq |C_4|$, and $|C_1| \leq |C_3|$.

An N-monotone 2-clause $\overline{xy}$ is called *good* if it holds at least one of the following conditions: (i) $|C_1| \geq 2$; (ii) $|C_1| = |C_3| = 1$; (iii) $|C_1| = 1$ and $|C_2|, |C_3| \geq 2$; (iv) $|C_1| = |C_2| = 1$, $|C_3| = 2$, and $|C_4| \geq 3$; (v) There is a negative $(1, 2)$-literal in $N(x, \mathcal{F})$ or $N(y, \mathcal{F})$. Thus, if $\overline{xy}$ is not good, then it holds that $|C_1| = |C_2| = 1$, $|C_3| = |C_4| = 2$, and all clauses containing x or y are P-monotone.

This step is going to deal with good N-monotone 2-clauses if they exist. **Let $\overline{xy}$ be a good N-monotone 2-clause. We branch with (1) $x = 1$ and $y = 0$; (2) $x = 0$ and $y = 1$.**

Lemma 11. *The branching vector generated by Step 3 is not worse than* $(5w_3, 7w_3)$.

Proof. We denote $\Delta_1 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1\&y=0}))$ and $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0\&y=1}))$. Let t_x and t_y be the number of 2-clauses containing literal x and literal y, respectively. In the branch $x = 1$ and $y = 0$, all variables in $var(C_1)$ and $var(C_2)$ are reduced, and all literals in $N^{(2)}(y, \mathcal{F})$ will get assignment by R-Rule 4 since the clauses containing them become 1-clauses. By Lemma 5, we have $\{x, y\} \cap (var(C_1C_2) \cup var(C_3C_4)) = \emptyset$. By Lemma 7, we know that $var(C_1C_2) \cap var(N^{(2)}(y, \mathcal{F})) = \emptyset$. So the number of reduced variables is at least $|\{x, y\}| + |var(C_1C_2)| + |var(N^{(2)}(y, \mathcal{F}))| = 2 + |C_1| + |C_2| + t_y$, and then we have

$$\Delta_1 \geq (2 + |C_1| + |C_2| + t_y)w_3.$$

Similarly, in the branch $x = 0$ and $y = 1$, we have

$$\Delta_0 \geq (2 + |C_3| + |C_4| + t_x)w_3.$$

Thus, it holds that $\Delta_0+\Delta_1 \geq (4+\sum_{i=1}^{4} |C_i|+t_x+t_y)w_3 \geq (4 + \sum_{i=1}^{4} \max(|C_i|, 2))w_3 \geq (4 + 8)w_3 = 12w_3$ since that for each C_i, it contributes to $\Delta_0 + \Delta_1$ at least $2w_3$ when $|C_i| = 1$ and at least $|C_i|w_3$ when $|C_i| \geq 2$.

We first claim that if at least one of conditions (i)–(iii) is satisfied, then we have $\min(\Delta_0, \Delta_1) \geq 5w_3$, which is sufficient to get a branching vector not worse than $(5w_3, 7w_3)$ since $\Delta_0 + \Delta_1 \geq 12w_3$ by Lemma 1. If condition (i) is satisfied, we have $|C_2| \geq |C_1| \geq 2$ and $|C_4| \geq |C_3| \geq |C_1| \geq 2$. It holds that $\Delta_1 \geq (2+|C_1|+|C_2|)w_3 \geq 6w_3$ and $\Delta_0 \geq (2+|C_3| + |C_4|)w_3 \geq 6w_3$. Thus, we have $\min(\Delta_0, \Delta_1) \geq 6w_3$. If condition (ii) is satisfied, we have $t_x \geq 1$ and $t_y \geq 1$. It holds that $\Delta_1 \geq (2 + |C_1| + |C_2| + t_y)w_3 \geq 5w_3$ and $\Delta_0 \geq (2 + |C_3| + |C_4| + t_x)w_3 \geq 5w_3$. And then we have $\min(\Delta_0, \Delta_1) \geq 5w_3$. If condition (iii) is satisfied, we have $|C_1| = 1$, $|C_2| \geq 2$, and $|C_4| \geq |C_3| \geq 2$. It holds that $\Delta_1 \geq (2 + |C_1| + |C_2|)w_3 \geq 5w_3$ and $\Delta_0 \geq (2 + |C_3| + |C_4|)w_3 \geq 6w_3$. Then we have $\min(\Delta_0, \Delta_1) \geq 5w_3$.

If condition (iv) is satisfied, we have $t_x = 2$. Then it holds that $\Delta_1 \geq (2 + |C_1| + |C_2|)w_3 = (2 + 1 + 1)w_3 = 4w_3$ and $\Delta_0 \geq (2 + |C_3| + |C_4| + t_x)w_3 \geq (2 + 2 + 3 + 2)w_3 = 9w_3$. Thus the branching vector is not worse than $(4w_3, 9w_3)$, which is better than $(5w_3, 7w_3)$.

Next, we assume that only condition (v) is satisfied. This will be the hardest case. We have $|C_1| = |C_2| = 1$, $|C_3| = |C_4| = 2$, and $t_x = 2$, and we show the branching vector is not worse than $(4w_3, 9w_3)$.

By Lemma 6 and $|C_1| = |C_2| = 1$, we know that both xC_1 and xC_2 are P-monotone, and thus we can assume, w.l.o.g, that $C_3 = \overline{z}\alpha$, where $\overline{z}$ is a negative $(1,2)$-literal and α is an $(1,2)/(2,1)$-literal by the condition (v). In the branch $x = 1$ and $y = 0$, we have $\Delta_1 \geq (2+|C_1|+|C_2|)w_3 = 4w_3$. In the branch $x = 0$ and $y = 1$, based on the previous analysis, we can first get $\Delta_0 \geq (2 + |C_3| + |C_4| + t_x)w_3 = 8w_3$, where only variables in $S = \{x, y\} \cup var(C_1C_2) \cup var(C_3C_4)$ were taken into consideration. We show that we can further reduce one more variable besides S to get $\Delta_0 \geq 9w_3$.

After assigning $x = 0$ and $y = 1$, literal z becomes pure. We can assign $z = 1$ by R-Rule 4 and further reduce $var(N(z, \mathcal{F}))$. Let zE_1 and zE_2 be the two clauses containing litearl z, i.e., $N(z, \mathcal{F}) = E_1E_2$. By Lemma 7, we have $x \notin E_1E_2$. By Lemma 5, we have $y \notin E_1E_2$ and $var(E_1E_2) \cap var(C_3) = \emptyset$. Let v be a positive $(2,1)$-literal in C_1C_2. We can apply R-Rule 14 if $v \in E_1E_2$ and apply R-Rule 12 in the case of $k = 3$ if $\overline{v} \in E_1E_2$. So we have $var(C_1C_2) \cap var(E_1E_2) = \emptyset$. If $|var(E_1E_2) \setminus var(C_4)| \geq 1$, then we can reduce at least $|S \cup var(N_z)| = |\{x, y\}| + |var(C_1C_2)| + |var(C_3C_4)| + |var(E_1E_2) \setminus var(C_4)| \geq 2+2+4+1 = 9$ variables. Next, we assume $var(E_1E_2) \subseteq var(C_4)$ and show that we still can further reduce one more variable besides S.

We have $|E_1| = |E_2| = 1$ since $var(E_1E_2) \subseteq var(C_4)$ and $|C_4| = 2$. Let $var(C_4) = \{a, b\}$ and assume, w.l.o.g., that $E_1 = a$ and $E_2 = b$ since that both zE_1 and zE_2 are P-monotone by Lemma 6. If both literals in C_4 are negative $(1,2)$-literals, then R-Rule 13 should be applicable. If there is only one negative $(1,2)$-literal in C_4, say $\overline{a}$, then $DP_a(\mathcal{F})$ satisfies the requirement of R-Rule 5. Thus, both literals in C_4 are positive $(2,1)$-literals, i.e., $C_4 = ab$. Let $\overline{a}G_1$ and $\overline{b}G_2$ be the clauses containing $\overline{a}$ and $\overline{b}$. We can reduce variables in $var(G_1G_2)$ by assigning $\overline{a} = \overline{b} = 1$ since they become pure after assigning $x = 0$, $y = 1$, and $z = 1$. Recall that $C_3 = \overline{z}\alpha$ and a is a positive $(2,1)$-literal contained in za. We can apply R-Rule 13 if $\alpha \in G_1G_2$ and R-Rule 12 in the case of $k = 2$ if $\overline{\alpha} \in G_1G_2$. With Lemma 5, we have $var(G_1G_2) \cap (\{x, y\} \cup var(C_3C_4)) = \emptyset$. If $|G_1| = 1$, then $\overline{a}G_1$ is an N-monotone 2-clause by Lemma 6, and it satisfies one of conditions (i)–(iv) since that there are clauses za and yab. Thus, we have $|G_1| \geq 2$, and similarly, $|G_2| \geq 2$. This together with Lemma 5 yields that $|var(G_1G_2)| \geq 3$. So we have $|var(G_1G_2) \setminus var(C_1C_2)| \geq 1$ since $|var(C_1C_2)| = 2$. Therefore, in the case $var(E_1E_2) \subseteq var(C_3C_4)$, we can reduce at least $|S \cup var(G_1G_2)| \geq |\{x, y\}| + |var(C_1C_2)| + |var(C_3C_4)| + var(G_1G_2) \setminus var(C_1C_2) \geq 2+2+4+1 = 9$ variables. This completes the proof. □

Step 4: Making the Formula Monotone

In this step, we pick up a positive literal x in a non-monotone clause C (if it exists) and branch on x. We will show that the branching vector is not worse than $(6w_3, 6w_3)$.

Lemma 12 (♣). *Let C be a non-monotone clause in a reduced formula after Step 3. If there is a positive literal $y \in C$, then $\overline{y}$ is contained in a 3^+-clause.*

Lemma 13 (♣). *Let $\mathcal{F}$ be a reduced formula after Step 3 and xD be a non-monotone clause in $\mathcal{F}$ containing a positive literal x. Then it holds that $\Delta = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1})) \geq 6w_3$.*

Lemma 14 (♣). *Let $\mathcal{F}$ be a reduced formula after Step 3 and $\overline{x}C'$ be a 3^+-clause in $\mathcal{F}$ containing a negative literal $\overline{x}$. It holds that $\Delta = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0\&C'=0})) \geq 6w_3$.*

Let $C = xD$ be a non-monotone clause containing positive literal x and $\overline{x}C'$ be the clause containing $\overline{x}$. In this step, we will branch with (1) $x = 1$; (2) $x = 0$ and $C' = 0$. Equipped with Lemmas 13 and 14, we can show the branching vector is not worse than $(6w_3, 6w_3)$. In the first branch, we can reduce the measure by at least $6w_3$ by Lemma 13. In the second branch, we have $|C'| \geq 2$ by Lemma 12 and then we can also reduce the measure by at least $6w_3$ by Lemma 14.

Step 5: Making the Formula 3-CNF

Now, the formula is monotone. In this step, we are going to deal with N-monotone clauses of length at least 4 and some N-monotone clauses of length 3.

Step 5.1: If there is an N-monotone clause $C = \overline{v_1v_2}\cdots\overline{v_k}$ with $k \geq 4$, then branch with (1) $\overline{v_1} = \cdots = \overline{v_{\lceil k/2 \rceil}} = 0$; (2) $\overline{v_{\lceil k/2 \rceil + 1}} = \cdots = \overline{v_k} = 0$. This branch is correct because all the literals in C are $(1,2)$-literals and we only need to let one of them be 1 and all others 0. We show that the branching vector is not worse that $(6w_3, 6w_3)$. The proof is based on the following property.

Lemma 15 (♣). *Let C be an N-monotone 4^+-clause in a formula $\mathcal{F}$ after Step 4, and $C' = \overline{v_1}\cdots\overline{v_r}$ be any subset of C of $r \geq 2$ literals. We have $\Delta = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{C'=0})) \geq 6w_3$.*

Based on Lemma 15, we know that Step 5.1 can always generate a branching vector not worse that $(6w_3, 6w_3)$. Since Step 5.1 deals with N-monotone 4^+-clauses, next we assume that each N-monotone clause in $\mathcal{F}$ is a 2-clause or 3-clause.

Lemma 16 (♣). *Let C be an N-monotone 3-clause in a reduced formula $\mathcal{F}$ after Step 5.1. For any two negative literals $\overline{y}$ and $\overline{z}$ in C, it holds that $|var(N(y, \mathcal{F}) \cup N(z, \mathcal{F}))| \geq 4$.*

Step 5.2: Assume there is an N-monotone 3-clause $\overline{xyz}$ such that $|C_1| + |C_2| \geq 3$ for the two clauses xC_1 and xC_2 containing x. In this step, we select a literal x such that $|C_1| + |C_2|$ is maximized, and branch with (1) $x = 1$; (2) $x = 0$ and $y = z = 1$.

Lemma 17. *The branching vector generated by Step 5.2 is not worse than $(5w_3, 7w_3)$.*

Proof. Let $\Delta_1 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=1}))$, $\Delta_0 = \mu(\mathcal{F}) - \mu(R(\mathcal{F}_{x=0\&y=z=1}))$, and $N_{yz} = N(y, \mathcal{F}) \cup N(z, \mathcal{F})$.

Case 1: $|C_1| + |C_2| \geq 4$. In the branch $x = 1$, all variables in C_1 and C_2 are reduced. Together with x, we can reduce at least 5 variables. In the branch $x = 0$ and $y = z = 1$, all variables in $var(N_{yz})$ are reduced. By Lemma 16, we know that $|var(N_{yz})| \geq 4$. Additionally, Lemma 5 guarantees that $var(N_{yz}) \cap x, y, z = \emptyset$. In this branch, we reduce at least $|\{x, y, z\}| + |var(N_{yz})| \geq 7$ variables in total. Thus, we can branch with $(5w_3, 7w_3)$ at least.

Case 2: $|C_1| + |C_2| = 3$. W.l.o.g., we assume $C_1 = a$ and $C_2 = bc$. We show that the branching vector is not worse than $(4w_3, 9w_3)$, which is better than $(5w_3, 7w_3)$. In the branch $x = 1$, at least 4 variables a, b, c, and x are reduced.

In the branch $x = 0$ and $y = z = 1$, we can further assign $a = 1$. Let aD be the other one clause containing literal a. Then D will also be reduced after we assign $a = 1$. To sum up, we can reduce variables in $\{x, y, z, a\} \cup var(N_{yz}) \cup var(D)$. We have $\{x, y, z\} \cap var(D) = \emptyset$ and $a \notin N_{yz}$ since R-Rule 13 is not applicable. Note that $|var(N_{yz})| \geq 4$, and $\{x, y, z\} \cap var(N_{yz}) = \emptyset$ by Lemma 5. We can reduce at least $|\{x, y, z, a\} \cup var(N_{yz}) \cup var(D)| = |\{x, y, z, a\}| + |var(N_{yz})| + |var(D) \setminus var(N_{yz})| \geq 9$ variables if $|var(D) \setminus var(N_{yz})| \geq 1$ or $|var(N_{yz})| \geq 5$, and thus $\Delta_0 \geq 9w_3$. Next, we show that we can still reduce 9 variables at least if $|var(N_{yz})| = 4$ and $var(D) \subseteq var(N_{yz})$.

Let yE_1, yE_2, zE_3, and zE_4 be the four clauses containing literal y or z. According to the choice of x, we know that $|E_1| + |E_2| \leq |C_1| + |C_2| = 3$ and so $|E_1| = 1$ or $|E_2| = 1$. Similarly, $|E_3| = 1$ or $|E_4| = 1$. W.l.o.g., we assume that $|E_1| = |E_3| = 1$. It holds that $var(E_1) \neq var(E_3)$ and $var(E_1E_3) \cap var(E_2E_4) = \emptyset$ otherwise R-Rule 13 would be applicable. Since $|var(E_1E_2E_3E_4)| = |var(N_{yz})| = 4$, we have $|var(E_2E_4)| = |var(E_1E_2E_3E_4)| - |var(E_1E_3)| = 2$. If $|E_2|, |E_4| \geq 2$, then $|var(E_2E_4)| \geq 3$ by Lemma 5, which is a contradiction. So we can assume, w.l.o.g., that $|E_2| = 1$ and $1 \leq |E_4| \leq 2$. If $|E_2| = 1$ and $|E_4| = 2$, then we have $var(E_2) \cap var(E_4) = \emptyset$, otherwise R-Rule 13 is applicable. However, this implies a contradiction that $|var(E_2E_4)| = |E_2| + |E_4| \geq 3$. Thus, we have $|E_2| = |E_4| = 1$. Let u be a literal in D. We know that $\overline{u}$ becomes pure after assigning $x = 0$ and $y = z = 1$ since that D and N_{yz} only contain positive literals, and u appears in both of them. So we can further assign $\overline{u} = 1$ in this branch. Assume that the clause containing literal $\overline{u}$ is $\overline{uvw}$, variables v and w are also reduced by $\overline{u} = 1$. Since $|E_i| = 1$ for $1 \leq i \leq 4$, we can apply R-Rule 13 if $\{v, w\} \subseteq var(N_{yz})$. So we have $|\{v, w\} \setminus var(N_{yz})| \geq 1$. Clearly, $\{v, w\} \cap \{x, y, z, a\} = \emptyset$. As a result, we can reduce at least $|\{x, y, z, a\} \cup var(N_{yz}) \cup \{v, w\}| = |\{x, y, z, a\}| + |var(N_{yz})| + |\{v, w\} \setminus var(N_{yz})| \geq 9$ variables, and so we have $\Delta_1 \geq 9w_3$ in this case. This completes the proof. □

Lemma 18 (♣). *After Step 5, all clauses have a length* ≤ 3.

Lemma 18 implies that the formula after Step 5 is a 3-CNF.

Step 6: Fast Solving the Remaining Part

Next, we call a fast 3-SAT algorithm by Beigel and Eppstein to solve our problem directly.

Lemma 19 ([Beigel and Eppstein, 2005])**.** *3-SAT can be solved in time* $O^*(1.3645^t)$ *and polynomial space, where* t *is the number of* 3*-clauses.*

Lemma 20. *Step 6 can be solved in* $O^*((1.1092^{\frac{1}{w_3}})^{\mu(\mathcal{F})})$ *time and polynomial space.*

Proof. Let $\overline{m_2}$ be the number of N-monotone 2-clauses, $\overline{m_3}$ be the number of N-monotone 3-clauses, and m_3 be the number of P-monotone 3-clauses in the formula. Note that the number of 3-clauses in $\mathcal{F}$ is $m_3 + \overline{m_3}$. Since the formula is monotone now and all negative literals appear once, we know that the number of variables is $n = 2\overline{m_2} + 3\overline{m_3}$.

For an N-monotone 3-clause in $\mathcal{F}$, say $\overline{xyz}$, all clauses containing literal $x/y/z$ are 2-clauses after Step 5.2. So, we can know that for a P-monotone 3-clause, say abc, all clauses containing literal $\overline{a}/\overline{b}/\overline{c}$ are 2-clauses. After Step 3, for an N-monotone 2-clause $\overline{xy}$ in $\mathcal{F}$, there are two P-monotone 2-clauses and two P-monotone 3-clauses containing x or y. Thus, we can know that $m_3 = 2\overline{m_2}/3$, and so we have

$$m_3 + \overline{m_3} = \frac{2}{3}m_3 + \overline{m_3} = \frac{2\overline{m_2} + 3\overline{m_3}}{3} = \frac{n}{3}.$$

Then, by calling Beigel and Eppstein's algorithm, we can solve the problem in time $O^*(1.3645^{\frac{n}{3}}) \subseteq O^*(1.1092^n)$. Finally, since the formula is 3-regular, we have $\mu(\mathcal{F}) = w_3 n$ and $O^*(1.1092^n) \subseteq O^*((1.1092^{\frac{1}{w_3}})^{\mu(\mathcal{F})})$. □

5 The Final Results

Now we present our main results. The algorithm will first apply the reduction rules to get a reduced formula and then apply the six branching steps in order. The worst branching vectors for each branching step are summarized in Table 2. Note that these worst branching vectors are not related to the value of the tunable parameter λ as long as $0 \leq \lambda \leq 2$.

By setting $\lambda = 2$ and $w_3 = 1$, we have $\mu(\mathcal{F}) \leq (d - 2)n$ and $w_i = i - 2$ for $i \geq 3$. The biggest branching factor for the first five steps is 1.1238, which is corresponding to the branching vector $(5w_3, 7w_3)$ in Steps 2.3, 3, and 5.2. Since $w_3 = 1$, Step 6 can be solved in time $O^*(1.1092^{\mu(\mathcal{F})})$. Thus, the whole instance can be solved in time $O^*(1.1238^{(d-2)n})$.

Theorem 1. *SAT without 1-variables can be solved in* $O^*(1.1238^{(d-2)n})$ *time and polynomial space.*

By setting $\lambda = 0$ and $w_3 = 1.87884$, we have $\mu(\mathcal{F}) \leq dn$, $w_4 = 3.75768$, and $w_i = i$ for $i \geq 5$. The biggest branching vector for the first five steps is 1.0641, and the bottlenecks are Steps 2.2, 2.3, 3, and 5.2. Under this setting, Step 6 can be solved in time $O^*(1.0568^{\mu(\mathcal{F})})$, and the whole instance can be solved in time $O^*(1.0641^{dn})$. Since $L = dn$, we get that:

Theorem 2. *SAT can be solved in* $O^*(1.0641^{dn})$ *time (resp.,* $O^*(1.0641^L)$ *time) and polynomial space.*

6 Conclusion

In this paper, we study the running-time bounds for SAT with the average occurrence of variables bounded by d. The results for $d = 3$ and 4, which were frequently used in other algorithms, have not been improved for a long time. One reason why it is hard to improve is that several cases will lead to the same bottleneck. In this paper, by using new reduction and branching techniques, we successfully avoid all the previous bottlenecks.

Another interesting technique is that we set a tunable parameter λ in our variable weights. By setting different values to the tunable parameter, we get different running-time bounds for the same algorithm with the same analysis framework. When the tunable parameter λ is smaller, more improvements for small d may propagate to that for large d. When λ reaches the upper bound 2, we get the best results for $d = 3$ and 4. When $\lambda = 0$, the running-time bound for $d \geq 5$ becomes better. This analysis technique may be able to apply to more problems.

Acknowledgements

The work is supported by the National Natural Science Foundation of China, under grant 61972070, and the Natural Science Foundation of Sichuan Province of China, under grant 2023NSFSC0059.

References

[Alferov and Bliznets, 2021] Vasily Alferov and Ivan Bliznets. New length dependent algorithm for maximum satisfiability problem. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3634–3641. AAAI Press, 2021.

[Bansal and Raman, 1999] Nikhil Bansal and Venkatesh Raman. Upper bounds for maxsat: Further improved. In Alok Aggarwal and C. Pandu Rangan, editors, *Algorithms and Computation, 10th International Symposium, ISAAC '99, Chennai, India, December 16-18, 1999, Proceedings*, volume 1741 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 1999.

[Beigel and Eppstein, 2005] Richard Beigel and David Eppstein. 3-coloring in time $O(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005.

[Belova and Bliznets, 2020] Tatiana Belova and Ivan Bliznets. Algorithms for (n, 3)-maxsat and parameterization above the all-true assignment. *Theor. Comput. Sci.*, 803:222–233, 2020.

[Bliznets, 2013] Ivan A Bliznets. A new upper bound for (n, 3)-max-sat. *Journal of Mathematical Sciences*, 188(1):1–6, 2013.

[Brilliantov *et al.*, 2023] Kirill Brilliantov, Vasily Alferov, and Ivan Bliznets. Improved algorithms for maximum satisfiability and its special cases. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, 2023. (to appear).

[Chen and Liu, 2009] Jianer Chen and Yang Liu. An improved SAT algorithm in terms of formula length. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, volume 5664 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2009.

[Chu *et al.*, 2021] Huairui Chu, Mingyu Xiao, and Zhe Zhang. An improved upper bound for SAT. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3707–3714. AAAI Press, 2021.

[Cook, 1971] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[Dantsin *et al.*, 2004] Evgeny Dantsin, Edward A. Hirsch, and Alexander Wolpert. Algorithms for SAT based on search in hamming balls. In Volker Diekert and Michel Habib, editors, *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings*, volume 2996 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 2004.

[Dantsin *et al.*, 2006] Evgeny Dantsin, Edward A. Hirsch, and Alexander Wolpert. Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms. In Tiziana Calamoneri, Irene Finocchi, and Giuseppe F. Italiano, editors, *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, volume 3998 of *Lecture Notes in Computer Science*, pages 60–68. Springer, 2006.

[Davis and Putnam, 1960] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[Fomin and Kratsch, 2010] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[Fomin *et al.*, 2009] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.

[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 Kutskov, 2009] A. S. Kulikov and K. Kutskov. New upper bounds for the problem of maximal satisfiability. *Discret. Appl. Math.*, 19(2):155–172, 2009.

[Liu, 2018] Sixue Liu. Chain, generalization of covering code, and deterministic algorithm for k-SAT. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 88:1–88:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Oliver and Luckhardt, 1997] Kullmann Oliver and Horst Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. *Preprint*, pages 1–82, 1997.

[Paturi *et al.*, 1999] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chic. J. Theor. Comput. Sci.*, 1999, 1999.

[Paturi *et al.*, 2005] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved

exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[Peng and Xiao, 2021] Junqiang Peng and Mingyu Xiao. A fast algorithm for SAT in terms of formula length. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 436–452. Springer, 2021.

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

[Scheder, 2021] Dominik Scheder. PPSZ is better than you think. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 205–216. IEEE, 2021.

[Schöning, 1999] Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414. IEEE Computer Society, 1999.

[Schuler, 2005] Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

[Tovey, 1984] Craig A. Tovey. A simplified np-complete satisfiability problem. *Discret. Appl. Math.*, 8(1):85–89, 1984.

[Wahlström, 2005a] Magnus Wahlström. An algorithm for the SAT problem for formulae of linear length. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2005.

[Wahlström, 2005b] Magnus Wahlström. Faster exact solving of SAT formulae with a low number of occurrences per variable. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2005.

[Xiao, 2022] Mingyu Xiao. An exact maxsat algorithm: Further observations and further improvements. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1887–1893. ijcai.org, 2022.

[Xu *et al.*, 2019] Chao Xu, Jianer Chen, and Jianxin Wang. Resolution and linear CNF formulas: Improved (n, 3)-maxsat algorithms. *Theor. Comput. Sci.*, 774:113–123, 2019.

[Xu *et al.*, 2021] Chao Xu, Wenjun Li, Jianxin Wang, and Yongjie Yang. An improved algorithm for the (n, 3)-maxsat problem: asking branchings to satisfy the clauses. *J. Comb. Optim.*, 42(3):524–542, 2021.