Electronic Colloquium on Computational Complexity, Report No. 36 (1999)

A New Algorithm for MAX-2-SAT

Edward A. Hirsch*

August 29, 1999

Abstract

Recently there was a significant progress in proving (exponential-time) worst-case upper bounds for the propositional satisfiability problem (SAT). MAX-SAT is an important generalization of SAT. Several upper bounds were obtained for MAX-SAT and its NP-complete subproblems. In particular, Niedermeier and Rossmanith recently proved the worst-case upper bound $O\left(2^{K/2.88\ldots}\right)$ for MAX-2-SAT (i.e. each clause contains at most two variables), where K is the number of clauses. In this paper we improve this bound to $O\left(2^{K_2/4}\right)$, where K_2 is the number of 2-clauses. In addition, our algorithm and the proof are much simpler than those of Niedermeier and Rossmanith. The key ideas are to use the symmetric flow algorithm of Yannakakis and to count only 2-clauses (and not 1-clauses).

1 Introduction.

SAT (the problem of satisfiability of a propositional formula in conjunctive normal form (*CNF*)) can be easily solved in time of the order 2^N, where N is the number of variables in the input formula. In the early 1980s this trivial bound was improved for formulas in 3-CNF by Monien and Speckenmeyer [14] (see also [15]) and independently by Dantsin [2] (see also [5, 3]). After that, many upper bounds for SAT and its NP-complete subproblems were obtained ([10, 18] are the most recent). Most authors consider bounds w.r.t. three main parameters: the length L of the input formula (i.e. the number of literal occurrences), the number K of its clauses and the number N of the variables occurring in it. In this paper we consider bounds w.r.t. the parameters K and L. The best such bounds for SAT are $p(L)2^{K/3.23\ldots}$ [10] (p is a polynomial) and $O(2^{L/9.7\ldots})$ (see the journal version of [10]).

The maximum satisfiability problem (*MAX-SAT*) is an important generalization of SAT. In this problem we are given a formula in CNF, and the answer is the maximal number of simultaneously satisfiable clauses. This problem is NP-complete[1] even if each clause contains at most two literals (*MAX-2-SAT*; see, e.g., [17]). This problem was widely studied in

*Steklov Institute of Mathematics at St.Petersburg, 27 Fontanka, 191011 St.Petersburg, Russia. Email: hirsch@pdmi.ras.ru, URL: http://logic.pdmi.ras.ru/~hirsch/index.html

[1] A more precise NP-formulation is, of course, "given a formula in CNF, decide whether there is an assignment that satisfies at least k clauses".

ISSN 1433-8092

the context of approximation algorithms (see, e.g., [21, 8, 11, 9]). As to the worst-case time bounds for the exact solution of MAX-SAT, Niedermeier and Rossmanith [16] recently proved two worst-case upper bounds: $O(L \cdot 2^{K/2.15\ldots})$ for MAX-SAT, and $O(2^{K/2.88\ldots})$ for MAX-2-SAT. For the latter bound, they presented an algorithm for MAX-SAT running in $O(2^{L/5.76\ldots})$ time; the desired bound follows since $L \leq 2K$. They also posed a question whether this bound can be improved by a direct algorithm (and not by an algorithm for general MAX-SAT for a bound w.r.t. L). In this paper, we answer this question by giving an algorithm which solves MAX-2-SAT in $O(2^{K/4})$ time. In addition, our algorithm and the proof are much simpler.

Most of the algorithms/bounds mentioned above use the Davis-Putnam procedure [7, 6]. In short, this procedure allows to reduce the problem for a formula F to the problem for two formulas $F[v]$ and $F[\overline{v}]$ (where v is a propositional variable). This is called "splitting". Before the algorithm splits each of the obtained two formulas, it can transform them into simpler formulas F_1 and F_2 (using some *transformation rules*). The algorithm does not split a formula if it is trivial to solve the problem for it; these formulas are the leaves of the *splitting tree* which corresponds to the execution of such algorithm. For most known algorithms, the leaves are trivial formulas (i.e. the formulas containing no non-trivial clauses).

In the algorithm presented in this paper, the leaves are satisfiable formulas and formulas for which the (polynomial time) "symmetric flow" algorithm of Yannakakis [21] finds an optimal solution (this algorithm either finds an optimal solution or simplifies the input formula). Transformation rules include the pure literal rule, a slightly generalized resolution rule (using these two rules one can solve MAX-SAT in a polynomial time in the case that each variable occurs at most twice; it was already observed in, e.g., [19]), and the frequent 1-clause rule [16]. Although in MAX-SAT 1-clauses cannot be eliminated by the usual unit propagation technique, in the case of MAX-2-SAT they can be eliminated by the symmetric flow algorithm of Yannakakis [21]. Thus, before each splitting we can transform a formula into one which consists of 2-clauses, and each variable occurs at least three times. Therefore, each splitting eliminates at least three 2-clauses in each branch. This observation would already improve the bound of [16] to $O(2^{K_2/3})$. However, by careful choice of a variable for splitting, we get a better bound $O(2^{K_2/4})$, which implies the bound $O(2^{L/8})$ (since $L \geq 2K_2$).

In Sect. 2 we give basic definitions and formulate in our framework the known results we use. In Sect. 3 we present the algorithm and the proof of its worst-case upper bound.

2 Background.

Let V be a set of Boolean variables. The negation of a variable v is denoted by $\overline{v}$. Given a set U, we denote $\overline{U} = \{\overline{u} \mid u \in U\}$. *Literals* (usually denoted by $l, l', l_1, l_2, \ldots$) are the members of the set $W = V \cup \overline{V}$. *Positive literals* are the members of the set V. *Negative literals* are their negations. If w denotes a negative literal $\overline{v}$, then $\overline{w}$ denotes the variable v.

Algorithms for finding the exact solution of MAX-SAT are usually designed for the unweighted MAX-SAT problem. However, the formulas are usually represented by multisets (i.e., formulas in CNF with integer positive weights). In this paper we consider the weighted MAX-SAT problem with positive integer weights. A *(weighted) clause* is a pair (ω, S) where ω is a strictly positive integer number, and S is a nonempty finite set of literals which does not contain simultaneously any variable together with its negation. We call ω the *weight* of a clause (ω, S).

An *assignment* is a finite subset of W which does not contain any variable together with its negation. Informally speaking, if an assignment A contains a literal l, it means that l has the value $True$ in A. In addition to usual clauses, we allow a special *true clause* $(\omega, \mathbb{T})$ which is satisfied by every assignment. (We also call it a $\mathbb{T}$-*clause*.)

The length of a clause (ω, S) is the cardinality of S. A k-*clause* is a clause of the length exactly k. In this paper a *formula in (weighted) CNF* (or simply *formula*) is a finite set of (weighted) clauses (ω, S), at most one for each S. The *length of a formula* is the sum of the lengths of all its clauses. The total weight of all 2-clauses of a formula F is denoted by $\mathfrak{K}_2(F)$.

The pairs $(0, S)$ are *not* clauses, however, for simplicity we write $(0, S) \in F$ for all S and all F. Therefore, the operators $+$ and $-$ are defined:

$$\begin{aligned} F + G &= \{(\omega_1 + \omega_2, S) \mid (\omega_1, S) \in F \text{ and } (\omega_2, S) \in G, \text{ and } \omega_1 + \omega_2 > 0\}, \\ F - G &= \{(\omega_1 - \omega_2, S) \mid (\omega_1, S) \in F \text{ and } (\omega_2, S) \in G, \text{ and } \omega_1 - \omega_2 > 0\}. \end{aligned}$$

For a literal l and a formula F, we define

$$\begin{aligned} F[l] = \; & (\{(\omega, S) \mid (\omega, S) \in F \text{ and } l, \bar{l} \notin S\} + \\ & \{(\omega, S \setminus \{\bar{l}\}) \mid (\omega, S) \in F \text{ and } S \neq \{\bar{l}\}, \text{ and } \bar{l} \in S\} + \\ & \{(\omega, \mathbb{T}) \mid \omega \text{ is the sum of the weights } \omega' \text{ of all clauses } (\omega', S) \text{ of } F \text{ such that } l \in S\}. \end{aligned}$$

(Note that no $(\omega, \emptyset)$ or $(0, S)$ is included in $F[l]$, $F + G$ or $F - G$.) For an assignment $A = \{l_1, \ldots, l_s\}$ and a formula F, we define $F[A] = F[l_1][l_2] \ldots [l_s]$ (evidently, $F[l][l'] = F[l'][l]$ for every literals l, l' such that $l \neq \overline{l'}$). For short, we write $F[l_1, \ldots, l_s]$ instead of $F[\{l_1, \ldots, l_s\}]$. For example, if $F = \{\, (1, \{x, y\}), (5, \{\overline{y}\}), (2, \{\overline{x}, \overline{y}\}), (10, \{\overline{z}\})\}$, then $F[x, \overline{z}] = \{\, (11, \mathbb{T}), (7, \{\overline{y}\})\,\}$.

The optimal value $\mathrm{OptVal}(F) = \max_A \{\, \omega \mid (\omega, \mathbb{T}) \in F[A]\,\}$. An assignment A is *optimal* if $F[A]$ contains only one clause $(\omega, \mathbb{T})$ (or does not contain any clauses, in this case $\omega = 0$) and $\mathrm{OptVal}(F) = \omega$ $(= \mathrm{OptVal}(F[A])\,)$.

A formula is in 2-*CNF* if it contains only 2-clauses, 1-clauses and a $\mathbb{T}$-clause. A formula is in $2E$-*CNF* if it contains only 2-clauses and a $\mathbb{T}$-clause.

If we say that a (positive or negative) *literal* v *occurs* in a clause or in a formula, we mean that this clause (more formally, its second component) or this formula (more formally, one of its clauses) contains the literal v. However, if we say that a *variable* v *occurs* in a clause or in a formula, we mean that this clause or this formula contains the literal v, or it contains the literal $\overline{v}$. A variable v *occurs positively*, if the literal v occurs, and *occurs negatively*, if the literal $\overline{v}$ occurs. A literal l is an (i,j)-literal if l occurs exactly i times in the formula and the literal $\bar{l}$ occurs exactly j times in the formula. A literal is *pure* in a formula F if it occurs in F, and its negation does not occur in F. The following lemma is well-known and straightforward.

Lemma 1 *If l is a pure literal in F, then* $\mathrm{OptVal}(F) = \mathrm{OptVal}(F[l])$.

In this paper, the *resolvent* $\mathfrak{R}(C,D)$ of clauses $C=(\omega_1,\{l_1,l_2\})$ and $D=(\omega_2,\{\overline{l_1},l_3\})$ is the formula

$$\{\ (\max(\omega_1,\omega_2),\ \mathbb{T}),\ \ (\min(\omega_1,\omega_2),\ \{l_2,l_3\})\ \}$$

if $l_2 \neq \overline{l_3}$, and the formula $\{(\omega_1+\omega_2,\mathbb{T})\}$ otherwise. This definition is not traditional, but it is very useful in MAX-SAT context.

The following lemma is a straightforward generalization of the resolution correctness (see, e.g., [20]) for the case when there are weights, but the literal on which we are resolving does not occur in other clauses of the formula.

Lemma 2 *If F contains clauses $C=(\omega_1,\{v,l_1\})$ and $D=(\omega_2,\{\overline{v},l_2\})$ such that the variable v does not occur in other clauses of F, then*

$$\mathrm{OptVal}(F)=\mathrm{OptVal}(\ (F-\{C,D\})+\mathfrak{R}(C,D)\ ).$$

The following simple observation is also well-known (see, e.g., [13, 4]).

Lemma 3 *Let F be a formula in weighted CNF, and v be a variable. Then*

$$\mathrm{OptVal}(F)=\max(\,\mathrm{OptVal}(F[v]),\ \mathrm{OptVal}(F[\overline{v}])\,).$$

We also note that a polynomial time algorithm for 2-SAT is known. In our context, a formula F is satisfiable if $\mathrm{OptVal}(F)$ is equal to the sum of the weights of all clauses occurring in F.

Lemma 4 (see, e.g. [1]) *There is a polynomial time algorithm for 2-SAT.*

Yannakakis presented in [21] an algorithm which transforms a formula in 2-CNF into a formula in $2E$-CNF which has the same optimal value. This algorithm consists of two stages. The first stage is a removal of a maximum symmetric flow from a graph corresponding to the formula; this stage can be considered as a combination of three transformation rules (it is not important for us now which combination):

1) replacing[2] of a "cycle"

$$\{\ (\omega,\{l_1,\overline{l_2}\}),\ (\omega,\{l_2,\overline{l_3}\}),\ \ldots,\ (\omega,\{l_k,\overline{l_1}\})\ \}$$

by another cycle

$$\{\ (\omega,\{\overline{l_1},l_2\}),\ (\omega,\{\overline{l_2},l_3\}),\ \ldots,\ (\omega,\{\overline{l_k},l_1\})\ \};$$

2) replacing of a set

$$\{\ (\omega,\{\overline{l_1}\}),\ (\omega,\{l_1,\overline{l_2}\}),\ (\omega,\{l_2,\overline{l_3}\}),\ \ldots,\ (\omega,\{l_{k-1},\overline{l_k}\})\ \}$$

by the set

$$\{\ (\omega,\{\overline{l_1},l_2\}),\ (\omega,\{\overline{l_2},l_3\}),\ \ldots,\ (\omega,\{\overline{l_{k-1}},l_k\}),\ (\omega,\{\overline{l_k}\})\ \};$$

[2] This replacing is made by subtracting the weights: e.g., if a formula contains a clause $(\omega',\{l_1,\overline{l_2}\})$ with $\omega'\geq\omega$, then it is split into two clauses $(\omega'-\omega,\{l_1,\overline{l_2}\})$ and $(\omega,\{l_1,\overline{l_2}\})$, and the latter clause is replaced as formulated.

3) replacing two contradictory clauses $(\omega, \{l\})$ and $(\omega, \{\bar{l}\})$ by a true clause of the weight ω.

The second stage is replacing of the obtained formula F' by the formula $F'[A]$ for some assignment A (it is not important for us now which assignment). Evidently, this algorithm does not increase the total weight of all 2-clauses.

Lemma 5 ([21]) *There is a polynomial time algorithm which given an input formula F in weighted 2-CNF, outputs a formula G in weighted 2E-CNF, such that $\mathfrak{K}_2(G) \le \mathfrak{K}_2(F)$, and* $\mathrm{OptVal}(F) = \mathrm{OptVal}(G)$.

The following fact was observed by Niedermeier and Rossmanith.

Lemma 6 ([16]) *If the weight of a 1-clause $(\omega, \{l\})$ of a formula F is not less than the total weight of all clauses of F containing the literal $\bar{l}$, then* $\mathrm{OptVal}(F) = \mathrm{OptVal}(F[l])$.

3 Results.

In this section we present Algorithm 1 which solves MAX-2-SAT in the time $O(2^{K_2/4})$, where K_2 is the total weight of 2-clauses in the input formula (in the case of unweighted MAX-2-SAT, K_2 is the number of 2-clauses).

Algorithm 1.

Input: *A formula F in weighted 2-CNF.*
Output: $\mathrm{OptVal}(F)$.

Method.

(1) Apply the symmetric flow algorithm from [21] (see Lemma 5) to F.

(2) If there is a pure literal l in F, assume $F := F[l]$.

(3) If there is a variable that occurs in F exactly once positively in a clause C and exactly once negatively in a clause D, then $F := (F - \{C, D\}) + \mathfrak{R}(C, D)$.

(4) If F has been changed at steps (2)–(3), then go to step (1).

(5) If F is satisfiable[3], return the sum of the weights of all its clauses.

(6) If there is a variable v occurring in the clauses of F of the total weight at least 4, execute Algorithm 1 for the formulas $F[v]$ and $F[\overline{v}]$, and return the maximum of its answers.

(7) Find[4] in F a clause $(\omega, \{l_1, l_2\})$ such that l_1 and l_2 are (2,1)-literals, and the two other clauses C and D containing the literals $l_1, \overline{l_1}$ do not contain the literals $l_2, \overline{l_2}$. Execute Algorithm 1 for the formulas $(F[l_1] - \{C, D\}) + \mathfrak{R}(C, D)$, and $F[\overline{l_1}, l_2]$, and return the maximum of its answers.

□

[3] We can check it in a polynomial time [1], see Lemma 4.
[4] Theorem 1 proves that it is possible to find a clause satisfying the conditions of this step.

Theorem 1 *Given a formula F in 2-CNF, Algorithm 1 always correctly finds* $\mathrm{OptVal}(F)$ *in time $O(2^{\mathfrak{K}_2(F)/4})$.*

Proof. Correctness. If Algorithm 1 outputs an answer, then its correctness follows from the lemmata of Sect. 2 (step (1): Lemma 5; step (2): Lemma 1; step (3): Lemma 2; step (6): Lemma 3; step (7): Lemmata 3, 2 and 6, note that at this step F consists of the clauses of weight 1).

Since any change at steps (2) and (3) decreases the total weight of 2-clauses in F, and the step (1) does not increase it, the cycle (1)–(4) is repeated a polynomial number of times. Now it remains to show that at step (7) Algorithm 1 always can find a clause satisfying its conditions.

Note that at step (7) the formula F is not satisfiable, consists only of 2-clauses (and, maybe, a $\mathbb{T}$-clause), does not contain pure literals, and each variable occurs in it exactly three times, i.e. F contains only (2,1)-literals and (1,2)-literals. Since F is not satisfiable, there exists at least one clause in it that contains two (1,2)-literals (otherwise the assignment consisting of (2,1)-literals is satisfying; cf. "Extended Sign Principle" of [12]). Thus, (1,2)-literals occur in at most $N-1$ clauses of F, where N is the number of variables occurring in F. There are $3N/2$ 2-clauses in F. Hence, F contains more than $N/2$ 2-clauses consisting only of (2,1)-literals. There are at least $N+1$ literals in these clauses, thus, there is at least one (2,1)-literal occurring in *two* such clauses. This literal, and at least one of the two literals occurring with it in these clauses, satisfy the condition of the step (7).

Running time. Each of the steps of Algorithm 1 (not including recursive calls) takes only a polynomial time (Lemmata 5 and 4). The steps (1)–(5) do not increase the total weight of 2-clauses in F. By the above argument, each of these steps is executed a polynomial number of times during one execution of Algorithm 1 (again not including recursive calls). It suffices to show that for each formula F' which is an argument of a recursive call, $\mathfrak{K}_2(F') \leq \mathfrak{K}_2(F) - 4$.

Note that at the moment of a recursive call, the formula consists only of 2-clauses (and, maybe, a $\mathbb{T}$-clause). Then the statement follows from the conditions of the steps (6) and (7). □

Corollary 1 *Given a formula F in unweighted*[5] *2-CNF of length L, Algorithm 1 always correctly finds* $\mathrm{OptVal}(F)$ *in time $O(2^{L/8})$.*

Remark 1 *Of course, in Corollary 1 only the number of literal occurrences in 2-clauses is essential.*

4 Conclusion

In this paper we improved the existing upper bound for MAX-2-SAT with integer weights to $O(2^{K_2/4})$, where K_2 is the total weight of 2-clauses of the input formula (or the number of 2-clauses for unweighted MAX-2-SAT). This also implies the $O(2^{L/8})$ bound for unweighted MAX-2-SAT, where L is the number of literal occurrences (in 2-clauses).

[5] I.e., all weights equal 1.

One of the key ideas of our algorithm is to count only 2-clauses (since MAX-1-SAT instances are trivial). It would be interesting to apply this idea to SAT, for example, by counting only 3-clauses in 3-SAT (since 2-SAT instances are easy). Also, it remains a challenge to find a "less-than-2^N" algorithm for MAX-SAT or even MAX-2-SAT, where N is the number of variables.

References

[1] S. A. Cook, *The complexity of theorem-proving procedure,* Proc. 3rd Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–159.

[2] E. Dantsin, *Tautology proof systems based on the splitting method*, Leningrad Division of Steklov Institute of Mathematics (LOMI), PhD Dissertation, Leningrad, 1982 (in Russian).

[3] E. Dantsin and L. O. Fuentes and V. Kreinovich, *Less than 2^n satisfiability algorithm extended to the case when almost all clauses are short*, Computer Science Department, University of Texas at El Paso, UTEP-CS-**91-5**, 1991.

[4] E. Dantsin, M. Gavrilovich, E. A. Hirsch, B. Konev, *Approximation algorithms for Max Sat: a better performance ratio at the cost of a longer running time*, PDMI preprint 14/1998, available from `ftp://ftp.pdmi.ras.ru/pub/publicat/preprint/1998/14-98.ps`

[5] E. Ya. Dantsin, V. Ya. Kreinovich, *Exponential upper bounds for the satisfiability problem*, Proc. of the IX USSR conf. on math. logic, Leningrad, 1988 (in Russian).

[6] M. Davis, G. Logemann, D. Loveland, *A machine program for theorem-proving*, Comm. ACM, vol. **5**, 1962, pp. 394–397.

[7] M. Davis, H. Putnam, *A computing procedure for quantification theory*, J. ACM, vol. **7**, 1960, pp. 201–215.

[8] U. Feige, M. X. Goemans, *Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT*, Proc. of the Third Israel Symposium on Theory of Computing and Systems, 1995, pp. 182–189.

[9] J. Håstad, *Some optimal inapproximability results*, Proc. of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 1–10.

[10] E. A. Hirsch, *Two new upper bounds for SAT*, Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 521–530. An improved version is accepted for publication in the special issue SAT-2000 of Journal of Automated Reasoning.

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

[12] O. Kullmann, H. Luckhardt, *Deciding propositional tautologies: Algorithms and their complexity*, Preprint, 1997, 82 pages; The ps-file can be obtained at `http://mi.informatik.uni-frankfurt.de/people/kullmann/kullmann.html`. A journal version, *Algorithms for SAT/TAUT decision based on various measures*, is submitted to in Information and Computation, 1998.

[13] M. Mahajan and V. Raman, *Parametrizing above guaranteed values: MaxSat and Max-Cut*, Technical Report TR97-033, Electronic Colloquium on Computational Complexity, 1997.

[14] B. Monien, E. Speckenmeyer, *3-satisfiability is testable in* $O(1.62^r)$ *steps,* Bericht Nr. 3/1979, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn.

[15] B. Monien, E. Speckenmeyer, *Solving satisfiability in less then* 2^n *steps,* Discrete Applied Mathematics, vol. **10**, 1985, pp. 287–295.

[16] R. Niedermeier and P. Rossmanith. *New upper bounds for MaxSat*, Technical Report KAM-DIMATIA Series 98-401, Charles University, Praha, Faculty of Mathematics and Physics, July 1998. Extended abstract appeared in Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, LNCS **1644**, pp. 575–584, 1999.

[17] Ch. H. Papadimitriou, *Computational Complexity*, Addison–Wesley, 1994, 532 p.

[18] R. Paturi, P. Pudlak, M. E. Saks, F. Zane, *An Improved Exponential-time Algorithm for k-SAT*, Proceedings of the 39th Annual Symposium on Foundations of Computer Science, 1998, pp. 628–637.

[19] V. Raman, B. Ravikumar and S. Srinivasa Rao, *A Simplified NP-complete MAXSAT problem*, Technical Report IMSc-TR-97/06/23 of The Institute of Mathematical Sciences, Chennai, India. (This citation is from [13].)

[20] J. A. Robinson, *Generalized resolution principle*, Machine Intelligence, vol. **3**, 1968, pp. 77–94.

[21] M. Yannakakis, *On the Approximation of Maximum Satisfiability*, Journal of Algorithms **17**, 1994, pp. 475–502.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'