New Upper Bounds for Maximum Satisfiability*

Rolf Niedermeier†
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
niedermr@informatik.uni-tuebingen.de

Peter Rossmanith
Institut für Informatik, Technische Universität München,
Arcisstr. 21, D-80290 München, Fed. Rep. of Germany
rossmani@in.tum.de

Abstract

The (unweighted) Maximum Satisfiability problem (MaxSat) is: given a boolean formula in conjunctive normal form, find a truth assignment that satisfies the most number of clauses. This paper describes exact algorithms that provide new upper bounds for MaxSat. We prove that MaxSat can be solved in time $O(|F| \cdot 1.3803^K)$, where $|F|$ is the length of a formula F in conjunctive normal form and K is the number of clauses in F. We also prove the time bounds $O(|F| \cdot 1.3995^k)$, where k is the maximum number of satisfiable clauses, and $O(1.1279^{|F|})$ for the same problem. For Max2Sat this implies a bound of $O(1.2722^K)$.

*An extended abstract of this paper was presented at the 26th *International Colloquium on Automata, Languages, and Programming* (ICALP'99), *LNCS* 1644, Springer-Verlag, pages 575–584, held in Prague, Czech Republic, July 11-15, 1999.

†Supported by a Feodor Lynen fellowship (1998) of the Alexander von Humboldt-Stiftung, Bonn, and the Center for Discrete Mathematics, Theoretical Computer Science and Applications (DIMATIA), Prague.

1 Introduction

Despite their intractability, many *NP*-hard problems have to be solved in practice. Aside from approaches like approximation or heuristic algorithms, it is often important to have exact algorithms with provable performance bounds. Thus, great effort has been made to develop efficient exponential time algorithms as many publications in this field, e.g., [7, 23, 29, 28, 37, 40, 42, 43, 47], show. With the advent of parameterized complexity theory [15] a special class of exact algorithms has become increasingly important [16, 32]. As an example, consider the *NP*-complete Vertex Cover problem: given a graph $G = (V, E)$ and a positive integer k, is there a subset of vertices $C \subseteq V$ of size $|C| \leq k$ such that each edge in E has at least one endpoint in C? Setting $n := |V|$, the best known exact algorithm for this problem has running time $O(1.211^n)$ [40]. There is, however, another exact ("fixed parameter") algorithm solving Vertex Cover in running time $O(1.29175^k + kn)$ [34], very recently further improved to $O(1.272^k + kn)$ [10, 33]. For instance, for $k \leq n/2$ it is already much more efficient than the $O(1.211^n)$ algorithm. The results for the Maximum Satisfiability problem (MaxSat) presented in this paper contribute to both lines of research on exact algorithms for *NP*-hard problems. Moreover, besides providing new upper worst case time bounds, due to the seemingly close connections between exact and heuristic algorithms [16, 32], they might also contribute to the development of practical (average case) algorithms for MaxSat.

MaxSat can be stated as follows: Given a boolean formula in conjunctive normal form, find a truth assignment satisfying the most number of clauses. Like the satisfiability problem itself, MaxSat plays an important role in computer science, since it is the basis for solutions of major problems in AI and combinatorial optimization [6, 21, 45, 46]. It has also been a subject of the second DIMACS challenge [26]. It has been termed 'a paradigmatic problem for the "algorithmic engineering" and scientific testing and tuning effort' [5]. According to Crescenzi and Kann [12], MaxSat is among the 15 most popular problems in combinatorial optimization.

MaxSat cannot be solved in polynomial time unless $P = NP$, since it generalizes the satisfiability problem [18, 35]. Basically, three approaches were suggested to overcome MaxSat's intractability implied by its *NP*-hardness:

1. *Approximation algorithms* [11, 25, 49]. If we do not demand that the solution be exact, rather only approximately correct, it is possible to solve MaxSat in polynomial time. There is a deterministic, polynomial time approximation algorithm for MaxSat with approximation factor 0.770 [2]. Very recently, this was improved to 0.7845 [3], indicating that further improvements are possible when making use of a conjecture of Zwick [51]. However, a polynomial time approximation algorithm with an approximation factor arbitrarily close to 1 will not exist unless $P = NP$ [1]. For Max3Sat the approximation factor cannot be better than $\frac{7}{8}$ and for Max2Sat no better than 0.955 [22]. On the other hand, Max3Sat can be approximated within a factor of 0.801 [44] and Max2Sat within a factor of 0.931 [17]. In addition, for Max3Sat there is a randomized approximation algorithm that gives a factor of $\frac{7}{8}$ for satisfiable instances, thus matching the above lower bound in this case [27].

   Dantsin *et al.* show how to improve the MaxSat approximation factor of 0.770 arbitrarily close to 1 using an exponential time algorithm [13]. More precisely, they describe, given a polynomial time α-approximation algorithm, how to construct an $(\alpha + \epsilon)$-approximation algorithm running in time exponential in the number of clauses.

2. *Heuristics.* There is a large body of work on exact algorithms for MaxSat that use clever heuristics which work fast for practical instances, e.g., [5, 8, 45, 46, 48]. They are analyzed and compared empirically. They do not give, however, any worst-case estimates. Thus, there is a large gap between theoretical and practical results for MaxSat.

3. *Fixed parameter tractability* [15, 16, 32, 39]. The natural parameterized version of MaxSat is to determine whether at least k clauses of a CNF formula F with K clauses can be satisfied. Cai and Chen [9] proved that parameterized Max$q$Sat for a fixed constant q is "fixed parameter tractable," meaning that there is an $O(f(k)|F|^{O(1)})$ algorithm for MaxSat. Here, f is an arbitrary function that *only* depends on k where k is the number of clauses to be satisfied. They found an $f(k) = 2^{O(k)}$. The fixed parameter tractability of MaxSat implies that every problem in the optimization class *MaxSNP* [36] is also fixed parameter tractable. Mahajan and Raman [30] introduced a

more meaningful[1] parameterization, asking whether at least $\lceil K/2 \rceil + k$ clauses of a CNF formula F can be satisfied. For the original problem, however, Mahajan and Raman [30] presented an algorithm running in time $O(|F| + (1.6181)^k k)$ that determines whether at least k clauses of a CNF formula F are satisfiable. This also gave the so far best worst case time bound for an exact MaxSat algorithm and was independently achieved by Dantsin *et al.* [13], using it as a basis in developing upper bounds for approximation algorithms.

So far, worst case complexity analysis for problems from logic has mainly focused on the classical Sat problem, e.g., [23, 28, 29, 31, 37, 38, 41, 50], but to our knowledge, comparatively little work has been done for MaxSat [13, 30]. Our results provide new worst case upper bounds for MaxSat. Besides improving previous work [13, 30], our result applies to *all* the above three points: It improves an existing approximation algorithm [13], improves known fixed parameter tractability results [9, 30], and finally, provides an algorithm that may serve as a basis for heuristic approaches in solving MaxSat.

Our main results are as follows. We prove that MaxSat can be solved in time $O(|F| \cdot 1.3803^K)$, where $|F|$ is the length of the formula in conjunctive normal form and K is the number of clauses in F. We also prove the time bound $O(1.1279^{|F|})$ for the same problem, which implies a bound of $O(1.2722^K)$ steps for Max2Sat, since then $|F| \leq 2K$. This also implies improvements in the results from Mahajan and Raman [30] concerning the MaxCut problem and a "different parameterization" of MaxSat. For example, in the case of MaxCut, where we are given an undirected graph on n vertices and m edges and a positive integer k and asked for the existence of a cut of size at least k, we improve Mahajan and Raman's time bound of $O(k4^k + m + n)$ [30] to $O(k^2 2.6196^k + m + n)$. Finally, we can improve the approximation algorithm of Dantsin *et al.* [13] by simply replacing their exponential time algorithm with our faster algorithm. For example, based on a given polynomial time α-approximation algorithm, they described an $(\alpha + \epsilon)$-algorithm running in time $O(|F|^{O(1)} \cdot \phi^{\epsilon(1-\alpha)^{-1}K})$, where $\phi \approx 1.6181$ is the golden ratio. By our result, we can replace ϕ with 1.3803. Very recently, based on our results, Bansal and Raman further improved the upper bounds [4], cf. Section 5.

[1]Note that it is easy to see that at least half of the clauses of a boolean formula in conjunctive normal form can always be satisfied.

The paper is structured as follows. In the following section, we introduce basic definitions needed for the rest of the paper. In Section 3, we present our main algorithm, solving MaxSat in $O(|F|1.3803^K)$ time. The algorithm is based on carefully designed transformation and splitting rules for propositional formulas. In Section 4, we present a modified algorithm and obtain the time bound $O(1.1279^{|F|})$. We conclude the paper with some open questions and suggestions for future study.

2 Basic definitions

We assume familiarity with the basic notions of logic and use a similar notation as in [23]. We represent the boolean values true and false by 1 and 0, respectively. A *truth assignment* I can be defined as a set of literals that contains no pairs of complementary literals. Then for a variable x we have $I(x) = 1$ iff $x \in I$ and $I(x) = 0$ iff $\bar{x} \in I$. We only deal with propositional formulas in conjunctive normal form. These are often represented in *clause form*, i.e., as a set of clauses, where a clause is a set of literals. We will represent formulas as *multisets* of sets, since a formula might contain some identical clauses. For the satisfiability problem multiple clauses can be eliminated, but this is of course no longer true if we are interested in the number of satisfiable clauses. The formula

$$(x \vee y \vee \bar{z}) \wedge (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee z)$$

will be represented as

$$\{\{x, y, \bar{z}\}, \{x, y, \bar{z}\}, \{\bar{x}, z\}, \{\bar{y}, z\}\}.$$

Note that the outer curly brackets denote a multiset and the inner curly brackets denote sets of literals. A subformula, i.e., a subset of clauses, is called *closed* if it is a minimal subset of clauses such that no variable in this subset also occurs outside this subset in the rest of the formula. A clause that contains the same variable positively and negatively, e.g., $\{x, \bar{x}, y, \bar{z}\}$, is satisfied by every assignment. We will not allow for such clauses, but we assume that such clauses are always replaced by a special clause $\top$, which denotes a clause that is always satisfied. We call a clause containing r literals simply an *r-clause.* A formula in *2CNF* is one consisting of 1- and 2-clauses. We assume that 0-clauses do not appear in our formula, since they clearly are not satisfiable. The *length of a clause* is its cardinality, and the *length of a formula* is the sum of the lengths of its clauses. Let l be a literal occurring in a formula F. We call it an (i,j)*-literal* if the variable corresponding to l occurs exactly i times as l and exactly j times as $\bar{l}$, respectively. In analogy, we get (i^+,j)-, (i,j^+)-, and (i^+,j^+)*-literals* by replacing "exactly" with "at least" at the appropriate positions and get (i^-,j)-, (i,j^-)- and (i^-,j^-)*-literals* by replacing "exactly" with "at most." We denote the number of occurrences of a literal l in a formula F by $\#_l(F)$.

For a literal l and a formula F, let $F[l]$ be the formula originating from F by replacing all clauses containing l by $\top$ and removing $\bar{l}$ from all clauses where it occurs. To estimate the time complexity of our algorithms, the following notions are useful: $S(F)$ denotes the number of $\top$-clauses in F, and $maxsat(F)$ denotes the maximum number of simultaneously satisfiable clauses in F. We say two formulas F and G are *equisatisfiable*, if $maxsat(F) = maxsat(G)$. A formula that contains only $\top$ as its clauses is called *final*. Obviously, there is exactly one final formula in the equivalence class of equisatisfiable formulas, assuming that 0-clauses are deleted from our formula as soon as they exist.

Definition 1 A formula is called *nearly monotone* if negative literals occur only in 1-clauses. It is called a *simple formula* if it is nearly monotone and each pair of variables occurs together in one clause at most.

Definition 2 For a variable x, we say $\tilde{x}$ occurs in a clause C if $x \in C$ or $\bar{x} \in C$.

For example, $\tilde{x}$ occurs in $\{\bar{x}, y, z\}$ and in $\{x, y, z\}$, but x occurs only in $\{x, y, z\}$ and $\bar{x}$ only in $\{\bar{x}, y, z\}$. As a rule, we will use x, y, z to denote variables and l to denote a literal.

3 The algorithm

In the following, we present algorithms that solve MaxSat by mapping a formula to the unique, equisatisfiable, final formula. We distinguish two possibilities: If a formula is replaced by another formula, we speak of a *transformation rule*; if one formula is replaced by several other formulas, we speak of a *splitting rule*. The resulting formula or formulas are then solved recursively, a technique that goes back to the Davis–Putnam procedure [14].

3.1 Transformation rules

A transformation rule $\frac{F}{F'}$ replaces F by F', where F' and F are equisatisfiable, but F' is simpler. We will use the following transformation rules, whose correctness is easily confirmed. We mention in passing that many rules that apply for the Satisfiability problem do not (directly) apply for MaxSat, thus requiring new techniques for MaxSat.

Pure literal rule

$$\frac{F}{F[x]} \text{ if } x \text{ is a } (1^+, 0)\text{-literal.}$$

The correctness of the pure literal rule is easy to prove. Obviously, there is an optimal assignment I that fulfills $I(x) = 1$.

Complementary unit-clause rule

$$\frac{F}{\{\top\} \cup G} \text{ if } F = \{\{\bar{x}\}, \{x\}\} \cup G.$$

For *every* assignment $maxsat(F) = maxsat(G) + 1$.

Dominating unit-clause rule

$$\frac{F}{F[l]} \text{ if } \bar{l} \text{ occurs in } i \text{ clauses, and } l \text{ occurs at least } i \text{ times in 1-clauses.}$$

Resolution rule

$$\frac{\{\{\bar{x}\} \cup K_1, \{x\} \cup K_2\} \cup G}{\{\top, K_1 \cup K_2\} \cup G} \text{ if } G \text{ does not contain } \tilde{x}.$$

Small subformula rule

Let $F = \{\{x', y', \dots\}, \{x'', y'', \dots\}, \{x''', y''', \dots\}\} \cup G$, where G contains neither $\tilde{x}$ nor $\tilde{y}$ and $x', x'', x''' \in \{x, \bar{x}\}$ and $y', y'', y''' \in \{y, \bar{y}\}$. Then

$$\frac{F}{\{\top, \top, \top\} \cup G},$$

since there is always an assignment to x and y only that already satisfies $\{\{x', y', \dots\}, \{x'', y'', \dots\}, \{x''', y''', \dots\}\}$.

Star rule

A formula $\{\{\bar{x}_1\}, \{\bar{x}_2\}, \ldots, \{\bar{x}_r\}, \{x_1, x_2, \ldots, x_r\}, \{x_1, x_2, \ldots, x_r\}\}$ is called an *r-star.* Let F be an r-star. Then

$$\frac{F}{\{\top, \ldots, \top\}},$$

where the "$\top$-multiset" contains $r+1$ many $\top$'s.

Definition 3 A formula is *reduced* if no transformation rule is applicable, each variable occurs at least as often positively as negatively, and it contains no empty clauses. Using the above transformation rules, $\mathit{Reduce}(F)$ denotes the corresponding reduced, equisatisfiable formula.

When reducing a formula, it can be necessary to rename literals. Observe that in the rest of the paper many arguments will rely on the fact that we are dealing with a reduced formula. Particularly note that a variable in a reduced formula occurs at least 3 times.

3.2 Splitting rules

The second important technique is *splitting.* It is based on dividing the search space, i.e., the set of all possible assignments into several parts, finding an optimal assignment within each part, and then taking the best of them. Taking splitting to its extreme is to look at each single assignment by itself, which, however, leads to poor performance. Careful splits enable us to simplify the formula in some of the branches. Take, for example, the formula

$$\{\{x, y\}, \{\bar{x}, y\}, \{x, \bar{y}\}, \{\bar{x}, \bar{y}\}\}$$

and split the set of all assignments into those with $x = 0$ and those with $x = 1$. If $x = 0$, the formula becomes

$$\{\{0, y\}, \{1, y\}, \{0, \bar{y}\}, \{1, \bar{y}\}\},$$

which simplifies to

$$\{\{y\}, \top, \{\bar{y}\}, \top\}.$$

We assume in the following that the elimination of 0 or 1 in clauses is done automatically whenever it occurs; a 0 is removed from its clause and a clause

Input: A formula F
Output: An equisatisfiable formula $A(F) = \{\top, \ldots, \top\}$
Method:
$F \leftarrow \mathit{Reduce}(F)$;
if F is final **then return** F
else
  let $\tilde{x}$ be a variable that occurs in F;
  return $\max\{A(F[x]), A(F[\bar{x}])\}$
fi

Figure 1: Algorithm A to compute a final, equisatisfiable formula. Note that $\max\{A(F[x]), A(F[\bar{x}])\}$ is the multiset with the maximum number of $\top$'s.

that contains 1 is replaced by $\top$. Finally, we can simplify $\{\{y\}, \{\bar{y}\}, \top, \top\}$ with the complementary unit-clause rule to get $\{\top, \top, \top\}$. The result is $|\{\top, \top, \top\}| = 3$ for assignments with $x = 1$. Similarly, we get the result 3 for assignments with $x = 0$, so the result is "3 satisfiable clauses," which is obviously correct.

If we remove m clauses in which l occurs from F to get $F[l]$, then obviously $S(F[l]) = S(F) + m$ if we look *only* at assignments where $l = 1$. In general, however, we can at least say that

$$S(F[l]) \geq S(F) + m,$$

since an assignment where $l = 0$ could be better than all assignments where $l = 1$. Thus, a simple algorithm to compute $\mathit{maxsat}(F)$ is easily developed and can be found in Figure 1.

In the rest of this subsection, we describe our set of splitting rules. We distinguish between three basic cases, the first being easy: Either there is a variable occurring at least five times in the given formula or all variables occur three or four times in the formula and either there is one occurring exactly three times or *all* variables occur exactly four times. Clearly, this gives a complete case distinction.

There is a variable that occurs at least five times

F1 Let F be reduced.

$$\frac{F}{F[x], F[\bar{x}]} \text{ if } \tilde{x} \text{ occurs at least five times in } F.$$

We get $S(F[x]) \geq S(F) + a$ and $S(F[\bar{x}]) \geq S(F) + b$ with $a, b \geq 1$ and $a + b = 5$.

Each variable occurs three or four times and some variable occurs exactly three times

In the following we present seven splitting rules **T1–T7** and an analysis with respect to $S(F)$. These rules are applicable if F is reduced and all literals in F are $(2,1)$, $(3,1)$, or $(2,2)$-literals. Moreover, there must be at least one $(2,1)$-literal x. In what follows, we firstly describe our set of rules and secondly show that it really handles all possible cases.

T1 $$\frac{F}{F[l'], F[\bar{l'}]} \text{ if } F = \{\{\bar{x}, l', \dots\}, \{x, \dots\}, \{x, \dots\}, \{l'', \dots\}, \{l''', \dots\}, \dots\} \text{ and } l', l'', l''' \in \{y, \bar{y}\}.$$

First, assume that $l' = y$. Then, in $F[l']$ we get the first clause satisfied and since y is a $(2,1)$-literal or better, at least one of the last two clauses is also satisfied. Additionally, x then becomes a pure literal and the second and third clauses can be satisfied setting $x = 1$ by the pure literal rule. In $F[\bar{l'}]$, trivially, at least one clause is directly satisfied. Altogether, we have $S(\mathit{Reduce}(F[l'])) \geq S(F) + 4$ and $S(F[\bar{l'}]) \geq S(F) + 1$. Second, assume that $l' = \bar{y}$. Arguing in an analogous way as before, we obtain $S(\mathit{Reduce}(F[l'])) \geq S(F) + 3$ and $S(F[\bar{l'}]) \geq S(F) + 2$.

T2 $$\frac{F}{F[l], F[\bar{l}]} \text{ if } F = \{\{\bar{x}, \bar{l}, \dots\}, \{x, l, \dots\}, \{x, \dots\}, \{l, \dots\}, \dots\}.$$

Clearly, $S(\mathit{Reduce}(F[l])) \geq S(F) + 3$, since two clauses containing l are satisfied and then another clause is satisfied by the resolution rule. We also get $S(\mathit{Reduce}(F[\bar{l}])) \geq S(F) + 3$: Since at least one clause containing $\bar{l}$ is satisfied, x becomes a pure literal, thus satisfying one or two more clauses because of the pure literal rule.

T3 $\dfrac{F}{F[x], F[\bar{x}]}$ if $F = \{\{\bar{x}, y, \dots\}, \{x, y, \dots\}, \{x, \dots\}, \{\bar{y}, \dots\}, \dots\}$ and y is a $(2,1)$-literal.

Now $S(\mathit{Reduce}(F[\bar{x}])) \geq S(F)+2$, because of one directly satisfied clause and the resolution rule on $\{x, y, \dots\}$ and $\{\bar{y}, \dots\}$; $S(\mathit{Reduce}(F[x])) \geq S(F)+3$ because of two directly satisfied clauses and the resolution rule on $\{\bar{x}, y, \dots\}$ and $\{\bar{y}, \dots\}$.

T4 $\dfrac{F}{F[y], F[\bar{y}]}$ if $F = \{\{\bar{x}, y, \dots\}, \{x, \bar{y}, \dots\}, \{x, \dots\}, \{y, \dots\}, \dots\}$ and y is a $(2,1)$-literal.

Then $S(\mathit{Reduce}(F[\bar{y}])) \geq S(F)+2$, since $\{x, \bar{y}, \dots\}$ is directly satisfied and we get one more clause from the resolution rule on $\{\bar{x}, y, \dots\}$ and $\{x, \dots\}$. Also, $S(\mathit{Reduce}(F[y])) \geq S(F)+4$, since two clauses are directly satisfied and x becomes a pure literal in two clauses.

T5 $\dfrac{F}{F[x], F[\bar{x}]}$ if $F = \{\{\bar{x}\}, \{x, y, \dots\}, \{x, z\}, \{y, \dots\}, \{\bar{y}\}, \{\bar{z}\}, \dots\}$ and y and z are $(2,1)$-literals.

Then $S(\mathit{Reduce}(F[x])) \geq S(F)+4$ because of two directly satisfied clauses and the resolution rule on y satisfying another clause. Then z becomes a $(1^-, 1)$-literal and resolution, pure literal rule, or complementary unit-clause rule are applicable. Clearly, $S(F[\bar{x}]) = S(F)+1$. Observe that we did not fix the third occurrence of z—for example, $\tilde{z}$ might occur in the clause $\{x, y, \dots\}$.

T6 $\dfrac{F}{F[l], F[\bar{l}]}$ if $F = \{\{\bar{x}, \dots\}, \{x, l, \dots\}, \{x, \dots\}, \dots\}$,
l is a $(3,1)$- or $(2,2)$-literal,
and l does not occur together with $\tilde{x}$ in three clauses.

We have $S(F[l]) \geq S(F)+a$, $S(F[\bar{l}]) \geq S(F)+b$, and $a+b=4$ with $a, b \geq 1$. In $F[l]$, however, $\tilde{x}$ occurs either 1 or 2 times. Hence, $S(\mathit{Reduce}(F[l])) - S(F) \geq a+1$.

T7 $\dfrac{F}{F[l], F[\bar{l}]}$ if $F = \{\{\bar{x}, y, \dots\}, \{x, y, \dots\}, \{x, y, \dots\}, \{\bar{y}, \dots\}, \dots\}$
and there is a literal l that occurs in a clause
with $\tilde{y}$ and $\tilde{l}$ occurs also in a clause with no $\tilde{y}$.

If l occurs in two clauses together with $\tilde{y}$, then these two clauses are directly satisfied in $F[l]$ and two others by the pure literal rule (first x and then y becomes pure or vice versa). If l occurs in a clause with no $\tilde{y}$ and in a clause with $\tilde{y}$, then these two clauses are directly satisfied by $l = 1$ and at least two others following transformation rules. Altogether, $S(\mathit{Reduce}(F[l]) \geq S(F) + 4$, if l occurs in at least two clauses of F.

If, however, l occurs only in one clause of F, then $S(\mathit{Reduce}(F[l])) \geq S(F)+3$, but now $\bar{l}$ occurs in at least two clauses and consequently $S(F[\bar{l}\,]) \geq S(F) + 2$.

We get $S(\mathit{Reduce}(F[l])) \geq S(F) + 4$ and $S(\mathit{Reduce}(F[\bar{l}])) \geq S(F) + 1$ or $S(\mathit{Reduce}(F[l])) \geq S(F) + 3$ and $S(\mathit{Reduce}(F[\bar{l}])) \geq S(F) + 2$.

Lemma 4 *Let F be a reduced formula with no closed subformulas and each variable occur in three or four clauses. Moreover, let there be at least one variable that occurs in exactly three clauses. Then one of the rules **T1-T7** is applicable.*

Proof. 1. *There are only $(2,1)$-literals in F.*
Firstly, assuming that F is not nearly monotone (cf. Definition 1), we can conclude that there is some variable x that occurs negatively in a clause together with at least one other literal, say l. If $\tilde{l}$ occurs together with $\tilde{x}$ in no other clause, then **T1** applies. Otherwise, $\tilde{x}$ and $\tilde{l}$ occur together in at least two clauses. Three joint occurrences are not possible, since F is reduced and that case is covered by the small subformula rule (Subsection 3.1). Depending on the combination of positive and negative occurrences, **T2**, **T3**, or **T4** apply, as they cover all combinations.

If, however, F happens to be nearly monotone, **T5** applies: Pick any variable x. Then pick a variable y that occurs together with x in exactly one clause. Such a y exists, since otherwise x would be part of a star or in a unit-clause. Then pick some arbitrary variable z from the other clause that contains x, but not y.

2. *There is also some $(3,1)$- or $(2,2)$-literal in F.*
Find a $(2,1)$-literal x and a $(3,1)$- or $(2,2)$-literal y such that $\tilde{x}$ and $\tilde{y}$ occur in the same clause. (Such a pair is available since otherwise there would be a closed subformula that contains only $(2,1)$-literals.) Let N be the number of clauses where $\tilde{x}$ and $\tilde{y}$ occur together. If $N = 1$ then **T1** or **T6** apply. If $N = 2$ then **T6** applies. Observe that in **T6** $\tilde{y}$ may occur together with $\bar{x}$ as

well as together with x. If $N = 3$ then **T6** or **T7** apply (the existence of l in the side condition of **T7** can be assumed, since otherwise there would be a small closed subformula). $\square$

All variables occur exactly four times

In this subsection, we assume that F is reduced, it contains no closed subformulas, and that each variable occurs exactly four times.

D1 $\dfrac{F}{F[l], F[\bar{l}]}$ if $F = \{\{\bar{x}, l, \ldots\}, \{x, \ldots\}, \{x, \ldots\}, \{x, \ldots\}, \ldots\}$.

In $F[l]$, the clause $\{\bar{x}, l, \ldots\}$ is satisfied. Moreover, $F[l]$ contains the pure literal x and we can apply the pure literal rule. Clearly, it follows that $S(\mathit{Reduce}(F[l])) \geq S(F) + 4$ and $S(F[\bar{l}]) \geq S(F) + 1$.

D2 $\dfrac{F}{F[x], F[\bar{x}, y]}$ if $F = \{\{\bar{x}\}, \{x, y\}, \{x, \ldots\}, \{x, \ldots\}, \ldots\}$.

There is always an optimal assignment I with $I(x) = 1$ or with $I(x) = 0$ and $I(y) = 1$: Let I' be an optimal assignment with $I'(x) = 0$ and $I'(y) = 0$. Let I be the assignment that coincides with I', except that $I(x) = 1$. Obviously, I satisfies at least as many clauses as I' and is therefore also optimal. Hence, it suffices to examine $F[x]$ and $F[\bar{x}, y]$. We get $S(F[x]) \geq S(F) + 3$ and $S(F[\bar{x}, y]) \geq S(F) + 3$.

D3 $\dfrac{F}{F[x], F[\bar{x}]}$ if $F = \{\{\bar{x}\}, \{x, l, \ldots\}, \{x, \ldots\}, \{x, \ldots\}, \ldots\}$ and $\tilde{l}$ occurs in 1 or 2 clauses that do not contain $\tilde{x}$.

In $F[x]$, three clauses containing x are satisfied and l is a $(1,1)$-, $(1^+, 0)$-, or $(0, 1^+)$-literal. Some transformation rule satisfies at least one other clause. We get $S(\mathit{Reduce}(F[x])) \geq S(F) + 4$ and, of course, $S(F[\bar{x}]) \geq S(F) + 1$.

D4 $\dfrac{F}{F[x], F[\bar{x}]}$ if $F = \{\{\bar{x}, \ldots\}, \{\bar{x}, \ldots\}, \{x, \ldots\}, \{x, \ldots\}, \ldots\}$.

Obviously, $S(F[x]) \geq S(F) + 2$ and $S(F[\bar{x}]) \geq S(F) + 2$.

D5 $\dfrac{F}{F[y], F[\bar{y}, z], F[\bar{y}, \bar{z}]}$ if $F = \{\{\bar{x}\}, \{x, y, z\}, \{x, \ldots\}, \{x, \ldots\},$
$\{\bar{y}\}, \{y, \ldots\}, \{y, \ldots\}, \{\bar{z}\}, \{z, \ldots\}, \{z, \ldots\}, \ldots\}.$

Obviously, $S(F[y]) = S(F) + 3$ and $S(F[\bar{y}, z]) = S(F) + 4$. Finally, we get $S(\mathit{Reduce}(F[\bar{y}, \bar{z}])) \geq S(F) + 5$, since $F[\bar{y}, \bar{z}]$ contains a subformula $\{\{\bar{x}\}, \{x\}, \{x, \ldots\}, \{x, \ldots\}\}$ and, applying the complementary unit-clause rule followed by the pure literal rule, satisfies three clauses.

D6 $\dfrac{F}{F[\bar{x}], F[x, \bar{y}], F[x, y, \bar{z}_1, \bar{z}_2, \ldots, \bar{z}_6]}$

if $F = \{\{\bar{x}\}, \{x, y, \ldots\}, \{x, \ldots\}, \{x, \ldots\},$
$\{y, z_1, z_2, z_3, \ldots\}, \{y, z_4, z_5, z_6, \ldots\}, \{\bar{y}\}, \ldots\},$
F is simple, and each positive clause has size at least 4.

We have to prove the following claim: If there is an optimal assignment I for F with $I(x) = 1$, then there is an optimal assignment I' with $I'(x) = 1$ and $I'(y) = 0$ unless $I(z_1) = \ldots = I(z_6) = 0$. Let us assume that I is indeed an optimal assignment with $I(x) = 1$, but $I(z_1) = \cdots = I(z_6) = 0$ does not hold; without loss of generality let us assume $I(z_1) = 1$. Now define I' as I, but $I'(y) = 0$. When changing from I to I', the clause $\{y, z_4, z_5, z_6, \ldots\}$ may no longer be satisfied. The number of satisfied clauses does, however, not decrease, since now $\{\bar{y}\}$ is satisfied. The status of all other clauses does not change. We get $S(F[\bar{x}]) \geq S(F) + 1$, $S(F[x, \bar{y}]) \geq S(F) + 4$, and $S(F[x, y, \bar{z}_1, \bar{z}_2, \bar{z}_3, \bar{z}_4, \bar{z}_5, \bar{z}_6]) \geq S(F) + 11$.

Lemma 5 *Let F be a reduced formula. Let each variable occur in exactly four clauses and let there be no closed subformula. Then one of the rules **D1-D6** is applicable.*

Proof. If there is at least one $(2, 2)$-literal then **D4** applies. Therefore, in the following we can assume that F contains only $(3, 1)$-literals.

Let us first assume that F is not nearly monotone. Then **D1** applies.

Next, let us assume that F is nearly monotone, but not simple. Then **D3** applies.

Finally, let F be simple. If a variable x occurs in a clause of size 2 (resp. 3), then **D2** (resp. **D5**) applies. Otherwise, all variables occur positively only in clauses of size at least 4 and **D6** applies. □

The following lemma shows that the relatively inefficient rule **D4** can always be followed by something efficient.

Lemma 6 *Let F be a reduced formula, such that each variable occurs in exactly four clauses and there is some $(2,2)$-literal x. Let F contain no closed subformulas. Then $S(\mathit{Reduce}(F[x])) > S(F[x])$ or $\mathit{Reduce}(F[x])$ contains a $(2,1)$-literal. The same applies for $F[\bar{x}]$.*

Proof. Let $\tilde{y}$ occur together with x in any one clause and also in any one other clause that does not contain x. Then $\tilde{y}$ occurs in $F[x]$ between 1 and 3 times. If it occurs 1 or 2 times, the pure literal or resolution rule is applicable to $F[x]$. If y occurs 3 times in $F[x]$, it is a $(2,1)$-literal. Then it remains a $(2,1)$-literal in $\mathit{Reduce}(F[x])$ unless a reduction that increases $S(F[x])$ was carried out. Analogously, prove the same for $F[\bar{x}]$. $\Box$

3.3 Details and analysis of the algorithm

One key to an efficient algorithm for MaxSat is a good data structure to represent formulas in conjunctive normal form. For the high-level description of transformation and splitting rules, we used the representation as a multiset of sets of literals. The actual implementation of the algorithm will use a refinement of this representation. We represent literals as natural numbers. A positive literal x_i is represented as the number i and the negative literal $\bar{x}_i$ by $-i$. A clause is represented as a list of literals and a formula as a list of clauses. The $\top$-clause is represented by a special symbol. Moreover, for each variable there is an additional list of pointers that point to each occurrence of the variable in the formula.

Algorithm B constructs an equisatisfiable final formula $\{\top, \dots, \top\}$ from a formula F by using transformation and splitting rules (see Figure 2).

Lemma 7 *A formula F can be decomposed into its closed subformulas in linear time.*

Proof. Simply find the connected components in the graph whose nodes are all variables and edges connect variables that occur in the same clause. $\Box$

Lemma 8 *A formula F can be transformed into an equisatisfiable, reduced formula F' with $S(F') \geq S(F)$ in time $O(|F| + |F|(S(F') - S(F)))$.*

Input: A formula F
Output: An equisatisfiable formula $B(F) = \{\top, \dots, \top\}$
Method:
$F \leftarrow \mathit{Reduce}(F)$;
if F is final **then return** F
else if $F = F_1 \oplus F_2 \oplus \cdots \oplus F_m$ **then return** $B(F_1) \cup B(F_2) \cup \cdots \cup B(F_m)$
else if F has less than 6 unresolved clauses **then return** $A(F)$
else
  choose an applicable rule $\dfrac{F}{F_1, \dots, F_r} \in \{\mathbf{F1,T1\text{–}T7,D1\text{–}D6}\}$;
  return $\max\{B(F_1), \dots, B(F_r)\}$
fi

Figure 2: Algorithm B. Note that $F_1 \oplus F_2 \oplus \cdots \oplus F_m$ denotes the decomposition of F into closed subformulas and $\max\{B(F_1), \dots, B(F_r)\}$ is the multiset with the maximum number of $\top$'s. Among the applicable rules some rule with minimum branching number is chosen. In particular, **D4** is chosen only if no other rule is applicable.

Proof. First check if the formula is a star, subsequently check for each variable in constant time if a transformation rule applies to it and if yes, apply it in linear time.

A technique to achieve these bounds easily is a dictionary that can be constructed from a formula in linear time and that can process queries such as "Give me a variable x such that x occurs in C_1, $\tilde{x}$ occurs in C_2 and $\bar{x}$ occurs in C_2, if such a variable exists." It is sufficient to have a dictionary for queries that involve at most 4 clauses and that answers with variables that occur at most 4 times in the formula.

For example, one can check in constant time whether the small subformula rule applies to a variable x: Check that $\tilde{x}$ occurs 3 times. Find the clauses C_1, C_2, and C_3 that contain $\tilde{x}$. Then, using a dictionary, ask the query "Give me two variables that occur exactly in C_1, C_2, and C_3." The rule is applicable iff such a pair exists. $\Box$

We follow Kullmann and Luckhardt [29] (also cf. [23]) in the analysis of the running time. Algorithm B generates a *branching tree* whose nodes are labeled by formulas that are recursively processed. The children of an inner node F are computed by a transformation rule (one child) or a splitting rule

Cases	*Branching vector*	*Branching number*
F1, T1, T5, T6, T7, D1, D3	$(4, 1)$	1.3803
F1, T1, T3, T7	$(3, 2)$	1.3248
T2, D2	$(3, 3)$	1.2600
T4	$(4, 2)$	1.2721
D4	$(2, 2)$	1.4143
D5	$(5, 4, 3)$	1.3248
D6	$(11, 4, 1)$	1.3996

Table 1: Lower bounds on branching vectors for Algorithm B referring to the number of $\top$-clauses $S(F)$.

(more than one child). The *value* of a node F is $S(F)$. The values of all children of F are bigger than $S(F)$. If the children of F were computed according to a rule

$$\frac{F}{F_1, F_2, \ldots, F_r},$$

then $(S(F_1) - S(F), \ldots, S(F_r) - S(F))$ is the *branching vector* of this node. The *branching number* of a branching vector $(k_1, \ldots, k_r)$ is $1/\xi$, where ξ is the unique zero in the unit interval of the reflected characteristic polynomial

$$1 - \sum_{i=1}^{r} z^{k_i}.$$

If $\alpha_{\max}$ is the maximal branching number in the whole branching tree, then the tree contains at most $O(\alpha_{\max}^V)$ nodes, where V is the maximum value of a node in the tree [29]. Here, the number of clauses K is an upper bound on the value $S(F)$. Hence, the size of the branching tree is at most $O(\alpha_{\max}^K)$.

Lemma 9 *The branching tree of Algorithm B has $O(\beta^k)$ nodes, where k is the number of satisfiable clauses in F and $\beta = 1.3995$.*

Proof. In Table 1 we list all branching vectors and numbers corresponding to the splitting rules given in Subsection 3.2. All branching numbers are smaller than 1.3995, which is the branching number of the branching vector $(1, 4, 11)$ (rule **D6**) except the branching number $\sqrt{2} \approx 1.4143$, which belongs to nodes that are split according to rule **D4** and whose branching vector is $(2, 2)$.

By Lemma 6, however, the children of nodes with branching vector $(2, 2)$ have a branching vector of at least $(1, 4)$ or $(2, 3)$, since they are split by any one rule **T1–T7**. The combined branching number of the nodes and its children is therefore at most $(3, 6, 3, 6)$, $(3, 6, 4, 5)$, or $(4, 5, 4, 5)$. The corresponding branching numbers are 1.3980, 1.3803, and 1.3644. The largest branching number remains 1.3995 and consequently the size of the branching tree is $O(1.3995^k)$. □

Theorem 10 *The running time of Algorithm B is $O(|F| \cdot 1.3995^k)$ and (for a slight modification in the algorithm) $O(|F| \cdot 1.3803^K)$, where $|F|$ is the length of the given formula F, k is the number of satisfiable clauses in F and K is the number of clauses in F.*

Proof. The size of each formula in the branching tree does not exceed $|F|$, since transformation and splitting rules never generate longer formulas. Reducing the formula (Lemma 8), selecting and applying a splitting rule, or decomposing the formula into minimal subformulas, take time $O(|F|)$.

The size of the tree is at most 1.3995^k (Lemma 9). This proves the time bound $O(|F| \cdot 1.3995^k)$ for Algorithm B.

Let $\mu(F')$ be K minus the number of clauses in F' that are not $\top$. Then obviously $\mu(F') \geq S(F')$. Hence, the branching numbers in the tree with respect to $\mu(F')$ are at least as big as those with respect to $S(F')$. In the following we analyze Algorithm B with respect to $\mu(F')$ and get in this way a bound on the size of the branching tree in terms of K.

Except for **D4** and **D6** all branching numbers for $S(F')$ and thus for $\mu(F')$ are at most 1.3803. The branching vector for **D6** is $(1, 5, 13)$ with respect to $\mu(F')$ (yielding a branching number of 1.3400). Thus, it remains to deal with **D4**. This problem occurs only if all variables occur four times.

If there are only $(2, 2)$-literals left we *do* apply **D4**. If a formula in one of the two branches is reducible, then we have at least a $(3, 2)$ or $(2, 3)$-branch. Otherwise, it is only a $(2, 2)$-branch. However, by Lemma 6 the formulas in *both* branches contain $(2, 1)$-literals. If both formulas do not contain $(2, 2)$-literals, then **D4** will never again be used and this single use plays no role asymptotically.

Let us assume we branch on the $(2, 2)$-literal x and $F[x]$ still contains $(2, 2)$-literals. Then there is a $(2, 2)$-literal y and a $(2, 1)$-literal z such that $\tilde{y}$ and $\tilde{z}$ occur together in the same clause in $F[x]$ (unless there are closed subformulas). Then in $F[x, y]$ or $F[x, \bar{y}]$ there are between one and two clauses that contain $\tilde{z}$ and therefore one of the two formulas is reducible. In total, if we branch according to $F[\bar{x}]$, $F[x, y]$, and $F[x, \bar{y}]$, we get a branching vector of $(2, 5, 4)$ or $(2, 4, 5)$. The corresponding branching number is 1.3803. This settles the case that there are only $(2, 2)$-literals in the formula.

If there are also $(3, 1)$-literals then either **D1** is applicable or all $(3, 1)$-literals occur negatively only in unit-clauses. If that is the case and there are also $(2, 2)$-literals then there is also a clause that contains a $(3, 1)$-literal x and some $(2, 2)$-literal l (otherwise there would exist a closed subformula). Then rule **D3** can be applied. If, however, no $(2, 2)$-literal exists, then **D4** is not applicable and therefore some other rule must be applicable. The branching numbers of all rules except **D4** are, however, at most 1.3803. □

From the parameterized complexity point of view, assuming a parameter value $k < K$, the following corollary is of interest.

Corollary 11 *To determine an assignment satisfying at least k clauses of a boolean formula F in CNF takes $O(k^2 \cdot 1.3995^k + |F|)$ steps.*

Proof. Mahajan and Raman show that an algorithm solving MaxSat in $O(|F| \cdot \gamma^k)$ steps can be transformed into an algorithm that solves the above problem in $O(k^2\gamma^k + |F|)$ steps [30]. □

Corollary 11 improves Theorem 7 of Mahajan and Raman [30] by decreasing the exponential factor from $\phi^k \approx 1.6181^k$ to 1.3995^k. Analogously, the running time for Max$q$Sat is improved to $O(qk \cdot 1.3995^k + |F|)$.

4 A bound with respect to formula length

In this section, we analyze the running time of MaxSat algorithms with respect to the length of a formula. In the last section, the value of a node F' in the branching tree was $S(F')$. In this section, it will be $|F| - |F'|$, i.e., the reduction in length relative to the root F. We define the *length* of a formula simply as the sum over the size of its clauses, where clause size is the number of literals occurring in a clause. For the analysis of Algorithm B in terms of the reduction in length, note that applying the resolution rule reduces the length by 2.

As in the previous section, our main concern is the size of the search tree. Hence, the transformation rules are "harmless," because they avoid a branching of the recursion. On the other hand, due to the dominating unit-clause rule (which is a transformation rule), we can often assume that when applying a splitting rule to a reduced formula, a satisfied clause is *not* a unit-clause, because this would have been handled by the dominating unit-clause rule to be applied before. With the dominating unit-clause rule, we can often assume that a satisfied clause is not a unit-clause. Hence, the length is reduced by at least 2: If x is an (a, b)-literal in a reduced formula F, then x occurs at most $b-1$ times in a unit-clause. Hence,

$$|F| - |F[x]| \geq 2(a - (b-1)) + b - 1 = 2a + 1.$$

In what follows, we analyze the various splitting rules (cf. Subsection 3.2) with respect to the formula length. With regards to **F1**, we have the following. If x is a $(4, 1)$-literal, the length reduction for $F[x]$ is at least 9, since the positive occurrences of x can be assumed to be in non-unit-clauses only—otherwise the dominating unit-clause rule would apply; for a $(3, 2)$-literal it is at least 7. Clearly, for $F[\bar{x}]$ we trivially get a length reduction of at least 5 in both cases. This proves the **F1**-entry of Table 2 that shows the branching vectors of Algorithm B in terms of the length reduction $|F| - |F'|$.

Now, we come to the "**T**-rules." Please refer to Subsection 3.2. For our analysis, it is important to note that considering a $(2, 1)$-literal x, we know (due to the dominating unit-clause rule) that no positive occurrence of x can be in a unit-clause.

As to **T1**, a direct analysis would give a branching vector that is too bad. So we add two new splitting rules **T1′** and **T1″** and modify the algorithm in such a way that **T1** is only applied if neither **T1′** nor **T1″** applies:

T1′ $\dfrac{F}{F[x], F[\bar{x}]}$ if $F = \{\bar{x}, l\}, \{x, \ldots\}, \{x, \ldots\}, \{\bar{l}\}, \{l, \ldots\}$.

Here, $|F| - |Reduce(F[x])| \geq 9$ and $|F| - Reduce(|F[\bar{x}])| \geq 4$.

T1″ $\dfrac{F}{F[y], F[\bar{y}]}$ if $F = \{\bar{x}, l, \ldots\}, \{x, \ldots\}, \{x, \ldots\}, \{\bar{l}\}, \{l, \ldots\}$.

Here, $|F| - |Reduce(F[y])| \geq 10$ and $|F| - |F[\bar{x}]| \geq 3$.

It remains to analyze **T1** for cases when neither **T1′** nor **T1″** applies. Due to **T1′** and **T1″** we may assume that none of the five clauses given in rule **T1** is a unit-clause. Setting l' to true, x becomes a pure literal and will be eliminated. Hence, at least 8 literals are eliminated. Setting l' to false, at least 4 literals are eliminated (without applying any transformation rule). Altogether, the branching vector is $(8, 4)$.

In **T2**, we branch into $F[l]$ and $F[\bar{l}]$. As to $F[l]$, we consider two subcases: If l is a $(2, 1)$-literal, then the fourth clause $\{l, \dots\}$ in **T2** is not a unit-clause (dominating unit-clause rule). Thus, setting l to true directly eliminates 5 literals from F. Afterwards, the resolution rule applies to the first and third clause, eliminating two further occurrences of $\bar{x}$ and x. If l is a $(3, 1)$- or $(2, 2)$-literal, then setting l to true trivially eliminates 5 literals and afterwards again the resolution rule applies. Hence, in both cases the length reduction is at least 7. As to $F[\bar{l}]$, it is easy to observe that at least 7 literals can be eliminated, additionally making use of the pure literal rule for x and noting that the third clause $\{x, \dots\}$ cannot be a unit-clause.

In **T3**, an analysis similar to **T2** can be done, yielding branching vector $(7, 6)$. The same holds for **T4**, where we obtain branching vector $(8, 6)$.

The analysis for **T5**, where we branch into $F[x]$ and $F[\bar{x}]$, again is a bit more involved: If x occurs in a clause of size exactly 2, e.g., together with y, then $|F| \geq |\mathit{Reduce}(F[x])| + 9$, since all x, y, z disappear. In $F[\bar{x}]$, exactly 3 occurrences of x disappear. Then y is a unit-clause and the dominating unit-clause rule gives $y = 1$, reducing the length by at least 4. Hence, the branching vector is at least $(9, 7)$. The case where x and z occur in a clause of size 2 is similar. If x only occurs in clauses of size at least three, then the branching vector is at least $(10, 3)$.

In **T6**, branching into $F[l]$ and $F[\bar{l}]$, l is a $(3, 1)$-literal or a $(2, 2)$-literal. In the first case, setting l to true directly eliminates at least 7 literals, since all positive occurrences have to be in non-unit-clauses. At least one further literal is eliminated using resolution on x. On the other hand, setting l to false trivially eliminates 4 literals, yielding branching vector $(8, 4)$ for this case. In the second case, setting l to true and subsequently applying the resolution rule for x clearly eliminates 7 literals. Setting l to false, we know that at least one occurrence of $\bar{l}$ has to be in a non-unit-clause and, therefore, at least 5 literals are eliminated. The branching vector is, therefore, $(7, 5)$.

In **T7**, we branch into $F[l]$ and $F[\bar{l}]$. No matter where l occurs in the given formula, setting l to true we subsequently can always eliminate all occurrences of $\tilde{x}$ and $\tilde{y}$. Hence, altogether at least 10 literals are eliminated. On the other hand, setting l to false trivially eliminates 3 literals, resulting in the branching vector $(10, 3)$.

Eventually, it remains to analyze the "**D**-rules." Again refer to Subsection 3.2 for the presentation of the splitting rules. In D1, we branch into $F[l]$

and $F[\bar{l}]$. Setting l to true, x becomes a pure literal, and so we can eliminate at least 8 literals. Setting l to false, we trivially eliminate 4 literals. The branching vector is $(8,4)$.

In **D2**, we branch into $F[x]$ and $F[\bar{x}, y]$. Setting x to true, due to the dominating unit-clause rule, we can additionally observe that the positive occurrences of x all are in non-unit-clauses. Hence, at least 7 literals are eliminated. Setting x to false and y to true, clearly at least 8 literals are eliminated. The branching vector is $(7,8)$.

In **D3**, we branch into $F[x]$ and $F[\bar{x}]$. Setting x to true, we know that two or three ocuurrences of $\tilde{l}$ are eliminated. Hence, applying resolution or some other transformation rule, the remaining occurrence(s) of l can be eliminated. Altogether, at least 8 literals are eliminated in this case. Setting x to false, we trivially eliminate 4 literals. The branching vector is $(8,4)$.

As to **D4**, this rule in general would give a branching vector that is too bad. Hence, we introduce a new splitting rule:

D4′ $\dfrac{F}{F[l_2], F[\bar{l}_2]}$ if $F = \{\{l_1, \ldots\}, \{l_1, l_2, \ldots\}, \{\bar{l}_1, \ldots\}, \{\bar{l}_1\},$
l_1 is a $(2,2)$-literal, and l_2 is a $(3,1)$- or $(2,2)$-literal

Here, $|F| - |\mathit{Reduce}(F[l_2])| \geq 8$ and $|F| - |F[\bar{l}_2]| \geq 4$.

We modify the algorithm in such a way, that **D4′** is applied whenever possible and **D4** is applied for a $(2,2)$-literal x only if $\tilde{x}$ does not occur in a unit-clause. In **D4'**, setting l_2 to true, the dominating unit-clause rule applies to $\tilde{l_1}$, eliminating also all occurrences of $\tilde{l_1}$. Setting l_2 to false, we trivially eliminate 4 literals. The branching vector is $(8,4)$. Now, **D4** only has to be applied if $\tilde{x}$ does not occur in a unit-clause. This trivially implies the branching vector $(6,6)$.

In **D5**, we branch into $F[y]$, $F[\bar{y}, z]$, and $F[\bar{y}, \bar{z}]$. Setting y to true eliminates 8 literals, because the positive occurrences of y are once in a 3-clause and two times in at least 2-clauses—otherwise the dominating unit-clause rule would apply. Setting y to false and z to true eliminates at least 11 literals, because in complete analogy to y, the positive occurrences of z are once in a 3-clause and two times in at least 2-clauses. Setting y and z to false, we obtain a unit-clause $\{x\}$ and thus can apply the dominating unit-clause rule for x. Since in complete analogy to y and z, the positive occurrences of x are once in a 3-clause and two times in at least 2-clauses, altogether at least 14 literals can be eliminated. The branching vector is $(8,11,14)$.

Case	*Branching vector*	*Branching number*
F1	$(9, 5)$ or $(7, 5)$	1.1074 or 1.1238
T1′	$(9, 4)$	1.1193
T1″	$(10, 3)$	1.1273
T1	$(8, 4)$ 1.1279	
T2	$(7, 7)$	1.1041
T3	$(7, 6)$	1.1128
T4	$(8, 6)$	1.1049
T5	$(10, 3)$ or $(9, 7)$	1.1273 or 1.0911
T6	$(8, 4)$ or $(7, 5)$	1.1279 or 1.1238
T7	$(10, 3)$	1.1273
D1	$(8, 4)$	1.1279
D2	$(7, 8)$	1.0970
D3	$(8, 4)$	1.1279
D4	$(6, 6)$	1.1225
D4′	$(8, 4)$	1.1279
D5	$(8, 11, 14)$	1.1082
D6	$(4, 16, 34)$	1.0918

Table 2: Lower bounds on branching vectors for Algorithm B in terms of the length reduction $|F| - |F'|$.

In **D6**, we branch into $F[\bar{x}]$, $F[\bar{x}, \bar{y}]$, and $F[x, y, \bar{z}_1, \bar{z}_2, \bar{z}_3, \bar{z}_4, \bar{z}_5, \bar{z}_6]$. Setting x to false, we trivially eliminate 4 literals. Setting x to true and y to false, we can eliminate at least 16 literals: Due to the assumptions made, each positive clause has size at least four. This implies the elimination of at least 13 literals with respect to the first four clauses given in the rule. The three occurrences of $\tilde{y}$ in the last three clauses given in the rule means the elimination of 3 more literals. Setting x and y to true and z_1, z_2, z_3, z_4, z_5, and z_6 to false implies the elimination of at least 34 literals: Setting x and y to true, we clearly eliminate at least 22 literals. Since the formula is simple, at most two occurrences of $z_1, \ldots, z_6$ can be in the clauses given in the rule. Hence, each of $z_1, \ldots, z_6$ has to occur at least two times outside the clauses given in rule **D6**. Hence, at least 12 more literals are eliminated. Altogether, the branching vector is $(4, 16, 34)$.

We summarize our findings in the following theorem.

Theorem 12 *We can solve* MaxSat *in time $O(1.1279^{|F|})$, where $|F|$ is the length of the formula.*

Proof. Take Algorithm B with preference of **D4′** over **D4**. The above analysis (also cf. Table 2) shows that the size of the branching tree is at most $O(1.1279^{|F|})$. Each node of the tree is processed in linear time, i.e., it takes $O(|G|)$ for a node that processes a formula G. The total running time is then

$$\sum_G O(|G|),$$

where the sum is taken over all formulas that are processed in all $O(1.1279^{|F|})$ nodes of the branching tree. While $|G|$ is large near the top of the tree, $|G| = O(1)$ holds for all nodes near the leaves. Since almost all nodes are near the leaves, the total running time is $O(1.1279^{|F|})$. It is easy to make this into a rigorous mathematical argument [33]. □

Theorem 12 should be compared to the best known result for the "simpler" satisfiability problem obtained by Hirsch [23]. He proves the time bound $O(1.0758^{|F|})$.

Corollary 13 *We can solve* Max2Sat *in time $O(1.2722^K)$, where K is the number of clauses in the given formula in 2CNF.*

Proof. Simply observe that for a formula in 2CNF we have $|F| = 2K$ and apply Theorem 12. □

5 Conclusion

Using refined techniques of Davis–Putnam character, we improved previous results on the worst case complexity of MaxSat, one of the fundamental optimization problems [6, 12]. A set of transformation and splitting rules that are more complicated than in previous work are the major factors of this improved performance. This faster algorithm also has applications for approximation and parameterized algorithms for MaxSat.

To improve the upper time bounds further is an interesting open problem. Very recently, building up on our work and increasing the number of case distinctions, Bansal and Raman [4] reported progress in this direction: They showed that MaxSat can be solved in $O(1.341294^K|F|)$ or

$O(k^2 1.380278^k + |F|)$ time. With respect to length, they give the new upper bound $O(1.105729^{|F|}|F|)$. A more special question is whether Max2Sat, Max3Sat, or even Max$q$Sat can be solved faster than the general problem. Note that so far the given faster algorithm for Max2Sat results only from the relationship between the length of a formula and the number of clauses, this not being due to a different algorithm. With regards to Max2Sat, recent research indicates that it can be solved faster than MaxSat in general [20, 24]. It is open whether or not Max2Sat can be solved in less than 2^n steps, where n is the number of variables. In addition, good upper bounds are still lacking for many similar problems, e.g., MaxCut [30] and constraint satisfaction (MaxCSP) [46]. A completely different question would be to investigate the performance of our algorithms in practice and whether or not they may also serve as a basis for efficient heuristic algorithms. As for Max2Sat, recent experimental work shows that the developed exact algorithms also do perform well in practice [19, 20].

Acknowledgments. We thank Jarik Nešetřil and DIMATIA, Prague, for inviting the second author to Prague, where essential parts of this work were done. In particular, we are grateful to Pavel Valtr (DIMATIA) for initial discussions on how to improve existing MaxSat upper bounds. Finally, we wish to thank Jens Gramm and two anonymous referees for many fruitful remarks improving presentation and correcting small mistakes.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33d IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[2] T. Asano, K. Hori, T. Ono, and T. Hirata. A theoretical framework of hybrid approaches to MAX SAT. In *Proceedings of the 8th International Symposium on Algorithms and Computation*, number 1350 in Lecture Notes in Computer Science, pages 153–162. Springer-Verlag, December 1997.

[3] T. Asano and D. P. Williamson. Improved approximation algorithms for MAX SAT. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[4] N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science, Chennai, India, December 1999. Springer-Verlag. To appear.

[5] R. Battiti and M. Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2:Article 2, 1997.

[6] R. Battiti and M. Protasi. Approximate algorithms and heuristics for MAX-SAT. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 77–148. Kluwer Academic Publishers, 1998.

[7] R. Beigel and D. Eppstein. 3-Coloring in time $o(1.3446^n)$: A no MIS algorithm. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 444–452, 1995.

[8] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1999.

[9] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 54:465–474, 1997.

[10] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Ascona, Switzerland, June 1999. Springer-Verlag. To appear.

[11] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, August 1998.

[12] P. Crescenzi and V. Kann. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, 29(4):90–97, 1998.

[13] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. Approximation algorithms for Max SAT: A better performance ratio at the cost of a longer running time. Technical Report PDMI preprint 14/1998, Steklov Institute of Mathematics at St. Petersburg, 1998. Available at http://logic.pdmi.ras.ru/∼hirsch/.

[14] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[15] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[16] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computatinal intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *AMS-DIMACS*, pages 49–99. AMS, 1999.

[17] U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *3d IEEE Israel Symposium on the Theory of Computing and Systems*, pages 182–189, 1995.

[18] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1979.

[19] J. Gramm. Exact algorithms for Max2Sat: and their applications. Diplomarbeit, Universität Tübingen, 1999. Available through http://www-fs.informatik.uni-tuebingen.de/∼niedermr/publications/index.html.

[20] J. Gramm and R. Niedermeier. Faster exact solutions for Max2Sat. Manuscript, submitted for publication, August 1999.

[21] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[22] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[23] E. A. Hirsch. Two new upper bounds for SAT. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 521–530, 1998.

[24] E. A Hirsch. A new algorithm for MAX-2-SAT. Technical Report TR99-036, ECCC Trier, 1999.

[25] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* Boston, MA: PWS Publishing Company, 1997.

[26] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Ser. Discr. Math. Theor. Comput Sci.* AMS, 1996.

[27] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.

[28] O. Kullmann. New methods for 3-SAT decison and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

[29] O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. 1997. Submitted to *Information and Computation.* Available at http://mi.informatik.uni-frankfurt.de/people/kullmann/papers.html.

[30] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.

[31] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[32] R. Niedermeier. Some prospects for efficent fixed parameter algorithms (invited paper). In B. Rovan, editor, *Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, number 1521 in Lecture Notes in Computer Science, pages 168–185. Springer-Verlag, 1998.

[33] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. Technical Report TUM-I9913, Institut für Informatik, Technische Universität München,Fed. Rep. of Germany, June 1999. Submitted to *Information Processing Letters*.

[34] R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.

[35] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[36] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[37] R. Paturi, P. Pudlák, M. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 628–637, 1998.

[38] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 566–574, 1997.

[39] V. Raman. Parameterized complexity. In *Proceedings of the 7th National Seminar on Theoretical Computer Science (Chennai, India)*, pages I–1–I–18, June 1997.

[40] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[41] I. Schiermeyer. Pure literal look ahead: An $O(1.497^n)$ 3-Satisfiability algorithm. Technical Report 96-230, Universität Köln, 1996.

[42] R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, 10(3):456–464, 1981.

[43] R. E. Tarjan and A. E. Trojanowski. Finding a Maximum Independent Set. *SIAM Journal on Computing*, 6(3):537–550, 1977.

[44] L. Trevisan, G. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 617–626, 1996.

[45] R. J. Wallace. Enhancing maximum satisfiability algorithms with pure literal strategies. In *11th Canadian Conference on Artificial Intelligence, AI'96*, volume 1081 of *Lecture Notes in Artificial Intelligence*, pages 388–401. Springer, 1996.

[46] R. J. Wallace and E. C. Feuder. Comparative studies of constraint satisfaction and Davis–Putman algorithms for maximum satisfiability problems. In Johnson and Trick [26], pages 587–615.

[47] J. Wiedermann. Fast simulation of nondeterministic Turing machines with application to the Knapsack problem. *Computers and Artificial Intelligence*, 8(6):591–596, 1989.

[48] M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT. In *Proceedings of the 4th Annual International Computing and Combinatorics Conference (COCOON)*, number 1449 in Lecture Notes in Computer Science, pages 105–116. Springer-Verlag, 1998.

[49] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17:475–502, 1994.

[50] W. Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 155:277–288, 1996.

[51] U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to MAX CUT and other problems. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 679–687, 1999.