A New Algorithm for Parameterized MAX-SAT*

Ivan Bliznets and Alexander Golovnev

St. Petersburg University of the Russian Academy of Sciences, St. Petersburg, Russia

Abstract. We show how to check whether at least k clauses of an input formula in CNF can be satisfied in time $O^*(1.358^k)$. This improves the bound $O^*(1.370^k)$ proved by Chen and Kanj more than 10 years ago. Though the presented algorithm is based on standard splitting techniques its core are new simplification rules that often allow to reduce the size of case analysis. Our improvement is based on a simple algorithm for a special case of MAX-SAT where each variable appears at most 3 times.

Keywords: exact algorithms, maximum satisfiability, parameterized algorithms, satisfiability.

1 Introduction

1.1 Problem Statement

Maximum Satisfiability (MAX-SAT) is a well known NP-hard problem where for a given boolean formula in conjunctive normal form one is asked to find the maximum number of clauses that can be simultaneously satisfied. In the parameterized version of MAX-SAT the question is to check whether it is possible to find an assignment that satisfies at least k clauses. The best known upper bound $O^*(1.370^k)$ for this problem was given in 2002 by Chen and Kanj [1]. The previously known bounds are listed in the following table.

Bound	Authors	Year
$O^*(1.618^k)$	Mahajan, Raman [2]	1999
$O^*(1.3995^k)$	Niedermeier, Rossmanith [3]	1999
$O^*(1.3803^k)$	Bansal, Raman [4]	1999
$O^*(1.3695^k)$	Chen, Kanj [1]	2002

In this paper, we present an algorithm with the running time $O^*(1.358^k)$ for parameterized MAX-SAT and $O^*(1.273^k)$ for parameterized $(n, 3)$-MAX-SAT. $(n, 3)$-MAX-SAT is a special case of MAX-SAT where each variable appears at most three times.

An alternative way to parametrize MAX-SAT is to ask whether at least $\lceil \frac{m}{2} \rceil + k'$ clauses can be satisfied ([2], [5], [6]). This is called parametrization above

* Research is partially supported by Federal Target Program "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009–2013 and Russian Foundation for Basic Research.

D.M. Thilikos and G.J. Woeginger (Eds.): IPEC 2012, LNCS 7535, pp. 37–48, 2012.

guaranteed values since one can always satisfy at least $\lceil \frac{m}{2} \rceil$ clauses (indeed, the expected number of clauses satisfied by a random assignment is $\frac{m}{2}$). It is shown in [2] that an upper bound ϕ^k for the parametrization considered in this paper implies an upper bound $\phi^{6k'}$ for an alternative parametrization. We do not known any better results for this parametrization.

1.2 General Setting

Literals and Formulas. Throughout the paper, n denotes the number of variables, m denotes the number of clauses, and k denotes the number of clauses one is asked to satisfy. By MAX-SAT(F) we denote the maximum number of clauses in F that can be satisfied simultaneously. MAX-SAT(F, k) = true iff MAX-SAT$(F) \geq k$. The constants true and false are denoted just by 1 and 0. Let $\#_F(l)$ be the number of occurrences of a literal l. By $F[l]$ we denote a formula obtained from F by removing all occurrences of $\bar{l}$ and deleting all clauses containing l. By $F[x = \bar{y}]$ we denote a formula obtained by replacing x and $\bar{x}$ by $\bar{y}$ and y, respectively. We say that a variable *has degree* p if it occurs in the formula exactly p times. Also we say that a variable x is of *type* (a, b) if the literal x occurs a times and the literal $\bar{x}$ occurs b times. We say that a variable x is a $(k, 1)$*-singleton (*$(k, 1)$*-non-singleton)* if it is of type $(k, 1)$ and the only negation is contained (is not contained) in a unit clause. Unit clause is a clause of length 1. A literal l is called *pure* if the literal $\bar{l}$ does not appear in the formula. A literal y *dominates* a literal x if all clauses containing x contain also y. Two literals are called *inconsistent* if one of them is a negation of the second. A literal y is a *neighbor* of a literal x if they appear in a clause together. We use "..." to indicate the rest of a clause. E.g., $(x \vee \bar{y} \vee \dots)$ is a clause containing literals $x, \bar{y}$ and probably something else.

Branchings. Instance of a problem is a pair (F, k). The question is whether it is possible to satisfy at least k clauses in a formula F. For $q > 1$, we say that there exists a branching $(a_1, \dots, a_q)$ if we can quickly construct formulas $F_1, F_2, \dots, F_q$ such that the answer for the original problem can be found from the answers to the problems $(F_1, k - a_1), (F_2, k - a_2), \dots, (F_q, k - a_q)$. If l is a literal of F, then clearly there exists a branching $(F[l], k - \#_F(l)), (F[\bar{l}], k - \#_F(\bar{l}))$. It is well-known that if an algorithm on each stage uses only branchings from the set

$$(a_{1,1}, \dots, a_{1,q_1}), (a_{2,1}, \dots, a_{2,q_2}), \dots, (a_{t,1}, \dots, a_{t,q_t}),$$

where $a_{i,1} \leq a_{i,2} \leq \dots \leq a_{i,q_i}$, for $1 \leq i \leq t$, then its running time is $O^*(c^k)$ where c is the largest positive root of a polynomial

$$p(X) = \prod_{j=1}^{t} (X^{a_{j,q_j}} - \sum_{i=1}^{q_j} X^{a_{j,q_j} - a_{j,i}}).$$

For branching $(a_1, \dots, a_q)$, where $a_1 \leq a_2 \leq \dots \leq a_q$, we denote by $\tau(a_1, a_2, \dots, a_q)$ the unique positive root of a polynomial $X^{a_q} - (X^{a_q - a_1} + X^{a_q - a_2} + \dots + X^{a_q - a_q})$. $\tau(a_1, a_2, \dots, a_q)$ is called *a branching number*.

We say that a branching $(b_1, b_2, \ldots, b_q)$ *is dominated* by branching a $(a_1, a_2, \ldots, a_q)$ iff for every $i, a_i \geq b_i$.

1.3 The Main Idea of the Algorithm

A straightforward branching on a variable of high degree immediately gives a good branching number. As it is common with branching algorithms, the main bottleneck is when a formula consists of variables of low degree only. It is easy to see that variables of degree at most 2 can be eliminated from the formula. Consider a variable x of degree 3: $(x \vee A)(x \vee B)(\bar{x} \vee C)$, where A, B, C are disjunctions of literals. If A or B consists of just one literal, then we can replace $(x \vee A)(x \vee B)(\bar{x} \vee C)$ with $(\bar{A} \vee B \vee C)(A \vee C)$. If A and B are long, then we can branch according to the following "resolution-like" rule:

- replace $(x \vee A)(x \vee B)(\bar{x} \vee C)$ by $(A \vee C)(B \vee C)$;
- set to 0 all literals from A, B, C.

More formally the correctness of these steps is shown in Simplification Rule 5 and Branching Rule 2.

For solving MAX-SAT restricted to instances consisting of $(3,1)$- and $(4,1)$-singletons we use the algorithm for the Minimum Set Cover problem by van Rooij and Bodlaender [7]. The running time of the algorithm is estimated in Theorem 3.

1.4 Organization of the Paper

In Section 3 we present a very simple algorithm for $(n,3)$-MAX-SAT. Its analysis is based on tricks mentioned above and contains no case analysis at all. In Section 4 we show that the presented rules can be used to simplify a case analysis of branching on a variable of degree 3. In Section 5 we improve the upper bound for Parameterized MAX-SAT.

2 Preliminaries: Simplification and Branching Rules

The following simplification rule is straightforward so we state it without a proof.

Simplification Rule 1. *A literal l can be assigned the value 1 if l is a pure literal or number of unit clauses (l) is not smaller than number of clauses containing $\bar{l}$.*

Simplification Rule 2. *A variable of degree ≤ 2 can be eliminated.*

Correctness: If l is a pure literal, then we can set $l = 1$. Otherwise, $F = G \wedge (l \vee A) \wedge (\bar{l} \vee B)$. It is easy to see that MAX-SAT(F, k) = MAX-SAT$(F \wedge (A \vee B), k-1)$. □

Simplification Rule 3. *Pairs of clauses (x) and $(\bar{x})$ can be removed.*

Correctness: Clearly, MAX-SAT$(F \vee (x) \vee (\bar{x}), k) =$ MAX-SAT$(F, k-1)$. It does not matter whether the variable x appears in F or not. □

Simplification Rule 4. *If two variables x and y of degree 3 appear together in 3 clauses, then all these 3 clauses can be satisfied by assigning x and y.*

Correctness: One can satisfy 2 clauses by assigning x the remaining clause can be satisfied by assigning y. □

Simplification Rule 5. *Let x be a variable of degree 3: $F = G \wedge (x \vee A) \wedge (x \vee B) \wedge (\bar{x} \vee C)$. If A or B has length < 2 then we can reduce the problem.*

Correctness: Wlog, assume that A has length < 2. If the length of A equals to 1 then A is a single literal. It is easy to see that

$$\text{MAX-SAT}(F, k) = \text{MAX-SAT}(G \wedge (\bar{A} \vee B \vee C) \wedge (A \vee C), k-1).$$

If A is empty we can set $x = 1$. The parameter is reduced by 2. □

Remark 1. It is easy to see that all Simplification Rules can be applied in polynomial time and decrease k at least by one. Note that some of the simplifications rules make several clauses of a formula satisfied while others may replace existing clauses with new clauses and reduce the parameter (like SR2 and SR5). Since as a result of applying a rule the number of satisfied clauses increases we usually say that applying a simplification rule satisfies some clauses.

Branching Rule 1. *For any literal l, one can branch as $(F[l], k - \#_F(l)), (F[\bar{l}], k - \#_F(\bar{l}))$.*

Branching Rule 2. *Let x be a variable of degree 3: $F = G \wedge (x \vee A) \wedge (x \vee B) \wedge (\bar{x} \vee C)$. Then there is a branching:*

- $(G \wedge (A \vee C) \wedge (B \vee C), k-1)$
- $(G', k-2)$*, where G' is obtained from G by assigning all literals from A, B, C to 0.*

Correctness: Let $R = (A \vee C) \wedge (B \vee C)$. It is a simple observation that if an optimal assignment satisfies s clauses from R, where $s = 1, 2$, then we can satisfy $s+1$ clauses from $F - G$ but cannot satisfy $s+2$. However, if an optimal assignment does not satisfy any clause from R we can still satisfy two clauses from $F - G$ by setting $x = 1$. □

Corollary 1. *If $A \cup B \cup C$ contains inconsistent literals, then one can consider only the first branch. It means that one can reduce (F, k) to $(G \wedge (A \vee C) \wedge (B \vee C), k-1)$.*

Remark 2. We write $BR2(x)$ if we apply Branching Rule 2 to a variable x. Branching on a variable means applying Branching Rule 1. We write $SRi(x)$ for $1 \leq i \leq 5$, if we apply Simplification Rule i to a variable x.

Lemma 1 (Kulikov, Kutzkov [8]). *If a literal y dominates a literal x, then one can branch as*

- $x = 1, y = 0$;
- $x = 0$.

Proof. If in some assignment the literals x and y both have the value 1, then flipping the value of x cannot decrease the number of satisfied clauses. Indeed, all clauses that can be satisfied by $x = 1$ are also satisfied by $y = 1$. □

Lemma 2. *Let x be a $(t, 1)$-non-singleton variable. Then branching on x is a $(t, 2)$-branching.*

Proof. Let y be a neighbor of $\bar{x}$. In the branch $x = 1$ we satisfy at least t clauses. In the branch $x = 0$ we can set $y = 0$ and satisfy at least 2 clauses. The lemma follows from Lemma 1, in this case we use literals $y, \bar{x}$ instead of y, x. □

Lemma 3. *If F contains a variable x of degree ≥ 6, then branching number on x is at most $\tau(1, 5)$.*

Proof. This follows from the fact that $\tau(1, 5) > \tau(2, 4) > \tau(3, 3)$. □

3 Solving $(n, 3)$-MAX-SAT in 1.2721^k Time

By $(n, 3)$-MAX-SAT we denote MAX-SAT restricted to instances in which each variable appears in at most 3 clauses. In this section we give a simple algorithm for $(n, 3)$-MAX-SAT. The running time of the algorithm is 1.2721^k. Note that the previous known upper bound for $(n, 3)$-MAX-SAT w.r.t. k is 1.3247^k and it follows from proof of Chen and Kanj for the general MAX-SAT. Throughout this section we assume that F is an $(n, 3)$-MAX-SAT formula.

Lemma 4. *Let x be a variable of degree 3: $F = G \wedge (x \vee A) \wedge (x \vee B) \wedge (\bar{x} \vee C)$ and rules SR1-4 are not applicable to F. If A has length < 2, then we have $(2, 4)$-branching and the resulting formulas are $(n, 3)$-MAX-SAT formulas.*

Proof. If A has length 0, then we can set $x = 1$. Otherwise, by Simplification Rule 5 we eliminate one clause and get a new formula $F' = G \wedge (\bar{A} \vee B \vee C) \wedge (A \vee C)$. Variables of degree 4 in the formula F' can appear only in A and C. Branching on the variable A gives $(n, 3)$-MAX-SAT formulas in both branches. A has degree 4, so the branching gives at least $(1, 3)$-branching (note that $\tau(2, 2) < \tau(1, 3)$). As one clause is already satisfied, the resulting branching is at least $(2, 4)$. □

Lemma 5 (Bliznets [9]). *If each variable of F appears once negatively and twice positively and all negative literals occur in unit clauses, then MAX-SAT(F) can be computed in polynomial time.*

Proof. Construct a graph $G_F = (V, E)$ in the following way. Introduce a vertex for each clause consisting of positive literals, introduce an edge between two vertices if the corresponding two clauses share a variable. Then MAX-SAT$(F) = n + \nu(G_F)$, where $\nu(G_F)$ is the size of a maximum matching in G_F. □

Algorithm 1. $(n,3)$-MAX-SAT-ALG — solving $(n,3)$-MAX-SAT in time 1.2721^k.

Input: F — instance of $(n,3)$-MAX-SAT.
Parameter: k — number of clauses asked to satisfy.
Output: *true*, if k clauses can be satisfied simultaneously; *false* otherwise.

1: apply Simplification Rules 1–4
2: **if** all negations are singletons **then**
3: return answer (use Lemma 5).
4: choose x, s.t. $\bar{x}$ is not a singleton: $(x \vee A)(x \vee B)(\bar{x} \vee C), |C| > 0, |A| \leq |B|$
5: **if** $|A| \leq 1$ **then**
6: use Lemma 4 for branching
7: **if** $|A| \geq 2$ **then**
8: branch BR 1(x)

Theorem 1. *Algorithm* $(n,3)$-MAX-SAT-ALG *solves* $(n,3)$*-MAX-SAT in time* 1.2721^k.

Proof. By Lemma 5 the running time of step 3 is polynomial. Step 6 gives $(2,4)$-branching by Lemma 4. Branching at step 8 gives at least $(4,2)$-branching. Indeed, $|C| > 0$ and Lemma 1 implies that in case $x = 0$ we can satisfy at least two clauses: one is $(\bar{x} \vee C)$ and one more with a literal $\bar{t}$ where t is some literal from C. By SR4, there are at least 4 clauses containing variables from A. In the branch $x = 1$ two clauses are satisfied by x and there exist variables y, z that appear one or two times in the new formula. There are at least 2 clauses that contain variable y or z. Hence, SR2 applied to variables y, z from A satisfies two new clauses. Thus, branching on x gives $(4,2)$-branching. The running time of the algorithm is $\max(\tau(2,4), \tau(3,3))^k = \tau(2,4)^k < 1.2721^k$. □

4 Removing Variables of Degree 3

In this section we show that if a formula contains a variable of degree 3 then we can either decrease k or find a good branching. Suppose that x occurs three times in F and no simplification rules are applicable to F. Assume that F contains clauses $(x \vee A), (x \vee B), (\bar{x} \vee C)$ where A, B, C are disjunctions of literals. We consider only cases where $|A|, |B| \geq 2$ because otherwise we can apply SR5(x) or assign 1 to x.

Definition 1. *Denote by* $LN(!A_1, \ldots, !A_k)$ *the set of all clauses of* F *containing a negation of some literal from* $A_1 \cup \cdots \cup A_k$

Throughout this section we assume that $A \cup B \cup C$ does not contain inconsistent literals. Otherwise, by Corollary 1 the formula can be simplified.

Lemma 6. *Let* y, z *be literals such that* $y, z \in A \cup B \cup C$, $\bar{y}$ *occurs two or more times and dominates* $\bar{z}$ *(call this situation* a first special case of domination*). Then there is a* $(2,4)$*-branching.*

Proof. Recall that $\bar{y}, \bar{z}$ appear in some clauses from $LN(!A, !B, !C)$. Consider the branching $F[y], F[\bar{y}]$. In the second case two clauses are satisfied with $\bar{y}$ and z can be substituted by 1, because it becomes a pure literal. $z = 1$ satisfies at least one of the following clauses: $(x \vee A), (x \vee B), (\bar{x} \vee C)$. After that we use SR2($x$) since x appears at most two times. This satisfies at least 4 clauses. In the formula $F[y]$ we use SR2(x) and that is why the parameter is decreased by 2. □

Lemma 7. *Let y, z be literals from $A \cup B \cup C$ such that $\bar{y}$ occurs once and dominates $\bar{z}$ (call this situation* a second special case of domination*). Then there is a $(3, 3)$-branching.*

Proof. Like in the previous lemma in $F[\bar{y}]$ we can assign 1 to z. A literal z appears at least two times because $\bar{z}$ occurs once in the formula. So $z = 1$ satisfies two more clauses(z and $\bar{z}$ do not appear in one clause). This satisfies at least 3 clauses. In $F[y]$ — we satisfy 2 clauses containing y and one using SR2(x) because after assigning $y = 1$ we have one or two clauses containing the variable x. We get a $(3, 3)-$branching. □

Lemma 8. *If $|LN(!A, !B, !C)| < 3$ then we have a special case of domination or $A = B = y \vee z$. In the former case we have a good branching ($(2, 4)$-, $(3, 3)$-, $(1, 6)$-branching or better), while in the latter case the parameter can be decreased.*

Proof. We know that $|A|, |B| \geq 2$. So $A \cup B$ contains more than two literals or equals $y \vee z$. In the former case three literals should occur in two clauses, so this is a domination. In the latter case we can replace clauses with the variable x by the clause $(y \vee z \vee C)$ and decrease the parameter by two. □

Now we can assume that we only work with formulas where $|LN(!A, !B, !C)| > 2$. If $|LN(!A, !B, !C)| > 3$ using BR2(x) we immediately get $(1, 6)$-branching. So for the rest of this section $|LN(!A, !B, !C)| = 3$.

Lemma 9. *If $|A \cup B \cup C| > 3$ then we have a special case of domination and hence one of the Lemmas 6,7 is applicable. So there is a $(2, 4)$- or $(3, 3)$-branching.*

Proof. At least 4 negations of the literals from $|A \cup B \cup C|$ should be placed in 3 clauses and it is impossible without special case of domination. □

From the previous lemma we conclude that it is enough to consider formulas where $|A \cup B \cup C| \leq 3$.

Lemma 10. *If $min\{|A|, |B|\} \geq 3$ then either there is a special case of domination or the parameter can be decreased.*

Proof. If $|A \cup B \cup C| > 3$ we have a special case of domination because of Lemma 9. Otherwise from $|A \cup B \cup C| \leq 3$ and $min\{|A|, |B|\} \geq 3$ it follows that $|A| = |B| = 3$ and $x \vee A = x \vee B = x \vee y_1 \vee y_2 \vee y_3$. So, we can replace $x \vee A, x \vee B, \bar{x} \vee C$ by $A \vee C$ and decrease the parameter by 2. □

Now wlog we can assume that $x \vee A = x \vee y \vee z$.

Lemma 11. *If we have an instance (F, k) and for all variables x that appear three times the following holds:*

- *x occurs in clauses $x \vee A_x, x \vee B_x, \bar{x} \vee C_x$*
- *$LN(!A_x, !B_x, !C_x) = 3$*

then we can decrease the parameter or apply one of the following branchings: $(3,3), (2,4)$ or better.

Proof. Suppose that all previous lemmas are not applicable otherwise we are done. So we can choose a variable x that occurs three times and $A_x = y \vee z$. Note that if $\bar{y}$ occurs three times then we have a case of domination and in this situation a good branching ($(2,4)$-, $(3,3)$-, $(1,6)$-branching or better) exists. Consider two cases: $\bar{y}$ appears exactly once or twice.

Case 1: $\bar{y}$ appears exactly once.

Case 1.1: $\bar{y}$ has a neighbor.
F contains the following clauses:

$$(x \vee y \vee z), \quad (x \vee \dots), \quad (\bar{x} \vee \dots), \quad (\bar{y} \vee w \vee \dots).$$

In $F[\bar{y}]$ by Lemma 1 we can assign 0 to w and use SR5(x) or SR2(x). So, we decrease the parameter by 3. In $F[y]$ we satisfy at least two clauses and using SR2(x) we decrease the parameter by 1. We obtain $(3,3)$-branching. So in all other cases we must have a clause $(\bar{y})$.

Case 1.2: y appears more than 2 times and there is an occurrence outside a variable x.
After $F[y]$ and SR2(x) we satisfy at least 4 clauses. In $F[\bar{y}]$ using SR5(x) we decrease the parameter by 2.

Case 1.3: a literal y appears in all clauses with a variable x.
We have the following clauses:

$$(x \vee y \vee z), \quad (x \vee y \vee \dots), \quad (\bar{x} \vee y \vee \dots), \quad (\bar{y}).$$

The variable y does not occur in the rest of the formula, otherwise we can treat it as a case 1.2. So, it is enough to consider $y = 0$. Because an assignment with $x = 1, y = 0$ is not worse than the same assignment with $x = y = 1$ and $x = 0, y = 1$. It means that we can satisfy one clause and one variable without branching.

Case 1.4: y occurs exactly twice
Using symmetry we can conclude that the clause with $\bar{x}$ does not contain any other literals. Otherwise we have case 1.1. Again using symmetry ideas we may conclude that either $\bar{z}$ appears twice or $\bar{z}$ appears once and z appears exactly twice and there is a clause $(\bar{z})$. Consider these two subcases separately.

Case 1.4.1: $\bar{z}$ appears once and z appears exactly twice

$$(x \vee y \vee z), \quad (x \vee \dots), \quad (\bar{x}), \quad (\bar{y}), \quad (\bar{z}).$$

In $F[x]$ using SR2(y), SR2(z) we decrease the parameter by 4. In $F[\bar{x}]$ it is easy to see that we can assume $y = \bar{z}$. So we obtain $(4, 4)$-branching.

Case 1.4.2: $\bar{z}$ appears twice.

In this case we have the following clauses:

$$(x \vee y \vee z), \quad (x \vee \dots), \quad (\bar{x}), \quad (\bar{y}), \quad (\bar{z} \vee \dots), \quad (\bar{z} \vee \dots).$$

$F[z]$ and then SR2(x), SR2(y) decrease the parameter by 3. In $F[\bar{z}]$ we can assume that $x = \bar{y}$. We get a $(3, 4)$-branching.

Case 2: $\bar{y}$ appears exactly twice.

Using symmetry we conclude that $\bar{z}$ also appears twice otherwise we have the situation described in case 1. So, we have the following family of clauses:

$$(x \vee y \vee z), \quad (x \vee B), \quad (\bar{x} \vee \dots), \quad (\bar{y} \vee \dots), \quad (\bar{y} \vee \dots).$$

Assume $y \in B$ (the case $z \in B$ is similar). $F[y]$ and then $x = 0$ removes 3 clauses. In $F[\bar{y}]$ we use SR5(x) and this removes 3 clauses. If y, z do not appear in B we have $|B| < 2$ or $|A \cup B| \geq 4$ and we get a special domination case, considered before.

□

Theorem 2. *If x occurs exactly 3 times in F, then either the parameter can be decreased or there is a $(1, 6)$-, $(2, 4)$- or $(3, 3)$-branching.*

5 Solving MAX-SAT in 1.358^k

In this section we present a simple algorithm that improves the upper bound for Parameterized MAX-SAT (Algorithm MAX-SAT-Alg). The main bottleneck of the analysis is when all variables are $(1, 3)$-singletons or $(1, 4)$-singletons. We consider this case separately.

We reduce an instance of this restricted MAX-SAT to the instance of Minimum Set Cover. The Minimum Set Cover is, given a universe U and a collection $\mathcal{S}$ of subsets of U, to find the minimum cardinality of a subset $S' \subset \mathcal{S}$ which covers U: $\bigcup_{S_i \in S'} S_i = U$. For $e \in U$, $f(e)$ (frequency of e) denotes the number of subsets of $\mathcal{S}$ in which e is contained.

It can be shown that algorithm for the Minimum Dominating Set given by van Rooij and Bodlaender [7] in fact solves also the Minimum Set Cover in time $1.28759^{k(U,\mathcal{S})}$, where $k(U, \mathcal{S}) = \sum_{e \in U} v(f(e)) + \sum_{S_i \in \mathcal{S}} w(|S_i|)$, and v, w are weight functions. The maximum value of v is 0.595723 and the maximum value of w is 1. We note that for sets of cardinality ≤ 4 the maximum value of w is 0.866888 (see the end of section 3 in [7]). Therefore, we can use the following lemma due to van Rooij and Bodlaender [7].

Theorem 3. *Algorithm* MSC *solves the Minimum Set Cover problem where the cardinality of each set in S is at most 4 in time* $O^*(1.28759^{0.595723|U|+0.866888|\mathcal{S}|}) = O^*(1.29^{0.6|U|+0.9|\mathcal{S}|})$.

The following theorem was proved by Lieberherr and Specker [10]. Later Yannakakis [11] gave a simple proof of this bound by the probabilistic method.

Theorem 4. *If any three clauses in F are satisfiable, then at least $\frac{2m}{3}$ clauses are simultaneously satisfiable.*

Theorem 3 is used for instances with $m < 1.5k$, while Theorem 4 is used for instances with $m \geq 1.5k$. We are now ready to prove an upper bound.

Theorem 5. *Algorithm* MAX-SAT-Alg *solves MAX-SAT in time* $O^*(1.3579^k)$.

Proof. Below we show that in each case the algorithm branches with branching number at most $\tau(5, 10, 1) < 1.3579$, so the running time of the algorithm is $O^*(1.3579^k)$.

- Step 3. If $deg(x) \geq 6$ then by Lemma 3 we get $(1, 5)$-branching. $\tau(1, 5) \approx 1.3248 < 1.3579$.
- Step 5. If $deg(x) = 3$ then by Theorem 2 we get $(1, 6)$-branching. $\tau(1, 6) \approx 1.2852 < 1.3579$.
- Step 7. A $(3, 2)$-variable gives $\tau(3, 2) \approx 1.3248 < 1.3579$. By Corollary 2, branching on $(4, 1)$-non-singleton or $(3, 1)$-non-singleton gives at least $\tau(3, 2) \approx 1.3248 < 1.3579$.
- Step 10. x is a $(2, 2)$-variable, y is a $(1, 4)$-singleton, neighbor of x and literal y does not dominate $x, \bar{x}$ simultaneously. Branching on y gives $\tau(4, 1)$ and the next iteration in branch $y = 1$ has a variable of degree 3 or smaller. The overall branching number is smaller than $\tau(4 + 1, 4 + 6, 1) = \tau(5, 10, 1) < 1.3579$.
- Step 12. x is a $(2, 2)$-variable. Neighbors of x are variables of degree 4 or $x, \bar{x}$ are dominated by y. So, both $F[x]$ and $F[\bar{x}]$ contain a variable of degree 3. By Theorem 2, a variable of degree 3 gives $(1, 6)$,- $(2, 4)$- or $(3, 3)$-branching. So the possible branchings are $\tau(2 + 1, 2 + 6, 2 + 1, 2 + 6), \tau(2 + 2, 2 + 3, 2 + 2, 2+3), \tau(2+3, 2+3, 2+3, 2+3)$ the worst case among them is $\tau(3, 8, 3, 8) \approx 1.3480 < 1.3579$.

In the following we assume that all variables are $(3, 1)$- or $(4, 1)$-singletons.

- Step 14. Now all variables are singletons. It means that we can satisfy n clauses by setting all variables to 0. If $k \leq n$ this solves the problem.
- Step 16. We assume that each variable occurs 3 or 4 times positively and once negatively in a unit clause. Note that all clauses are either negative singletons or positive clauses (all variables in positive clauses occur only positively). We claim that for such a formula there always exists an optimal assignment satisfying all positive clauses. Indeed, if some positive clause is not satisfied then by flipping any of its variables we can only increase the number of

Algorithm 2. MAX-SAT-ALG — solving MAX-SAT in time 1.3579^k.

Input: F — instance of MAX-SAT.
Parameter: k — number of clauses asked to satisfy.
Output: 1, if k clauses can be satisfied simultaneously; 0 otherwise.

1: apply Simplification Rules 1–5.
2: **if** there is x, s.t. $deg(x) \geq 6$ **then**
3: branch on x
4: **if** there is x of degree 3 **then**
5: branch on x according to Theorem 2
{Now we have only variables of degree 4 and 5.}
6: **if** F contains a variable x of type $(3,2)$, $(3,1)$-non-singleton or $(4,1)$-non-singleton **then**
7: branch on x
{Now we have only singletons and $(2,2)$-variables.}
8: **if** F contains a variable x of type $(2,2)$ **then**
9: **if** x has a neighbor $(4,1)$-singleton y and $x, \bar{x}$ are not simultaneously dominated by y **then**
10: branch on y
11: **else**
12: branch on x
{Now all variables are $(3,1)$-singletons or $(4,1)$-singletons.}
13: **if** $k \leq n$ **then**
14: **return** 1
15: **if** $m < 1.5k$ **then**
16: **return** $k \leq \text{MSC}(F)$
17: **if** there is a clause of length 2: $(x \vee y)$ **then**
18: branch as $F[x, y]; F[x = \bar{y}]$.
19: **else**
20: **return** 1

satisfied clauses. It means that we want to assign 1 to the minimal number of variables to satisfy all positive clauses. It is the Minimum Set Cover problem. We construct an instance of the Minimum Set Cover problem in the following way. U is the set of positive clauses ($|U| = m-n$). $\mathcal{S}$ contains n sets. Set $S_i \in \mathcal{S}$ consists of positive clauses, which contains a variable x_i. Now we would like to cover U by the minimal number of sets from $\mathcal{S}$. If t is the minimal number of sets required to cover U, then the maximum number of satisfied clauses is $m-t$. We can compare this number to k and return the result. By Theorem 3, the algorithm for Minimum Set Cover for sets of cardinality ≤ 4 has running time $T(F) = O^*(1.29^{(0.6|U|+0.9|\mathcal{S}|)}) = O^*(1.29^{(0.6(m-n)+0.9n)})$. We know that $k > n$ and $m < 1.5k$. Thus $T(F) \approx 1.3574^k < 1.3579^k$.

- Step 18. We know that the formula contains clauses $(\bar{x})$ and $(\bar{y})$. If there is a clause $(x \vee y)$, then some optimal solution satisfies clause $(x \vee y)$. Indeed, if it does not, we can just set $x = 1$ and the number of satisfied clauses does not decrease. So, we can branch as $x = y = 1$ and $x = \bar{y}$. In the first branch we satisfy at least 3 clauses, because x is a $(3,1)-$ or $(4,1)-$variable. In

the second branch we satisfy clause $(x \vee y)$ and by Simplification Rule 3 we satisfy one of the clauses (x) and $(\bar{x})$. We obtain $(2,3)$-branching.

- Step 20. Now we have a formula with $m \geq 1.5k$ clauses. F does not contain clauses of length 2. Therefore, every triple of clauses is satisfiable. By Theorem 4 there is an assignment, which satisfies at least $\frac{2m}{3} \geq k$ clauses. □

Acknowledgments. We are greatful to Konstantin Kutzkov for fruitful discussions and suggesting Branching Rule 2 . We would like to thank our supervisor Alexander S. Kulikov for help in writing this paper and valuable comments. Also we thank anonymous reviewers who helped improve the paper.

References

1. Chen, J., Kanj, I.A.: Improved Exact Algorithms for MAX-SAT. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 341–355. Springer, Heidelberg (2002)
2. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSat and MaxCut. J. Algorithms 31(2), 335–354 (1999)
3. Niedermeier, R., Rossmanith, P.: New Upper Bounds for MaxSat. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 575–584. Springer, Heidelberg (1999)
4. Bansal, N., Raman, V.: Upper Bounds for MaxSat: Further Improved. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 247–258. Springer, Heidelberg (1999)
5. Alon, N., Gutin, G., Kim, E., Szeider, S., Yeo, A.: Solving MAX-r-SAT Above a Tight Lower Bound. Algorithmica 61, 638–655 (2011)
6. Crowston, R., Gutin, G., Jones, M., Yeo, A.: A New Lower Bound on the Maximum Number of Satisfied Clauses in Max-SAT and Its Algorithmic Applications. Algorithmica 64(1), 56–68 (2012)
7. van Rooij, J.M., Bodlaender, H.L.: Exact algorithms for dominating set. Discrete Applied Mathematics 159(17), 2147–2164 (2011)
8. Kulikov, A., Kutzkov, K.: New Bounds for MAX-SAT by Clause Learning. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 194–204. Springer, Heidelberg (2007)
9. Bliznets: A New Upper Bound for $(n,3)$-MAX-SAT. Zapiski Nauchnikh Seminarov POMI, 5–14 (2012)
10. Lieberherr, K.J., Specker, E.: Complexity of Partial Satisfaction. J. ACM 28, 411–421 (1981)
11. Yannakakis, M.: On the approximation of maximum satisfiability. In: SODA 1992, pp. 1–9 (1992)