LCNS, Vol 1644, pp. 575–584, Springer 1999

New Upper Bounds for MaxSat

Rolf Niedermeier[1]* and Peter Rossmanith[2]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
niedermr@informatik.uni-tuebingen.de
[2] Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80290 München, Fed. Rep. of Germany
rossmani@in.tum.de

Abstract. Given a boolean formula F in conjunctive normal form and an integer k, is there a truth assignment satisfying at least k clauses? This is the decision version of the Maximum Satisfiability (MaxSat) problem we study in this paper. We improve upper bounds on the worst case running time for MaxSat. First, Cai and Chen showed that MaxSat can be solved in time $|F|2^{O(k)}$ when the clause size is bounded by a constant. Imposing no restrictions on clause size, Mahajan and Raman and, independently, Dantsin *et al.* improved this to $O(|F|\phi^k)$, where $\phi \approx 1.6181$ is the golden ratio. We present an algorithm running in time $O(|F|1.3995^k)$. The result extends to finding an optimal assignment and has several applications, in particular, for parameterized complexity and approximation algorithms. Moreover, if F has K clauses, we can find an optimal assignment in $O(|F|1.3972^K)$ steps and in $O(1.1279^{|F|})$ steps, respectively. These are the fastest algorithm in the number of clauses and the length of the formula, respectively.

1 Introduction

Despite being *NP*-hard, many problems have to be solved in practice. Besides approximation or heuristic algorithms, it is often important to have exact algorithms with proven performance bounds. Thus, there are big efforts to develop efficient exponential time algorithms as recent publications show [5, 2, 10, 12–14, 19, 22]. In particular, it becomes more and more clear that there are tight connections between improving the worst-case upper bound and and developing heuristics that are used in practical algorithms [10, 17].

There are two problems from logic playing a major role in computational complexity theory—Satisfiability (Sat) and Maximum Satisfiability (MaxSat). Research on nontrivial upper bounds for Sat was initiated by Monien and Speckenmeyer [15, 16] and subsequently has seen several improvements. We refer to Pudlák [23] for a recent survey, omitting any details here. Let us only emphasize

* 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.

that in this field even small or seemingly tiny improvements in the bases of the exponential terms can mean significant progress. For instance, for the Satisfiability of a propositional formula consisting of K clauses, Hirsch [12] improved the upper bound of order 1.25993^K of Monien and Speckenmeyer [15] to 1.23883^K (omitting polynomial terms). With respect to the length $|F|$ of the given formula, he improved Kullmann and Luckhardt's [13] upper bound from $1.08006^{|F|}$ to $1.07578^{|F|}$.

So far, with respect to developing good upper bounds researchers have paid a bit less attention to MaxSat. However, it is a problem of central importance in computer science: It plays a major role in the theory of computational complexity and approximation algorithms [1, 20, 21], it is an important problem in parameterized complexity theory [6, 9, 10], and it has numerous applications, see, e.g., [3, 4, 11]. We refer to Crescenzi and Kann [7] for a survey on approximation results for MaxSat. Here, let us only mention that there does not exist a polynomial time approximation algorithm with an approximation factor arbitrarily close to 1 unless $P = NP$ [1]. Dantsin *et al.* show how to move the approximation factor arbitrarily close to 1 using an exponential time algorithm [8]. As to parameterized complexity, the following is known: 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 [6] proved that for constant q parameterized Max$q$Sat is *fixed parameter tractable,* implying that every problem in the optimization class *MaxSNP* is also fixed parameter tractable. More precisely, Cai and Chen showed that it can be solved in time $O(|F|2^{O(k)})$. Without any restriction on clause size, Mahajan and Raman [14] improved the running time to $O(|F|1.6181^k)$. The same running time was achieved independently by Dantsin *et al.* [8] who use it as a basis for developing an exponential time approximation algorithm. Note that the time bound of $O(|F|1.6181^k)$ transfers easily into the time bound $O(|F| + k^2 1.6181^k)$ using a standard technique of parameterized complexity theory called reduction to problem kernel [14]. This technique also applies to our algorithms, giving analogous results. By more refined techniques the polynomial factor before the exponential term can be made even smaller.

Our main results are as follows. We present an $O(|F|1.3995^k)$ time algorithm for MaxSat, where $|F|$ is the length of the formula in conjunctive normal form. Surprisingly, if we want to determine the maximum number of satisfiable clauses, we get the time bound $O(|F|1.3972^K)$, where 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 to results of Mahajan and Raman [14] 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 ask for the existence of a cut of size at least k. We improve Mahajan and Raman's time bound of $O(m + n + k4^k)$ to $O(m + n + k^2 2.6196^k)$. Finally, we can improve the approximation algorithm of Dantsin *et al.* [8] by simply replacing their exponential time algorithm by our faster one. A technical report contains all proofs that are omitted here [18].

2 Basic definitions

We assume familiarity with basic notions of logic and use a similar notation as in [12]. 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 pairs. 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 several 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 occurs also outside of 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 consider 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."

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 $\top$-clauses is called *final*. Among equisatisfiable ones there is exactly one final formula.

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.

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 $\bar{x}$ occurs only in $\{\bar{x}, y, z\}$. As a rule, we will use x, y, z to denote variables and l to denote a literal. To simplify presentation, we usually assume w.l.o.g. that a variable occurs at least as often positively as it occurs negatively in the formula.

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 one other 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. In what follows, we provide a quite extensive list of transformation and splitting rules, which form the key to our improved algorithm (see Subsection 3.3).

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

Complementary 1-clause rule. $\dfrac{F}{\{\top\} \cup G}$ if $F = \{\{\bar{x}\}, \{x\}\} \cup G$.

Dominating 1-clause rule. $\dfrac{F}{F[l]}$ if $\bar{l}$ occurs i times in F, and l occurs at least i times in a 1-clause.

Resolution rule. $\dfrac{\{\{\bar{x}\} \cup K_1, \{x\} \cup K_2\} \cup G}{\{\top, K_1 \cup K_2\} \cup G}$ if G 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 $\{\{\bar{x}_1\}, \{\bar{x}_2\}, \dots, \{\bar{x}_r\}, \{x_1, x_2, \dots, x_r\}, \{x_1, x_2, \dots, x_r\}\}$ is an *r-star*. Let F be an r-star, then $\frac{F}{\{\top, \dots, \top\}}$, where the "$\top$-multiset" contains $r + 1$ many $\top$'s.

Definition 2 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

A splitting rule replaces a formula F by several formulas F_i such that F and at least one F_i are equisatisfiable. The *branching vector* of a rule $\frac{F}{F_1,\ldots,F_r}$ is $(b_1,\ldots,b_r)$ if $S(F)+b_i \leq S(Reduce(F_i))$ for all $1 \leq i \leq r$ and all F for which the rule is applicable. The branching vectors will be used to determine the time complexity of the algorithm using these rules. Determining the values of the branching vectors often involves some further case distinction. The full paper contains the proof that always one of the subsequent transformation rules is applicable to a reduced formula [18].

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 w.l.o.g. $(2,1)$, $(3,1)$, or $(2,2)$-literals. Moreover, there must be at least one $(2,1)$-literal x.

The branching vectors are: **T1:** $(3,2)$ or $(4,1)$, **T2:** $(3,3)$, **T3:** $(3,2)$, **T4:** $(4,2)$, **T5:** $(4,1)$, **T6:** $(2,3)$ or $(3,2)$ or $(4,1)$, **T7:** $(4,1)$ or $(3,2)$.

We prove here only representatively **T3** (see below): Two clauses contain x, hence $S(F)+2 \leq S(F[x])$. In $F[x]$, however, y is a $(1,1)$-literal and the resolution rule is applicable leading to $S(F)+3 \leq S(Reduce(F[x])$. In $F[\bar{x}]$, y is a $(1,1)$-literal, too. So, $S(F)+2 \leq S(Reduce(F[\bar{x}]))$. Together, we obtain the branching vector $(3,2)$.

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

T2 $\dfrac{F}{F[l],F[\bar{l}]}$ if $F=\{\{\bar{x},\bar{l},\ldots\},\{x,l,\ldots\},\{x,\ldots\},\{l,\ldots\},\ldots\}$.

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

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

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.

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 $\tilde{l}$ occurs with $\tilde{x}$ in exactly one clause.

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

All variables occur exactly four times. Here, we assume that F is reduced, contains no closed subformulas, and each variable occurs exactly four times. The following branching vectors can be proved: **D1:** $(4,1)$, **D2:** $(3,3)$, **D3:** $(4,1)$, **D4:** $(2,2)$, **D5:** $(3,4,5)$, **D6:** $(1,4,11)$.

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

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

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

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

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

D6 $\dfrac{F}{F[\bar{x}], F[x, \bar{y}], F[x, y, \bar{z}_1, \bar{z}_2, \dots, \bar{z}_6]}$
if $F = \{\{\bar{x}\}, \{x, y, \dots\}, \{x, \dots\}, \{x, \dots\}, \{y, z_1, z_2, z_3, \dots\}, \{y, z_4, z_5, z_6, \dots\}, \{\bar{y}\}, \dots\}$ and F is simple and each positive clause has size at least 4.

One interesting point in **D2** is that $F[\bar{x}, \bar{y}]$ is missing, which is possible since if there is an optimal assignment with $x = y = 0$, then there is also an optimal assignment with $x = 1$. **D5** and **D6** are similar.

There is a literal that occurs at least five times. Let F be reduced.

F1 $\dfrac{F}{F[x], F[\bar{x}]}$ if $\tilde{x}$ 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

The correctness of Algorithm B in Fig. 1 to compute the maximum number of satisfiable clauses follows from the correctness of all transformation and splitting rules and the fact that at least one rule is always applicable. A good data structure to represent formulas in conjunctive normal form is important. 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. It is easy to see that finding an applicable rule and applying it takes only polynomial time in the length of the formula. With more care, it can be done in linear time.

Algorithm B (Fig. 1) constructs an equisatisfiable formula $\{\top, \ldots, \top\}$ from a formula F by using transformation and splitting rules. It can be easily converted to answer our decision version of MaxSat and to compute also an optimal assignment.

Input: A formula F in clause form
Output: An equisatisfiable formula $B(F) = \{\top, \ldots, \top\}$
Method:
$F \leftarrow$ *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 $\dfrac{F}{F_1, \ldots, F_r} \in \{\textbf{T1–T7,D1–D6,F1}\}$;
return $\max\{B(F_1), \ldots, B(F_r)\}$
fi

Fig. 1. Algorithm B. Note that $F_1 \oplus F_2 \oplus \cdots \oplus F_m$ denotes the decomposition of F into closed subformulas and $\max\{B(F_1), \ldots, B(F_r)\}$ is the multiset with the maximum number of $\top$'s. **D4** is chosen only if no other rule is applicable. For small formulas the result is computed as $A(F)$ by some naive exponential time algorithm A.

We follow Kullmann and Luckhardt [13] (also cf. [12]) 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 $(S(\mathit{Reduce}(F_1)) - S(F), \ldots, S(\mathit{Reduce}(F_r)) - S(F))$ 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 $O(\alpha_{\max}^V)$ nodes, where V is the maximum value of a node in the tree [13]. Here, the number of clauses K is an upper bound on the value $S(F)$. Hence, the size of the branching tree is $O(\alpha_{\max}^K)$. Here $\alpha_{\max} = \sqrt{2}$ is the branching number of **D4**, but it cannot occur in all nodes, so the result is slightly better than $\sqrt{2}^K$. (See the full paper for details.)

Theorem 3 *For a formula F in conjunctive normal form, maxsat(F) can be computed in $O(|F| \cdot 1.3995^k)$ steps or $O(|F| \cdot 1.3972^K)$ steps, 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.*

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

Corollary 4 *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. Using *reduction to problem kernel*, Mahajan and Raman show that an algorithm that solves 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 [14].

Corollary 4 improves Theorem 7 of Mahajan and Raman [14] by decreasing the exponential term from $\phi^k \approx 1.6181^k$ to 1.3995^k. Analogously, the running time for Max$q$Sat is improved to $O(qk \cdot 1.3995^k + |F|)$. In addition, Mahajan and Raman also introduced a "more suitable parameterization" of MaxSat, asking whether at least $\lceil K/2 \rceil + k$ clauses are satisfiable. Recall that, K being the total number of clauses in F, *every* formula has an assignment that fulfills at least half of its clauses. Mahajan and Raman reduce this variant to the original problem we study here. They obtain a time complexity of $O(|F| + k^2\phi^{6k}) = O(|F| + k^2 17.9443^k)$ [14]. Plugging in our better bound, we automatically get $O(|F| + k^2 1.3995^{6k}) = O(|F| + k^2 7.5135^k)$.

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 previous 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.

Owing to the dominating 1-clause rule, we can often assume that a satisfied clause is not a 1-clause. In these cases, 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 1-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 that the branching vector with respect to length reduction of **F1** is at least $(7, 5)$.

We introduce a new splitting rule:

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

Here, $|F| - |Reduce(F[l_2])| \geq 8$ and $|F| - |F[\bar{l}_2]| \geq 4$, so the branching vector is $(8, 4)$.

We modify the algorithm such 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 1-clause. The branching vector for **D4** is then $(6,6)$. Altogether, the branching vectors are: **T1:** $(9,4)$ or $(8,5)$, **T2:** $(7,7)$, **T3:** $(7,6)$, **T4:** $(8,6)$, **T5:** $(10,3)$ or $(9,7)$, **T6:** $(8,4)$ or $(7,5)$, **T7:** $(10,3)$, **D1:** $(8,4)$, **D2:** $(7,8)$, **D3:** $(8,4)$, **D4:** $(6,6)$, **D4′**: $(8,5)$, **D5:** $(8,11,14)$, **D6:** $(4,16,32)$, **F1:** $(9,5)$ or $(7,5)$.

Theorem 5 *We can solve* MaxSat *in* $O(1.1279^{|F|})$ *steps.*

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

Theorem 5 should be compared to the best known result for the "simpler" Satisfiability problem obtained by Hirsch [12]. He proves the time bound $O(1.0757^{|F|})$. Moreover, Theorem 5 implies an upper bound of $O(1.2722^K)$ for the Max2Sat problem, using $|F| \leq 2K$, where K denotes the number of clauses in F. Mahajan and Raman also studied the Maximum Cut problem (MaxCut) for undirected graphs: Given a graph $G = (V, E)$ on n vertices and m edges and an integer k, is there a set $S \subset V$ such that at least k edges of G have one endpoint in V and one endpoint in $V - S$? They prove that MaxCut can be solved in time $O(m+n+k4^k)$ [14]. Using a reduction to Max2Sat (also see "Method 2" in [14]), MaxCut can be solved in $O(m+n+k^2 1.2722^{4k}) = O(m+n+k^2 2.6196^k)$ steps, thus basically decreasing the base of the exponential function from 4 to 2.6196.

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.

To improve the upper time bounds further is an interesting open problem. In particular, can Max2Sat, Max3Sat, or, generally, Max$q$Sat be solved faster than the general problem? 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.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proc. of 33d FOCS*, pages 14–23, 1992.
2. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.
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. 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.

5. R. Beigel and D. Eppstein. 3-Coloring in time $o(1.3446^n)$: A no MIS algorithm. In *Proc. of 36th FOCS*, pages 444–452, 1995.
6. L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *J. Comput. Syst. Sci.*, 54:465–474, 1997.
7. P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, Apr. 1997.
8. 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.
9. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.
10. R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 49, 1999.
11. P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.
12. E. A. Hirsch. Two new upper bounds for SAT. In *Proc. of 9th SODA*, pages 521–530, 1998.
13. O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. 1997. Submitted to *Information and Computation.*
14. M. Mahajan and V. Raman. Parametrizing above guaranteed values: MaxSat and MaxCut. Technical Report TR97-033, ECCC Trier, 1997. To appear in *Journal of Algorithms.*
15. B. Monien and E. Speckenmeyer. Upper bounds for covering problems. Technical Report Reihe Theoretische Informatik, Bericht Nr. 7/1980, Universität Gesamthochschule Paderborn, 1980.
16. B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.
17. 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.
18. R. Niedermeier and P. Rossmanith. New upper bounds for MaxSat. Technical Report KAM-DIMATIA Series 98-401, Faculty of Mathematics and Physics, Charles University, Prague, July 1998. Extended abstract to appear at *26th International Colloquium on Automata, Languages, and Programming (ICALP'99)*, Prague, July 1999.
19. 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.
20. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
21. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43:425–440, 1991.
22. R. Paturi, P. Pudlák, M. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proc. of 39th FOCS*, pages 628–637, 1998.
23. P. Pudlák. Satisfiability—algorithms and logic (invited paper). In *Proceedings of the 23d Conference on Mathematical Foundations of Computer Science*, number 1450 in Lecture Notes in Computer Science, pages 129–141, Brno, Czech Republic, Aug. 1998. Springer-Verlag.