Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 36 (1999)

A $2^{K/4}$-time Algorithm for MAX-2-SAT: Corrected Version

Edward A. Hirsch*

Abstract

Recently there was an explosion in proving (exponential-time) worst-case upper bounds for the propositional satisfiability problem (SAT) and related problems, mainly, k-SAT, MAX-SAT and MAX-2-SAT. The previous version of this paper contained an algorithm for MAX-2-SAT, and a "proof" of the theorem stating that this algorithm runs in time of the order $2^{K_2/4}$, where K_2 is the number of 2-clauses in the input formula. This bound and the corresponding bound $2^{L/8}$ (where L is the length of the input formula) are still the best known. However, Jens Gramm pointed out to the author that the algorithm in the previous revision of this paper had an error. In this revision of the paper, we present a corrected version of the algorithm and the proof of the same upper bound. The proof is still based on the key idea to count only 2-clauses (and not 1-clauses). However, the use of Yannakakis' symmetric flow algorithm is replaced by several transformation rules.

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 [17, 18] and independently by Dantsin [4]. After that, many upper bounds for SAT and its NP-complete subproblems were obtained ([15, 11, 21, 14, 24] 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 $\mathrm{poly}(L) \cdot 2^{K/3.23...}$ [11] and $\mathrm{poly}(L) \cdot 2^{L/9.7...}$ (see the journal version of [11]).

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 maximum number of simultaneously satisfiable clauses. This problem is NP-complete[1] even if each clause

*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. Supported in part by grants from INTAS and RFBR.

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

ISSN 1433-8092

contains at most two literals (*MAX-2-SAT*; see, e.g., [20]). MAX-SAT was widely studied in the context of approximation algorithms (see, e.g., [25, 8, 13, 10, 1]). Recently there was an explosion in the domain of the worst-case time bounds for the exact solution of MAX-SAT and MAX-2-SAT [16, 5, 19, 2, 9]. The best bounds from the listed papers are $O(L \cdot 2^{K/2.36...})$ and $O(L \cdot 2^{L/6.89...})$ for MAX-SAT [2], and the bounds $O(L \cdot 2^{K/3.74...})$ and $O(L \cdot 2^{L/7.48...})$ for MAX-2-SAT [9].

Most of the algorithms/bounds mentioned above, as well as the algorithm presented in this paper, use a kind of 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*). In a *splitting tree* corresponding to the execution of such an algorithm, the node labelled by F has two sons labelled by F_1 and F_2. 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. The running time of the algorithm is within a $\mathrm{poly}(L)$ factor of the number of leaves of the splitting tree.

In the previous revision of this paper [12], an algorithm achieving the bounds $\mathrm{poly}(L) \cdot 2^{K/4}$ and $\mathrm{poly}(L) \cdot 2^{L/8}$ was given. However, Jens Gramm [private communication] has pointed out that there is an error in this algorithm: it uses Yannakakis' symmetric flow algorithm [25] as a subroutine, Yannakakis' algorithm may introduce non-integer weights, and the algorithm from [12] relies on integer weights.

In this paper, we give a corrected version of this algorithm. One of the main ideas remains the same: to count only 2-clauses (clearly, MAX-1-SAT is trivial). However, the present version of our algorithm does not use Yannakakis' symmetric flow algorithm. Instead, it uses a transformation rule that allows to eliminate a variable occurring only in two 2-clauses (with the same sign) and in one 1-clause (with the opposite sign)[2]. This rule, together with pure literals elimination, resolution rule, the annihilation of 1-clauses and dominating 1-clause rule (this one is from [19]), guarantees that we can transform a formula into one in which every variable occurs in at least three (unweighted) 2-clauses. A splitting therefore eliminates at least three 2-clauses in each branch. This observation already gives a very simple $\mathrm{poly}(L) \cdot 2^{K_2/3}$-time algorithm. In this paper we show that by choosing a variable v occurring in the maximum number of 2-clauses we can get a better bound $\mathrm{poly}(L) \cdot 2^{K_2/4}$ which implies the bound $\mathrm{poly}(L) \cdot 2^{L/8}$ (since $L \geq 2K_2$).

In Section 2 we give basic definitions. In Section 3 we describe the transformation rules and show their correctness. In Section 4 we present the algorithm and the proof of its worst-case time upper bound.

2 Background

Let V be a set of Boolean variables. The negation of a variable v is denoted by $\overline{v}$. *Literals* are variables and their negations. If l denotes a negative literal $\overline{v}$, then $\overline{l}$ denotes the variable

[2] In fact, this rule inspired the author to replace (unfortunately, incorrectly) several transformation rules by a "similar" Yannakakis' algorithm in a very early draft of this paper.

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 positive integer 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 set of literals which does not contain any variable together with its negation. Informally speaking, if an assignment A contains a literal l, then the literal 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. A formula is in *2-CNF* if it contains only 2-clauses, 1-clauses and a $\mathbb{T}$-clause. 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 $\aleph_2(F)$.

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:

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

Example 1. If

$$F = \{\ (2, \mathbb{T}),\ (3, \{x, y\}),\ (4, \{\overline{x}, \overline{y}\})\ \}$$

and

$$G = \{\ (2, \{x, y\}),\ (4, \{\overline{x}, \overline{y}\})\ \},$$

then

$$F - G = \{\ (2, \mathbb{T}),\ (1, \{x, y\})\ \}.$$

□

For a literal l and a formula F, the formula $F[l]$ is obtained by setting the value of l to $True$. More precisely, we define

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

(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 $F[\{l_1, \ldots, l_s\}]$.

Example 2. If

$$F = \{\ (1, \mathbb{T}),\ (1, \{x, y\}),\ (5, \{\overline{y}\}),\ (2, \{\overline{x}, \overline{y}\}),\ (10, \{\overline{z}\}),\ (2, \{\overline{x}, z\})\ \},$$

then

$$F[x, \overline{z}] = \{\ (12, \mathbb{T}),\ (7, \{\overline{y}\})\ \}.$$

□

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 $\mathrm{OptVal}(F) = \omega$ $(= \mathrm{OptVal}(F[A])\,)$.

If we say that a *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 $\overline{v}$.

For a literal l, we write $\#_l(G)$ to denote the total weight of the clauses of a formula G in which l occurs. We omit G when the meaning of F is clear from the context. We also write $\#_l^{(k)}$ to denote the total weight of k-clauses in which l occurs.

A *closed subformula* G is a subset of a formula F such that none of the variables occurring in G occur in $F - G$.

3 Transformation rules

In this section we formulate the transformation rules we use. We also show their correctness. We say that a rule is *correct* if it preserves OptVal.

Pure literal. 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 a is a pure literal in F, then $\mathrm{OptVal}(F) = \mathrm{OptVal}(F[a])$.

Rule $\mathbf{T_{pure}}$ replaces F by $F[a]$ if a is a pure literal.

Annihilation of 1-clauses. Rule $\mathbf{T_{ann}}$ replaces F by $(F - \{\,(1, \{a\})\,,\ (1, \{\overline{a}\})\,\}) + (1, \mathbb{T})$ if F contains clauses $(\omega_1, \{a\})$ and $(\omega_2, \{\overline{a}\})$. In other words, this rule "annihilates" opposite 1-clauses.

Resolution. In this paper, the *resolvent* $\mathfrak{R}(C, D)$ of 2-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 a statement about usual resolution (see, e.g., [23]).

Lemma 2. If F contains 2-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

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

Rule $\mathbf{T_{DP}}$ replaces F by $(F - \{C, D\}) + \mathfrak{R}(C, D)$ if F, C and D satisfy the condition of Lemma 2.

Remark 1. Note that if at least one of C and D is a 1-clause, the situation is handled by $\mathbf{T_{ann}}$ or $\mathbf{T_{dom}}$ (described below) followed by $\mathbf{T_{pure}}$.

Dominating 1-clause. The following fact was observed by Niedermeier and Rossmanith.

Lemma 3 (Niedermeier, Rossmanith, [19]). If for a literal a and a formula F, $\#_a^{(1)} \geq \#_{\overline{a}}$, then $\mathrm{OptVal}(F) = \mathrm{OptVal}(F[a])$.

Rule $\mathbf{T_{dom}}$ replaces F by $F[a]$ in such a case.

Small closed subformula. We can easily compute the optimal value for a closed subformula G containing at most two variables. Clearly, $\mathrm{OptVal}(F) = \mathrm{OptVal}(F-G)+\mathrm{OptVal}(G)$. Rule $\mathbf{T_{small}}$ replaces F by $(F - G) + (\mathrm{OptVal}(G), \mathbb{T})$ in such a case.

Rare literal. Let F be a formula, and let a be a literal, such that $\#_a^{(2)} = 2$, $\#_{\overline{a}}^{(2)} = \#_a^{(1)} = 0$ and $\#_{\overline{a}}^{(1)} = 1$. Consider a 2-clause $(\omega, \{a, b\})$ in F. Rule $\mathbf{T_{rare}}$ replaces this clause by $(\omega, \mathbb{T})$ and replaces the literals $a, \overline{a}$ by the literals $\overline{b}, b$ respectively in all other clauses.

Lemma 4. Rule $\mathbf{T_{rare}}$ is correct.

Proof. Let F' be the obtained formula. It is trivial that $\mathrm{OptVal}(F') \leq \mathrm{OptVal}(F)$. We now prove the opposite inequality.

Let A be an optimal assignment for F. Let $b \in A$. Consider $F[b]$. Note that we can apply Rule $\mathbf{T_{dom}}$ to the literal $\overline{a}$ in this formula, i.e.,

$$\begin{aligned}&\mathrm{OptVal}(F) = \mathrm{OptVal}(F[A]) \leq \mathrm{OptVal}(F[b])\\ &= \mathrm{OptVal}(F[\overline{a}, b]) = \mathrm{OptVal}(F'[\overline{a}, b]) \leq \mathrm{OptVal}(F').\end{aligned}$$

Let now $\overline{b} \in A$. Consider $F[\overline{b}]$. Note that we can apply Rule $\mathbf{T_{ann}}$ and then Rule $\mathbf{T_{pure}}$ to the literal a in this formula, i.e.,

$$\begin{aligned}&\mathrm{OptVal}(F) = \mathrm{OptVal}(F[A]) \leq \mathrm{OptVal}(F[\overline{b}])\\ &= \mathrm{OptVal}(F[a, \overline{b}]) = \mathrm{OptVal}(F'[a, \overline{b}]) \leq \mathrm{OptVal}(F').\end{aligned}$$

□

4 Results

In this section we present Algorithm 1 which solves MAX-2-SAT in time of the order $2^{K_2/4}$, where K_2 is the total weight of 2-clauses of 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: OptVal(F).

Method.

(A1) Apply Rules $\mathbf{T_{pure}}$, $\mathbf{T_{ann}}$, $\mathbf{T_{DP}}$, $\mathbf{T_{dom}}$, $\mathbf{T_{small}}$, $\mathbf{T_{rare}}$ to F as long as at least one of them is applicable.

(A2) If F contains only a $\mathbb{T}$-clause, return the weight of this clause.

(A3) Choose an arbitrary variable v occurring in F with maximum $\#_v^{(2)} + \#_{\overline{v}}^{(2)}$, and form the formulas $F'_1 := F[v]$ and $F'_2 := F[\overline{v}]$. For each $i = 1, 2$, apply Rules $\mathbf{T_{pure}}$, $\mathbf{T_{ann}}$, $\mathbf{T_{DP}}$, $\mathbf{T_{dom}}$, $\mathbf{T_{rare}}$ to F'_i in such order that for the obtained formula F_i,

$$\mathfrak{K}_2(F) - \mathfrak{K}_2(F_i) \geq 4$$

holds[3]. Execute Algorithm 1 for the formulas F_1 and F_2 and return the maximum of its answers.

□

Theorem 1. Given a formula F in 2-CNF, Algorithm 1 always correctly finds OptVal(F) in time $\mathrm{poly}(L) \cdot 2^{K_2/4}$, where L is the length of F, and $K_2 = \mathfrak{K}_2(F)$.

Proof. *Running time.* Every transformation rule takes polynomial time and does not increase the total weight of non-$\mathbb{T}$-clauses. When the condition of a rule is satisfied, the rule decreases the total weight of non-$\mathbb{T}$-clauses. Thus, the transformation rules are executed a polynomial number of times during step (A1).

None of the steps of Algorithm 1 increases the total weight of 2-clauses of F. When the algorithm splits F at step (A3), the condition of this step guarantee that the weight of 2-clauses is decreased by at least four in each of the two branches. Algorithm 1 does not split formulas that do not contain 2-clauses, hence, the bound on the running time follows.

Correctness. The correctness of transformation rules is shown in Section 3. It remains to show that at step (A3) appropriate rules can be easily found. Clearly, if $\#_v^{(2)} + \#_{\overline{v}}^{(2)} \geq 4$, then no rules are needed. We now suppose the contrary.

Note that if for a variable u, $\#_u^{(2)} + \#_{\overline{u}}^{(2)} \leq 2$ holds, then u satisfies the condition of at least one of our transformation rules (and this is impossible). Moreover, $\#_u^{(2)} + \#_{\overline{u}}^{(2)} \leq \#_v^{(2)} + \#_{\overline{v}}^{(2)}$ by the choice of v. Thus, for every variable u occurring in F, we have $\#_u^{(2)} + \#_{\overline{u}}^{(2)} = 3$.

We now note that for each $i = 1, 2$, we have $\mathfrak{K}_2(F) - \mathfrak{K}_2(F'_i) \geq 3$.

Let $(\omega, \{a, b\})$ be one of the 2-clauses of F containing the variable v, where $a = v$ or $a = \overline{v}$, and let u be the variable corresponding to the literal b. Note that every F'_i contains a 2-clause containing u since F does not satisfy the condition of Rule $\mathbf{T_{small}}$. On the other hand, the variable u occurs in at most two 2-clauses of F'_i. Clearly, applying the transformation rules to this variable we can reduce the number of 2-clauses in F'_i yet by one obtaining F_i such that $\mathfrak{K}_2(F) - \mathfrak{K}_2(F_i) \geq 4$ (first apply $\mathbf{T_{ann}}$ to 1-clauses containing u (if possible), then apply $\mathbf{T_{pur}}$, $\mathbf{T_{DP}}$, $\mathbf{T_{dom}}$ or $\mathbf{T_{rare}}$ to u depending on the remaining clauses containing u). □

[3]Theorem 1 proves that it can be easily decided which rules should be applied and in which sequence.

Corollary 1. Given a formula F in unweighted[4] 2-CNF of length L, Algorithm 1 always correctly finds OptVal(F) in time poly(L) $\cdot 2^{L/8}$.

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

Remark 3. Clearly, the polynomial factor poly(L) in Theorem 1 and Corollary 1 is nearly linear.

Remark 4. Algorithm 1 can be easily redesigned so that it finds the optimal assignment (or one of them, if there are several assignments satisfying the same number of clauses) instead of OptVal(F).

5 Conclusion

In this paper we proved the upper bound poly(L)$\cdot 2^{K_2/4}$ for MAX-2-SAT with positive integer weights, 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), L is the number of literal occurrences. This also implies the bound poly(L) $\cdot 2^{L/8}$ for unweighted MAX-2-SAT.

The key idea of our algorithm is to count only 2-clauses (we can do it 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 for MAX-2-SAT, where N is the number of variables.

Acknowledgement

The author is very grateful to Jens Gramm for pointing out an error in [12], and for simplifying the algorithm.

References

[1] T. Asano, D. P. Williamson, *Improved Approximation Algorithms for MAX SAT*, Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000. To appear.

[2] N. Bansal, V. Raman, *Upper Bounds for MaxSat: Further Improved,* Proceedings of ISAAC'99.

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

[4] E. Dantsin, *Two propositional proof systems based on the splitting method (in Russian)*, Zapiski Nauchnykh Seminarov LOMI 105, 1981, pp.24-44. English translation: Journal of Soviet Mathematics 22(3):1293–1305, 1983.

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

[5] 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`

[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. Gramm, R. Niedermeier, *Faster exact solutions for MAX-2-SAT*, CIAC'2000. To appear.

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

[11] 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 to appear in Journal of Automated Reasoning special issue "SAT-2000".

[12] E. A. Hirsch, *A New Algorithm for MAX-2-SAT*, Revision 1 of Technical Report TR99-036, Electronic Colloquium on Computational Complexity, 1999. To appear in Proceedings of STACS'2000.

[13] 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.

[14] O. Kullmann, *New methods for 3-SAT decision and worst-case analysis*, Theoretical Computer Science 223(1-2), 1999, pp. 1–72.

[15] O. Kullmann, H. Luckhardt, *Deciding propositional tautologies: Algorithms and their complexity*, Preprint, 1997, 82 p., available from `http://www.cs.utoronto.ca/~kullmann`. A journal version, *Algorithms for SAT/TAUT decision based on various measures*, is to appear in Information and Computation.

[16] M. Mahajan and V. Raman, *Parametrizing above guaranteed values: MaxSat and MaxCut*, Technical Report TR97-033, Electronic Colloquium on Computational Complexity, 1997. To appear in Journal of Algorithms.

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

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

[19] 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. To appear in Journal of Algorithms.

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

[21] R. Paturi, P. Pudlák, 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.

[22] V. Raman, B. Ravikumar and S. Srinivasa Rao, *A simplified NP-complete MAXSAT problem*, Information Processing Letters (65)1, 1998, pp. 1–6.

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

[24] U. Schöning, *A probabilistic algorithm for k-SAT and constraint satisfaction problems,* Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999.

[25] 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'