New Upper Bounds for MaxSat

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@informatik.tu-muenchen.de

Abstract

We describe exact algorithms that provide new upper bounds for the Maximum Satisfiability problem (MaxSat). We prove that MaxSat can be solved in time $O(|F| \cdot 1.3972^K)$, 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 exponential time approximation algorithm by Dantsin *et al.* uses an exact algorithm for MaxSat as a building block and is therefore also improved.

*Supported by a Feodor Lynen fellowship of the Alexander von Humboldt-Stiftung, Bonn, and the Center for Discrete Mathematics, Theoretical Computer Science and Applications (DIMATIA), Prague. Author's address in 1998: DIMATIA MFF UK, Charles University, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic.

1 Introduction

The *maximum satisfiability problem* (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 base for solutions of major problems in AI and combinatorial optimization [14, 30, 31]. It also has been a subject of the second DIMACS challenge [17]. It has been termed 'a paradigmatic problem for the "algorithmic engineering" and scientific testing and tuning effort' [3]. In this paper, we improve the worst case bounds for exact algorithms solving MaxSat.

MaxSat cannot be solved in polynomial time unless $P = NP$, since it generalizes the satisfiability problem [12, 23]. Basically three approaches were suggested to overcome MaxSat's intractability implied by its NP-hardness:

1. *Approximation algorithms* [6, 10, 33]. If we do not demand the solution to be exact, but 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 0.758 [13, 21] and a randomized one with approximation factor 0.770 [2]. There does not exist, however, a polynomial time approximation algorithm with an approximation factor arbitrarily close to 1 unless $P = NP$ [1]. For Max3Sat the approximation factor cannot be better than $\frac{7}{8}$ and for Max2Sat not better than 0.955 [15]. On the other hand, Max3Sat can be approximated within a factor of 0.801 [29] and Max2Sat within a factor of 0.931 [11].

   Dantsin *et al.* show how to improve the MaxSat approximation factor of 0.770 arbitrarily close to 1 using an exponential time algorithm [7].

2. *Heuristics.* There is a large body of work on exact algorithms for MaxSat that use clever heuristics to be fast on instances occurring in practice, e.g., [3, 4, 30, 31, 32]. 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* [9, 27]. 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 [5] proved that parameterized Max$q$Sat is "fixed parameter tractable," implying that every problem in the optimization class *MaxSNP* [24] is also fixed parameter tractable. Mahajan and Raman [20] introduced a more meaningful 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 [20] 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 is also the so far best worst case time bound for an exact MaxSat algorithm and was independently achieved by Dantsin *et al.* [7], using it as a basis for developing upper bounds for approximation algorithms.

So far, worst case complexity analysis has mainly focused on the classical Sat problem, e.g., [16, 18, 19, 22, 26, 25, 28, 34], but to our knowledge comparatively little work has been done for MaxSat [7, 20]. Our results provide new worst case upper bounds for MaxSat. Besides improving previous work [7, 20], our result applies to *all* the above three points: It improves an existing approximation algorithm [7], improves known fixed parameter tractability results [5, 20], and finally, provides an algorithm that may serve as basis for heuristic approaches in solving MaxSat.

Our main results are as follows. We prove that MaxSat can be solved in time $O(|F| \cdot 1.3972^K)$, 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 $|F| \leq 2K$.

The paper is organized as follows. In the next section, we introduce basic definitions needed in the rest of the paper. In Section 3, we present our main algorithm, solving MaxSat in $O(|F|1.3972^K)$ 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 directions for future work.

2 Basic definitions

We assume familiarity with the basic notions of logic and use a similar notation as in [16]. 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 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 itself 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 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. 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 l occurs exactly i times positively and exactly j times negatively in F. In analogy, we get (i^+, j)-, (i, j^+)-, and *(i^+, j^+)-literals* by replacing "exactly" by "at least" at the appropriate positions and get (i^-, j)-, (i, j^-)- and *(i^-, j^-)-literals* by replacing "exactly" by "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. To estimate the time complexity of our algorithms, the following notion is useful: $S(F)$ denotes the number of $\top$-clauses in F, $maxsat(F)$ 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.

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 at most one clause.

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 only occurs in $\{x, y, z\}$ and $\bar{x}$ only occurs 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 [8].

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 easy to check.

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 with $I(x) = 1$, as $I(x)$ has no influence on G, but satisfies all clauses outside of G.

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

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

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 $F = \{\{x', y', \ldots\}, \{x'', y'', \ldots\}, \{x''', y''', \ldots\}\} \cup G$, 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 $\{\{x', y', \ldots\}, \{x'', y'', \ldots\}, \{x''', y''', \ldots\}\}$.

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 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 literal occurs at least as often positively as negatively, and it contains no empty clauses. Using the above transformation rules, *Reduce*(F) denotes the corresponding reduced, equisatisfiable formula.

Observe that in the rest of the paper many arguments will rely on the fact that we are dealing with a reduced formula.

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

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 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.

and a clause 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 $S(F) = S(F[l]) + m$ if we look *only* on assignments with $l = 1$. In general, however, we can at least say that

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

since some assignment with $l = 0$ could be better than all assignments with $l = 1$. So,

$$S(F) = \max\{S(F[l]) + \#_l(F), S(F[\bar{l}]) + \#_{\bar{l}}(F)\}.$$

Thus, a simple algorithm to compute $S(F)$ appears as follows (see Figure 1).

Some variable occurs exactly three times

In the following we present five 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.

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

In $F[y']$, either at least one clause (if $y' = \bar{y}$) or at least two clauses (if $y' = y$) are directly satisfied and then x is a pure literal. In $F[\bar{y}']$, two or one clauses are directly satisfied. Hence, we have $S(\mathit{Reduce}(F[y'])) \geq S(F) + 3$ and $S(F[\bar{y}']) \geq S(F) + 2$ or $S(\mathit{Reduce}(F[y'])) \geq S(F) + 4$ and $S(F[\bar{y}']) \geq S(F) + 1$.

T2 $\dfrac{F}{F[l], F[\bar{l}]}$ 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. Here, we 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 $F = \{\{\bar{x}\}, \{x, y, \ldots\}, \{x, z, \ldots\}, \{y, \ldots\}, \{\bar{y}\}, \{\bar{z}\}, \ldots\}$
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, $S(F[\bar{x}]) = S(F) + 1$.

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

We have $S(F[l]) \geq S(F) + a$ and $S(F[\bar{l}]) \geq S(F) + b$ with $a + b = 4$. 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 $F = \{\{\bar{x}, y, \ldots\}, \{x, y, \ldots\}, \{x, y, \ldots\}, \{\bar{y}, \ldots\}, \ldots\}$,
x is a $(2,1)$-literal, 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 by 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 and each variable occurs in three or four clauses. Moreover, there is 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.
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 another 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 would be 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. (We can find such a pair 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. If $N=3$ then **T6** or **T7** apply (the existence of l in the side condition of **T7** is easy to see, since otherwise there would be a small closed subformula). □

All variables occur exactly four times

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

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

In $F[l]$, the clause $\{\bar{x},l,\dots\}$ is satisfied. Moreover, $F[l]$ contains the pure literal x and we can apply the pure literal rule. 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 $F=\{\{\bar{x}\},\{x,y\},\{x,\dots\},\{x,\dots\},\dots\}$.

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 another 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 $\dfrac{F}{F[y], F[\bar{y}, z], F[\bar{y}, \bar{z}]}$

$$\text{if } F = \{\{\bar{x}\}, \{x, y, z\}, \{x, \ldots\}, \{x, \ldots\}, \{\bar{y}\}, \{y, \ldots\}, \{y, \ldots\}, \{\bar{z}\}, \{z, \ldots\}, \{z, \ldots\}, \ldots\}.$$

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

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

$$\text{if } F = \{\{\bar{x}\}, \{x, y, \ldots\}, \{x, \ldots\}, \{x, \ldots\}, \{y, z_1, z_2, z_3, \ldots\}, \{y, z_4, z_5, z_6, \ldots\}, \{\bar{y}\}, \ldots\}$$

and 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 $I(z_1) = \cdots = I(z_6) = 0$. 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 $\{y, z_4, z_5, z_6, \dots\}$ might 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. Each variable occurs in exactly four clauses and there is 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 some 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 $S(Reduce(F[x])) > S(F[x])$ or $Reduce(F[x])$ contains a $(2,1)$-literal. The same holds for $F[\bar{x}]$.*

Proof. Let $\tilde{y}$ occur together with x in some clause and also in some 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 $Reduce(F[x])$ unless a reduction that increases $S(F[x])$ was carried out. Analogously, prove the same for $F[\bar{x}]$. □

There is a literal that occurs at least five times

F1 Let F be reduced.

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

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

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. Moreover, for each variable there is additionally a list of pointers that point to each occurrence of the variable in the formula.

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

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. □

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 splitting rule

$$\frac{F}{F_1, \dots, F_r} \in \{\mathbf{T1}\text{–}\mathbf{T7}, \mathbf{D1}\text{–}\mathbf{D6}, \mathbf{F1}\};$$

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. **D4** is chosen only if no other rule is applicable.

Proof. First check if the formula is a star, then check subsequently 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 like "Give me a variable x such that x occurs in C_1, $\tilde{x}$ occurs 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, you 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 $\tilde{x}$. Use a dictionary and 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. $\square$

We follow Kullmann and Luckhardt [19] (also cf. [16]) 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 (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 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 $\alpha_{\max}^V$ nodes, where V is the maximum value of a node in the tree [19]. 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 $\alpha_{\max}^K$.

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

Proof. 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 $\sqrt{2} \approx 1.414$, 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 number of at most 1.3803, since they are split by some rule **T1**–**T7**. Consequently, in each path of the branching tree there are at least as many nodes with branching number 1.3803 as with branching number $\sqrt{2}$. Furthermore there can be additional nodes in the path with branching number at most 1.3995. It is easy to see that the size of the tree is at most the maximum of 1.3995^k and β^k, where $\beta = 1.3972$ is the geometric mean of $\sqrt{2}$ and 1.3803 because of $S(F') \leq k$ holds for all nodes in the tree. $\square$

Theorem 10 *The running time of Algorithm B is $O(|F| \cdot 1.3995^k)$ and $O(|F| \cdot 1.3803^K)$, 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). 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 most as big as those with respect to $S(F')$.

Except for **D4** and **D6** all branching numbers for $S(F')$ and thus for $\mu(F')$ are consequently at most 1.3803. The branching vector for **D6** is $(1, 5, 13)$ with respect to $\mu(F')$. Since $\mu(F')$ is bounded by K we get analogously to the proof of Lemma 9 that the size of the branching tree is at most 1.3972^K. □

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 function 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 [20]. □

Corollary 11 improves Theorem 7 of Mahajan and Raman [20] 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 the length of a formula

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. For the analysis of Algorithm B in terms of the reduction in length, note that applying the resolution rule reduces the length by 2.

With the dominating unit-clause rule, we can often assume that a satisfied clause is not a unit-clause. Hence, the length is reduced at least by 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, $|F| - |F[x]| \geq 2a + 1$. For example, look at **F1** (Subsection 3.2). If x is a $(4, 1)$-literal, the length reduction for $F[x]$ is at least 9; for a $(3, 2)$-literal it is at least 7. This proves the **F1**-entry of Table 1 that shows the branching vectors of Algorithm B in terms of the length reduction $|F| - |F'|$.

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. For **D4'**, we thus get the entry $(8, 5)$ in Table 1. All other entries can be checked (not so) easily, only **T5** is a little bit more involved: If x occurs in a clause of size 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 sets $y = 1$ reducing the length by at least 4. Hence, the branching vector is at least $(9, 7)$. The case that 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 $(10, 3)$.

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

Case	*Branching vector*	*Branching number*
T1	$(9,4)$ or $(8,5)$	1.1193 or 1.1148
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'	$(8,5)$	1.1148
D5	$(8,11,14)$	1.1082
D6	$(4,16,32)$	1.0930
F1	$(9,5)$ or $(7,5)$	1.1074 or 1.1238

Table 1: Lower bounds on branching vectors for Algorithm B in terms of the length reduction $|F| - |F'|$.

Proof. Take Algorithm B with preference of **D4'** over **D4**. The above analysis shows that the size of the branching tree is at most $1.1279^{|F|}$. Each node of the tree is processed in linear time. □

Theorem 12 should be compared to the best known result for the "simpler" Satisfiability problem obtained by Hirsch [16]. He proves the time bound $O(1.0758^{|F|})$.

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. A set of transformation and splitting rules that are more complicated as in previous work are the main ingredients for the improved performance. This faster algorithm has also applications for approximation and parameterized algorithms for MaxSat.

To improve the upper time bounds further is an interesting open problem. In particular, can Max2Sat, Max3Sat, or even Max$q$Sat be solved faster than the general problem? Note that the presented faster algorithm for Max2Sat results only from the relationship between length of a formula and the number of clauses, but not owing to a different algorithm. It is also open whether 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 as, e.g., MaxCut [20] and constraint satisfaction (MaxCSP) [31]. A completely different question is to investigate the performance of our algorithms in practice and whether they may also serve as a basis for efficient heuristic algorithms.

Acknowledgment. We are grateful to 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 for initial discussions on how to improve existing MaxSat upper bounds.

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 Conference on Foundations of Computer Science*, pages 14–23, 1992.

[2] T. Asano. Approximation algorithms for MAX SAT: Yannakakis vs. Goemans-Williamson. In *5th IEEE Israel Symposium on the Theory of Computing and Systems*, pages 24–37, 1997.

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

[4] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. 1997. To appear in *Journal of Combinatorial Optimization*.

[5] 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.

[6] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available as http://www.nada.kth.se/theory/problemlist.html, April 1997.

[7] 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. Submitted for publication, 1998.

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

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1998. To appear.

[10] D. S. Hochbaum (ed.). *Approximation algorithms for NP-hard problems.* Boston, MA: PWS Publishing Company, 1997.

[11] 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.

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

[13] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

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

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

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

[17] 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.

[18] O. Kullmann. New methods for 3-SAT decison and worst-case analysis. 1998. To appear in *Theoretical Computer Science*.

[19] O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. 1997. Submitted to *Information and Computation*.

[20] M. Mahajan and V. Raman. Parametrizing above guaranteed values: MaxSat and MaxCut. Technical Report TR97-033, ECCC Trier, 1997.

[21] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of the 36th IEEE Conference on Foundations of Computer Science*, pages 162–169, 1995.

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

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

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

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

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

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

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

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

[30] 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*. Springer, 1996.

[31] R. J. Wallace and E. C. Feuder. Comparative studies of constraint satisfaction and Davis–Putman algorithms for maximum satisfiability problems. In Johnson and Trick [17], pages 587–615.

[32] M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT. To appear in Proc. of COCOON'98 (published in Springer, *LNCS*), 1998.

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

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