Electronic Colloquium on Computational Complexity, Report No. 102 (2005)

Clause Shortening Combined with Pruning Yields a New Upper Bound for Deterministic SAT Algorithms

Evgeny Dantsin* Edward A. Hirsch† Alexander Wolpert*

Abstract

We give a deterministic algorithm for testing satisfiability of formulas in conjunctive normal form with no restriction on clause length. Its upper bound on the worst-case running time matches the best known upper bound for randomized satisfiability-testing algorithms [5]. In comparison with the randomized algorithm in [5], our deterministic algorithm is simpler and more intuitive.

1 Introduction

The problem of satisfiability of a propositional formula in conjunctive normal form (SAT) can be easily solved in 2^n polynomial-time steps, where n is the number of variables in the input formula. Since the early 1980s, this upper bound has been successively improved for k-SAT (the restricted case of SAT where clauses have at most k variables). The best bound to date for deterministic k-SAT algorithms is $(2-2/(k+1))^n$ up to a polynomial factor [2]. For randomized k-SAT algorithms, the currently best known bound is due to [8]; a close bound is given in [11]. These general bounds are improved for $k=3$ in [1, 7].

The list of successive improvements for SAT (with no restriction on clause length) is shorter:

deterministic algorithms		randomized algorithms	
$2^{n\left(1-\frac{2}{\sqrt{n\log n}}\right)}$	[3]	$2^{n\left(1-\frac{1}{2\sqrt{n}}\right)}$	[10]
$2^{n\left(1-\frac{1}{\log(2m)}\right)}$	[4]	$2^{n\left(1-\frac{1}{\log(2m)}\right)}$	[12]
		$2^{n\left(1-\frac{1}{\ln(m/n)+O(\ln\ln m)}\right)}$	[5]

Here n and m are respectively the number of variables and the number of clauses. For simplicity, we give the bounds above omitting polynomial factors; such a factor is typically linear in the length of the input formula (yet there are several exceptions).

*Department of Computer Science, Roosevelt University, 430 S. Michigan Ave., Chicago, IL 60605, USA. Email: {edantsin,awolpert}@roosevelt.edu

†Steklov Institute of Mathematics, 27 Fontanka, St. Petersburg 191023, Russia. Web: http://logic.pdmi.ras.ru/~hirsch/. Supported in part by Russian Science Support Foundation, Russian Foundation for Basic Research, and INTAS grant 04-77-7173.

ISSN 1433-8092

In this paper we give a deterministic algorithm for SAT with no restriction on clause length. Its upper bound on the worst-case running time is

$$2^{n\left(1-\frac{1}{\ln(m/n)+O(\ln\ln m)}\right)}$$

up to a polynomial factor. This bound matches the best known upper bound for randomized SAT algorithms [5]. In comparison with the randomized algorithm in [5], our deterministic algorithm is simpler and more intuitive.

Clause shortening approach. Our algorithm employs the *clause shortening* technique first used by Schuler [12] in his randomized algorithm. This technique is based on the following idea:

> For any "long" clause (longer than some k), either we can shorten this clause by choosing any k literals in the clause and dropping the other literals, or we can substitute false for these k literals in the entire formula.

Schuler's algorithm shortens every clause to its first k literals and applies the k-SAT algorithm [9] to the resulting k-CNF formula. If no satisfying assignment is found, Schuler's algorithm simplifies the initial formula by choosing a long clause at random and substituting false for its first k literals. This procedure is recursively applied to the simplified formula until no clause contains more than k literals. The upper bound in [12] is obtained when taking $k = \log(2m)$.

The derandomization [4] of Schuler's algorithm uses the same idea. Let F be an input formula consisting of clauses $C_1, \ldots, C_m$. Assume that the first m' clauses are longer than k and the other clauses have length $\leq k$. For each C_i where $i \leq m'$, let D_i be the clause that is made up from the first k literals of C_i. Then F is equivalent to the disjunction of the following $m' + 1$ formulas:

$$\begin{aligned} F_1 &= F\ [D_1 = \mathsf{false}] \\ &\vdots \\ F_{m'} &= F\ [D_{m'} = \mathsf{false}] \\ F_{m'+1} &= D_1 \wedge \ldots \wedge D_{m'} \wedge T \end{aligned}$$

where T is $C_{m'+1} \wedge \ldots \wedge C_m$, i.e., T is the "tail" consisting of "short" clauses. The derandomized algorithm first tests satisfiability of $F_{m'+1}$ using a k-SAT subroutine. If no satisfying assignment is found, the algorithm is recursively applied to each of $F_1, \ldots, F_{m'}$.

Clause shortening combined with pruning. There is some inefficiency in the derandomized version of Schuler's algorithm. Namely, when testing F_i, we may have to test its subformula corresponding to $D_j = \mathsf{false}$. On the other hand, when testing F_j, we may come to the same subformula. To eliminate this inefficiency, we prune the tree of recursively tested formulas as follows: for each formula F_i, we replace all clauses $C_1, \ldots, C_{i-1}$ by their counterparts $D_1, \ldots, D_{i-1}$. In other words, we use the fact that F is equivalent to the disjunction of the following formulas:

$$\begin{array}{lcl} F_1 &=& (C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_{m'-1} \wedge C_{m'} \wedge T)\ [D_1 = \mathsf{false}] \\ F_2 &=& (D_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_{m'-1} \wedge C_{m'} \wedge T)\ [D_2 = \mathsf{false}] \\ F_3 &=& (D_1 \wedge D_2 \wedge C_3 \wedge \ldots \wedge C_{m'-1} \wedge C_{m'} \wedge T)\ [D_3 = \mathsf{false}] \\ \vdots && \\ F_{m'} &=& (D_1 \wedge D_2 \wedge D_3 \wedge \ldots \wedge D_{m'-1} \wedge C_{m'} \wedge T)\ [D_{m'} = \mathsf{false}] \\ F_{m'+1} &=& (D_1 \wedge D_2 \wedge D_3 \wedge \ldots \wedge D_{m'-1} \wedge D_{m'} \wedge T) \end{array}$$

Similarly to the derandomization above, our algorithm first tests $F_{m'+1}$ and then, if no satisfying assignment is found, it tests each of $F_1, \ldots, F_{m'}$. We give details of our algorithm in Sect. 3 and prove its worst-case upper bound in Sect. 4.

2 Definitions and Notation

We deal with Boolean formulas in conjunctive normal form (CNF). By a *variable* we mean a Boolean variable that takes truth values true or false. A *literal* is a variable x or its negation $\neg x$. A *clause* C is a set of literals such that C contains no complementary literals. A *formula* F is a set of clauses; n and m denote, respectively, the number of variables and the number of clauses in F. If each clause in F contains at most k literals, we say that F is a k-*CNF formula*.

An *assignment* to variables $x_1, \ldots, x_n$ is a mapping from $\{x_1, \ldots, x_n\}$ to $\{\mathsf{true}, \mathsf{false}\}$. This mapping is extended to literals: each literal $\neg x_i$ is mapped to the complement of the truth value assigned to x_i. We say that a clause C is *satisfied* by an assignment A if A assigns true to at least one literal in C. The formula F is *satisfied* by A if every clause in F is satisfied by A. In this case, A is called a *satisfying* assignment for F. We consider substitutions of truth values for some variables in a formula. If D is a set of literals, we write $F[D = \mathsf{false}]$ to denote the formula obtained from F as follows: any clause that contains the negation of a literal in D is removed from F, the literals occurring in D are deleted from the other clauses.

Here is a summary of the notation used in the paper.

- F denotes a CNF formula; n denotes the number of variables in F; m denotes the number of clauses in F.
- If C is a clause then $|C|$ denotes its length (the number of literals).
- We write $\log x$ to denote $\log_2 x$.
- $H(x)$ denotes the binary entropy function: $H(x) = -x \log x - (1-x)\log(1-x)$.

3 Algorithm

We describe an algorithm parameterized by a function $k(n, m)$. This function determines the length to which input clauses are to be shortened. The algorithm computes the value of $k(n, m)$ for particular n and m, then it runs a recursive procedure that implements the clause shortening approach combined with pruning. This recursive `Procedure` $\mathcal{S}$ described below uses a k-SAT algorithm of [2] as a subroutine.

Lemma 1 ([2])**.** There exists a deterministic algorithm that tests satisfiability of an input formula F in time at most

$$m \cdot q(n) \cdot \left(2 - \frac{2}{k+1}\right)^n$$

where $q(n)$ is a polynomial in n, and k is the maximum length of clauses in F.

`Procedure` $\mathcal{S}$

`Input:` a CNF formula F and a positive integer k.

1. Assume F consists of clauses $C_1, \ldots, C_m$. Change each clause C_i to a clause D_i as follows: If $|C_i| > k$ then choose any k literals in C_i and drop the other literals; otherwise leave C_i as is, i.e., $D_i = C_i$. Let F' denote the resulting formula.

2. Test satisfiability of F' using the algorithm defined in Lemma 1.

3. If F' is satisfiable, output "satisfiable" and halt. Otherwise, for each i, do the following:

   (a) Convert F to F_i as follows:

      i. Replace C_j by D_j for all $j < i$;

      ii. Assign false to all literals in D_i.

   (b) Recursively invoke `Procedure` $\mathcal{S}$ on (F_i, k).

4. Return "unsatisfiable".

`Algorithm` $\mathcal{A}_{k(n,m)}$

`Parameter:` a positive integer function $k(n, m)$

`Input:` a CNF formula F with m clauses over n variables ($n \leq m$)

1. Compute $k = k(n, m)$.

2. Invoke `Procedure` $\mathcal{S}$ on (F, k).

4 Upper Bound

First we give an upper bound for `Algorithm` $\mathcal{A}_{k(n,m)}$. Then we find a particular function $k(n, m)$ that approximately minimizes this upper bound.

Theorem 1. Let $k(n, m)$ be an integer function such that:

$$3 \leq k(m, n) \leq \log m. \tag{1}$$

Then `Algorithm` $\mathcal{A}_{k(n,m)}$ runs in time

$$O(\sqrt{m}) \cdot \tfrac{n}{k} \cdot q(n) \cdot 2^{n\left(1-\frac{\log e}{k+1}\right)+O(m \cdot 2^{-k})}, \tag{2}$$

where $q(n)$ is the polynomial appearing in Lemma 1.

Proof. Let $t(F)$ be the running time of `Procedure` $\mathcal{S}$ on (F, k). It is not difficult to see that $t(F)$ can be estimated as follows:

$$t(F) \leq t_0(F') + \sum_{i=1}^{m} t(F_i) \tag{3}$$

where F' and F_i are as described in `Procedure` $\mathcal{S}$, and $t_0(F')$ is the running time of the k-SAT algorithm from Lemma 1 on F'. Let $T(n, m, m')$ denote the maximum of the running time of `Procedure` $\mathcal{S}$ on (G, k) where G is a formula with $\leq n$ variables and $\leq m$ clauses such that at most m' of its clauses contain $> k$ literals. For the k-SAT algorithm, we define $T_0(n, m)$ as the maximum running time on a different set of formulas, namely let $T_0(n, m)$ be the maximum running time of the algorithm from Lemma 1 on the set of formulas F' such that each F' has $\leq m$ clauses over $\leq n$ variables and the maximum length of clauses is not greater than k.

Then for any n and m, inequality (3) implies the following recurrence relation:

$$T(n, m, m') \leq T_0(n, m) + \sum_{i=0}^{m-1} T(n-k, m, m'-i). \tag{4}$$

If we iteratively substitute $T(n-L, m, m'-i)$ into this recurrence, we turn its right-hand side into the sum of terms of the form $T_0(n-lk, m)$ for $l \leq n/k$.

Our proof strategy is as follows. We consider the recursion tree of our algorithm and estimate the total amount T_l of work done at its l-th level (i.e., the sum of terms $T_0(n-lk, m)$). We then find l^* that maximizes this estimation. The total running time is then at most n/k times the estimation for the level l^*.

To estimate T_l, we note that the number of nodes at the l-th level

$$\sum_{i_1=1}^{m} \sum_{i_2=1}^{i_1} \cdots \sum_{i_l=1}^{i_{l-1}} 1$$

is the number of ways to choose l possibly equal elements out of m, i.e., $\binom{m+l-1}{l}$ (see, e.g., [13, Sect. 1.2]). Then

$$T_l \;\leq\; m \cdot q(n) \cdot \left(2 - \tfrac{2}{k+1}\right)^{n-lk} \cdot \binom{m+l-1}{l}. \tag{5}$$

Let E_l denote the right-hand side of the estimation (5). It is straightforward to see that $E_{l+1} \leq E_l$ if and only if

$$\tfrac{m+l}{l+1} \cdot \left(2 - \tfrac{2}{k+1}\right)^{-k} \leq 1,$$

which is equivalent to

$$\tfrac{m+l}{l+1} \cdot 2^{-k} \cdot \left(1 + \tfrac{1}{k}\right)^{k} \leq 1.$$

Therefore, the maximum of E_l over l is attained at the following integer l^*:

$$l^* = \tfrac{m\alpha - 2^k}{2^k - \alpha} + \delta,$$

where $\alpha = (1 + 1/k)^k$ and $-1 < \delta < 1$.

The next step is to give lower and upper bounds on l^*. We prove that

$$m \cdot 2^{-k} \;\leq\; l^* \;\leq 5.12 \cdot m \cdot 2^{-k} \tag{6}$$

To prove the lower bound, we use $k \leq \log m$ and $\alpha \geq (1 + 1/3)^3 \approx 2.37$ (which follows from $k \geq 3$):

$$\begin{aligned} l^* &= \tfrac{m\alpha - 2^k}{2^k - \alpha} + \delta \\ &\geq m \cdot 2^{-k} \cdot \left(\tfrac{\alpha - 2^k/m}{1 - \alpha/2^k}\right) - 1 \\ &\geq m \cdot 2^{-k} \cdot \left(\tfrac{\alpha - 1}{1}\right) - 1 \\ &\geq m \cdot 2^{-k}. \end{aligned}$$

The upper bound is proved using condition (1) and $\alpha < e$. Indeed,

$$\begin{aligned} l^* &= \tfrac{m\alpha - 2^k}{2^k - \alpha} + \delta \\ &\le m \cdot 2^{-k} \cdot \left(\tfrac{\alpha - 2^k/m}{1-\alpha/2^k} \right) + 1 \\ &\le m \cdot 2^{-k} \cdot \left(\tfrac{e}{1-e/8} \right) + 1 \\ &\le m \cdot 2^{-k} \cdot \left(\tfrac{e}{1-e/8} + 1 \right) \\ &\le 5.12 \cdot m \cdot 2^{-k}. \end{aligned}$$

Now we estimate the total amount of work done at level the l^*:

$$E_{l^*} \;=\; m \cdot q(n) \cdot 2^{n-kl^*} \cdot \left(1 - \tfrac{1}{k+1}\right)^{n-kl^*} \cdot \tbinom{m+l^*-1}{l^*}. \tag{7}$$

The last factor in the right-hand side of (7) can be estimated using Stirling's approximation as in [6, exercise 9.42]:

$$\begin{aligned} \tbinom{m+l^*-1}{l^*} &= O\left(\tfrac{1}{\sqrt{m+l^*}}\right) \cdot 2^{H\left(\frac{l^*}{m+l^*-1}\right)(m+l^*-1)} \\ &= O\left(\tfrac{1}{\sqrt{m}}\right) \cdot e^{-l^* \ln \frac{l^*}{m+l^*-1} - (m-1)\ln \frac{m-1}{m+l^*-1}}. \end{aligned}$$

Using $l^* - 1 < m$ and $\ln(1+x) < x$, we have

$$\begin{aligned} \tbinom{m+l^*-1}{l^*} &= O\left(\tfrac{1}{\sqrt{m}}\right) \cdot e^{l^* \ln \frac{m}{l^*} + l^* \ln\left(1 + \frac{l^*-1}{m}\right) + (m-1)\ln\left(1+\frac{l^*}{m-1}\right)} \\ &= O\left(\tfrac{1}{\sqrt{m}}\right) \cdot e^{l^*\left(\ln \frac{m}{l^*} + 2\right)}. \end{aligned}$$

The factor $\left(1 - \frac{1}{k+1}\right)^{n-kl^*}$ in (7) can be estimated using the inequality $\ln(1-x) < -x$:

$$\left(1 - \tfrac{1}{k+1}\right)^{n-kl^*} \;=\; e^{(n-kl^*)\ln\left(1-\frac{1}{k+1}\right)} \;\le\; e^{-\frac{n-kl^*}{k+1}} \;<\; e^{-\frac{n}{k+1}+l^*}.$$

Hence, we can estimate E_{l^*} as follows:

$$\begin{aligned} E_{l^*} &\le O(\sqrt{m}) \cdot q(n) \cdot 2^{n-kl^*} \cdot e^{-\frac{n}{k+1}+l^*} \cdot e^{l^*\left(\ln \frac{m}{l^*}+2\right)} \\ &= O(\sqrt{m}) \cdot q(n) \cdot 2^n \cdot 2^{-\frac{n \log e}{k+1}} \cdot e^{-kl^* \ln 2} \cdot e^{l^*} \cdot e^{l^*\left(\ln \frac{m}{l^*}+2\right)} \\ &= O(\sqrt{m}) \cdot q(n) \cdot 2^{n\left(1 - \frac{\log e}{k+1}\right)} \cdot e^{\beta l^*}, \end{aligned}$$

where

$$\beta \;=\; 3 + \ln \tfrac{m}{l^*} - k \ln 2 \;=\; 3 + \ln \tfrac{m}{2^k \cdot l^*}.$$

The lower bound on l^* in (6) implies $\beta < 3$. Therefore, using the upper bound in (6), we have

$$\begin{aligned} E_{l^*} &\le O(\sqrt{m}) \cdot q(n) \cdot 2^{n\left(1-\frac{\log e}{k+1}\right)} \cdot e^{3l^*} \\ &\le O(\sqrt{m}) \cdot q(n) \cdot 2^{n\left(1-\frac{\log e}{k+1}\right)} \cdot e^{3\cdot(5.12 \cdot m \cdot 2^{-k})} \\ &\le O(\sqrt{m}) \cdot q(n) \cdot 2^{n\left(1-\frac{\log e}{k+1}\right)} \cdot 2^{O(1)\cdot m \cdot 2^{-k}}. \end{aligned}$$

□

Remark 1. What value of k minimizes bound (2)? Straightforward differentiation of the exponent

$$n\left(1 - \frac{\log e}{k+1}\right) + O(m \cdot 2^{-k})$$

gives the following equation:

$$k = \log(m/n) + 2\log(k+1) + O(1).$$

We can approximate a fix-point solution to this equation taking

$$k = \log(m/n) + d \cdot \log\log m$$

where $d > 1$ is a constant close to 1.

Theorem 2. For any number $d > 1$, let $\mathcal{A}_d$ be an algorithm obtained from `Algorithm` $\mathcal{A}_{k(m,n)}$ by taking the following function $k(m, n)$:

$$k(m,n) = \begin{cases} \lfloor \log(m/n) + d \cdot \log\log m \rfloor & \text{if } \log m < n^{1/d}, \\ \lfloor \log m \rfloor & \text{otherwise.} \end{cases}$$

Then $\mathcal{A}_d$ runs in time

$$O(\sqrt{m}) \cdot \tfrac{n}{k} \cdot q(n) \cdot 2^{n\left(1 - \frac{1}{\ln(m/n) + d \cdot \ln\log m} + o\left(\frac{1}{k}\right)\right)} \tag{8}$$

on formulas such that $\log m < n^{1/d}$ and runs in time

$$O(\sqrt{m}) \cdot \tfrac{n}{k} \cdot q(n) \cdot 2^{n\left(1 - \frac{1}{\ln(2m)}\right)} \tag{9}$$

on all other formulas, where $q(n)$ is the polynomial from Lemma 1.

Proof. We prove both bounds by applying Theorem 1. Note that the function $k(m, n)$ defined in the claim satisfies the inequality $k \le \log m$ required by Theorem 1. This is obvious for $k = \lfloor \log m \rfloor$ and follows from $\log m < n^{1/d}$ for

$$k = \lfloor \log(m/n) + d \cdot \log\log m \rfloor. \tag{10}$$

To prove bound (8), we first write the upper bound given by Theorem 1 in the following form:

$$O(\sqrt{m}) \cdot \tfrac{n}{k} \cdot q(n) \cdot 2^{n(1-\gamma)}, \text{ where } \gamma = \tfrac{\log e}{k+1} - \tfrac{O(1) \cdot m}{n \cdot 2^k}.$$

Substituting the value of k from (10) in the second term of γ, we have

$$\begin{aligned} \gamma &\ge \tfrac{\log e}{k+1} - \tfrac{O(1)}{(\log m)^d} \\ &\ge \tfrac{\log e}{k} - \tfrac{\log e}{k(k+1)} - \tfrac{O(1)}{(\log m)^d} \\ &\ge \tfrac{\log e}{k} - o\left(\tfrac{1}{k}\right) \quad \text{using } k \le \log m \text{ and } d > 1 \\ &\ge \tfrac{1}{\ln(m/n) + d \cdot \ln\log m} - o\left(\tfrac{1}{k}\right). \end{aligned}$$

Bound (9) is easily obtained from the upper bound given by Theorem 1 by substitution of $\lfloor \log m \rfloor$ for k.

□

Remark 2. Both bounds (8) and (9) hold for all formulas. Bound (8) is asymptotically better for formulas such that $\log m < n^{1/d}$, while bound (9) is better for all other formulas.

Remark 3. What is the best value of d? On the one hand, the smaller d is, the smaller k we have, which yields a better asymptotics of bound (8). In addition, the smaller d is, the weaker the $\log m \leq n^{1/d}$ restriction becomes. On the other hand, the smaller d we take, the slower $o(1/k)$ tends to zero (or, equivalently, the asymptotic behavior starts with lager values of m).

Remark 4. The randomized algorithm for SAT in [5] runs in time

$$2^{n\left(1-\frac{1}{\ln(m/n)+O(\ln\ln m)}\right)}$$

up to a polynomial factor. It is straightforward to check that for any $d > 1$, the exponential part of the bound in Theorem 2 also can be written in this form, i.e., our upper bound for deterministic algorithms matches the best known upper bound for randomized algorithms.

Acknowledgement. We thank Natalia Tsilevich for her contribution to the proof of Theorem 1 and for helpful discussions.

References

[1] T. Brueggemann and W. Kern. An improved local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1-3):303–313, December 2004.

[2] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

[3] E. Dantsin, E. A. Hirsch, and A. Wolpert. Algorithms for SAT based on search in Hamming balls. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science, STACS 2004*, volume 2996 of *Lecture Notes in Computer Science*, pages 141–151. Springer, March 2004.

[4] E. Dantsin and A. Wolpert. Derandomization of Schuler's algorithm for SAT. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing, SAT 2004*, volume 3542 of *Lecture Notes in Computer Science*, pages 80–88. Springer, 2005.

[5] E. Dantsin and A. Wolpert. An improved upper bound for SAT. In *Proceedings of the 8th International Conference on Theory and Applications on Satisfiability Testing, SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 400–407. Springer, June 2005.

[6] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 2nd edition, 1994.

[7] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, page 328, January 2004. A preliminary version appeared in Electronic Colloquium on Computational Complexity, Report No. 53, July 2003.

[8] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, FOCS'98*, pages 628–637, 1998.

[9] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97*, pages 566–574, 1997.

[10] P. Pudlák. Satisfiability – algorithms and logic. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS'98*, volume 1450 of *Lecture Notes in Computer Science*, pages 129–141. Springer, 1998.

[11] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS'99*, pages 410–414, 1999.

[12] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, 54(1):40–44, January 2005. A preliminary version appeared as a technical report in 2003.

[13] R. P. Stanley. *Enumerative combinatorics.* Wadsworth & Brooks/Cole Advanced Books & Software, 1986.

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'