Electronic Colloquium on Computational Complexity, Report No. 17 (2004)

Derandomization of Schuler's Algorithm for SAT

Evgeny Dantsin Alexander Wolpert

School of Computer Science, Roosevelt University
430 S. Michigan Av., Chicago, IL 60605, USA
{edantsin,awolpert}@roosevelt.edu

Abstract

Recently Schuler [Sch03] presented a randomized algorithm that solves SAT in expected time at most $2^{n(1-1/\log_2(2m))}$ up to a polynomial factor, where n and m are, respectively, the number of variables and the number of clauses in the input formula. This bound is the best known upper bound for testing satisfiability of formulas in CNF with no restriction on clause length (for the case when m is not too large comparing to n). We derandomize this algorithm using deterministic k-SAT algorithms based on search in Hamming balls, and we prove that our deterministic algorithm has the same upper bound on the running time as Schuler's randomized algorithm.

1 Introduction

Known upper bounds.

There has been recent progress in developing algorithms for SAT and k-SAT that have "record" worst-case upper bounds on the running time. Typically, such bounds have the form α^n up to a polynomial factor, where n is the number of variables in the input formula, and $\alpha < 2$ is either a constant or depends on some parameters of the input. The currently best known upper bounds are listed as follows (we give only the exponential parts of the bounds, omitting polynomial factors):

Randomized algorithms for k-SAT. The bounds for 3-SAT and 4-SAT are 1.324^n and 1.474^n respectively [IT03]. These and other recent bounds for 3-SAT, e.g., [BS03, HSSW02, Rol03], are obtained using algorithms based on the multistart local search [Sch99, Sch02] or on the randomized DPLL approach [PPZ97, PPSZ98]. The bounds for $k > 4$ are also obtained with these approaches:

$(2-2/k)^n$ [Sch99, Sch02];
$2^{n(1-\epsilon)+o(n)}$ where $\epsilon = \mu_k/(k-1)$ and $\lim_{k\to\infty} \mu_k = \pi^2/6$ [PPSZ98].

Deterministic algorithms for k-SAT. The algorithms [DGHS00, DGH+02] cover the Boolean cube $\{0,1\}^n$ by Hamming balls and search for a satisfying assignment inside these balls. The have the bound

$$(2-2/(k+1))^n.$$

For $k = 3$, the bound can be improved to 1.481^n.

Randomized algorithms for SAT (no limit on clause length). The algorithm [Sch03] uses the randomized DPLL approach and has the bound

$2^{n(1-1/\log_2(2m))}$ where m is the number of clauses in the input formula.

ISSN 1433-8092

Also, there is another bound: $2^{n-c\sqrt{n}}$, where c is a constant. This bound is obtained using two different algorithms: [Pud98] based on the randomized DPLL approach and [DHW03] based on search in Hamming balls. Schuler's bound [Sch03] is more interesting because it is better than $2^{n-c\sqrt{n}}$ for the case when m is not too large comparing to n, namely when $m = o(2^{\sqrt{n}})$.

Deterministic algorithms for SAT (no limit on clause length). The algorithm [DHW03] based in search in Hamming balls has the bound

$$2^{n-2\sqrt{n/\log_2 m}}.$$

There are also other types of bounds that are "more" dependent on the number of clauses or other input parameters, e.g., 1.239^m [Hir00]. In this paper we give a deterministic algorithm that has the same bound

$$2^{n(1-1/\log_2(2m))}$$

as in the case of randomized algorithms for SAT.

Our result.

We prove that SAT can be solved by a deterministic algorithm with the same upper bound on the running time as Schuler's randomized algorithm, i.e., with the bound $2^{n(1-1/\log_2(2m))}$ up to a polynomial factor.

Like Schuler's algorithm, our deterministic algorithm can be described in terms of two algorithms $\mathcal{M}$ (stands for *Main*) and $\mathcal{S}$ (stands for *Subroutine*). The algorithm $\mathcal{S}$ is used to test satisfiability of formulas with "short" clauses (of length at most $\log(2m)$). The algorithm $\mathcal{M}$ is the main algorithm that transforms an input formula F into to F' by "shortening" the clauses in F. Then $\mathcal{M}$ invokes $\mathcal{S}$ to check whether F' is satisfiable. If so, we are done. Otherwise, the algorithm $\mathcal{M}$ simplifies F and recursively invokes itself on the results of simplification.

Theorem 1 in Sect. 3 gives an upper bound on the running time of the algorithm $\mathcal{M}$ under an assumption on the running time of the subroutine $\mathcal{S}$. More exactly, the assumption is that $\mathcal{S}$ runs in time at most $2^{n(1-1/k)}$ up to a polynomial factor, where k is the maximum length of clauses in F. Then $\mathcal{M}$ runs in time at most $2^{n(1-1/\log_2(2m))}$ up to a polynomial factor. Does there exist any deterministic subroutine $\mathcal{S}$ that meets this assumption? The answer is positive (Theorem 2): the algorithms [DGH+02] have the required upper bound on the running time. Thus, taking any of them as the subroutine $\mathcal{S}$, we obtain a deterministic algorithm that solves SAT with the bound $2^{n(1-1/\log_2(2m))}$.

Notation

By a *formula* we mean Boolean formulas in conjunctive normal form (CNF) defined as follows. A *literal* is a Boolean variable x or its negation $\neg x$. A *clause* is a finite set C of literals such that C contains no opposite literals. The *length* of C (denoted by $|C|$) is the number of literals in C. A *formula* is a set of clauses. An *assignment* to variables $x_1, \ldots, x_n$ is a mapping from $\{x_1, \ldots, x_n\}$ to the truth values $\{\text{TRUE}, \text{FALSE}\}$. This mapping is extended to literals: each literal $\neg x_i$ is mapped to the truth value opposite to the 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.

By *SAT* we mean the following computational problem: Given a formula F in CNF, decide whether F is satisfiable or not. The *k-SAT* problem is the restricted version of SAT that allows only clauses of length at most k.

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

- F denotes a formula;
- n denotes the number of variables in F;
- m denotes the number of clauses in F;

- k denotes the maximum length of clauses in F;
- $|C|$ denotes the length of clause C;
- $\log x$ denotes $\log_2 x$.

2 Algorithms Based on Clause Shortening

Schuler's algorithm.

We first sketch Schuler's algorithm [Sch03]. The algorithm is based on a randomized satisfiability-testing procedure $\mathcal{R}$ that runs in polynomial time and finds a satisfying assignment (if any) with probability at least $2^{-n(1-1/\log(2m))}$. This probability can be increased to a constant by repetitions in the usual way.

Let F be an input formula consisting of clauses $C_1, \ldots, C_m$. Assuming that F is satisfied by an (unknown) assignment A, we show how $\mathcal{R}$ finds A. The first step of $\mathcal{R}$ is to shorten the clauses in F as follows:

1. For each clause C_i such that $|C_i| > \log(2m)$, choose any $\log(2m)$ literals in C_i and delete the other literals.
2. Leave the shorter clauses as is.

Let $F' = \{D_1, \ldots, D_m\}$ be the result of the shortening. The next step of $\mathcal{R}$ is to apply the randomized polynomial-time k-SAT algorithm [PPZ97] to F' with $k = \log(2m)$. There are two possible cases:

Case 1. A satisfies F'. In this case, the procedure [PPZ97] finds A with probability at least $2^{-n(1-1/\log(2m))}$.

Case 2. A does not satisfy F'. Then there is a clause D_i such that all of its literals are false under A. Therefore, if we "guess" this clause correctly, we may simplify F by assigning the corresponding values to the variables occurring in D_i. We choose a clause D_j in F' uniformly at random. The probability that we have "guessed' the clause correctly (i.e., $j = i$) is at least $1/m$. Then we simplify F': for each literal l in D_j, we remove all clauses that contain $\neg l$ and delete l from the remaining clauses. Finally, we recursively apply $\mathcal{R}$ to the result of the simplification.

The analysis of $\mathcal{R}$ in [Sch03] shows that $\mathcal{R}$ finds A with the required probability. Note that the same bound holds if the subroutine [PPZ97] is replaced by another subroutine that finds a satisfying assignment with the same or higher probability, for example by Schöning's algorithm [Sch99, Sch02].

Algorithms $\mathcal{M}$ (Main) and $\mathcal{S}$ (Subroutine).

Schuler's algorithm invokes the algorithm [PPZ97] for testing satisfiability of formulas with "short" clauses. Our derandomized version will also use a subroutine to check formulas with "short" clauses. However, we first describe our algorithm without specifying the invoked subroutine. That is, assuming that $\mathcal{S}$ is an arbitrary satisfiability-testing algorithm, we define our main algorithm $\mathcal{M}$ as a procedure that invokes $\mathcal{S}$ as a subroutine.

Algorithm $\mathcal{S}$

Input: Formula F (with no restriction on clause length).
Output: Satisfying assignment or "no".
Any method of testing satisiability.

Algorithm $\mathcal{M}$

Input: Formula F with clauses $C_1, \ldots, C_m$ over n variables.
Output: Satisfying assignment or "no".

1. Change each clause C_i to a clause D_i as follows: If $|C_i| > \log(2m)$ then choose any $\log(2m)$ literals in C_i and delete 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 $\mathcal{S}$.
3. If F' is satisfiable, return the satisfying assignment found in the previous step. Otherwise, for each clause D_i in F', do the following:
   (a) Convert F to F_i by assigning FALSE to all literals in D_i. Namely, for each literal l in D_i, remove all clauses containing $\neg l$ and delete l from the remaining clauses.
   (b) Recursively invoke $\mathcal{M}$ on F_i.
4. Return "no".

3 Bound for SAT

The choice of the subroutine $\mathcal{S}$ determines the main algorithm $\mathcal{M}$. In this section, we specify $\mathcal{S}$ so that the algorithm $\mathcal{M}$ solves SAT in time at most $2^{n(1-1/(\log(2m)))}$ up to a polynomial factor. First, we prove an upper bound for $\mathcal{M}$ assuming a specific upper bound on the running time of $\mathcal{S}$. Then we choose a subroutine that meets this assumption. As a result, we obtain the claimed upper bound for the main algorithm $\mathcal{M}$.

Theorem 1. Suppose that for any formula F, the algorithm $\mathcal{S}$ runs on F in time at most $2^{n(1-1/k)}$ up to a polynomial factor, where k is the maximum length of clauses in F. Then the running time of the algorithm $\mathcal{M}$ is at most $2^{n(1-1/\log(2m))}$ up to a polynomial factor.

Proof. Let $t_{\mathcal{S}}(F)$ and $t_{\mathcal{M}}(F)$ be, respectively, the running times of the algorithms $\mathcal{S}$ and $\mathcal{M}$ on a formula F. It is not difficult to see that $t_{\mathcal{M}}(F)$ can be estimated (up to a polynomial factor) as follows:

$$t_{\mathcal{M}}(F) \leq t_{\mathcal{S}}(F') + m \cdot t_{\mathcal{M}}(F_i) \tag{1}$$

where F' and F_i are as described in the algorithm $\mathcal{M}$. Let $T_{\mathcal{M}}(n,m)$ denote the maximum of the running time of $\mathcal{M}$ on formulas with m clauses over n variables. For the subroutine $\mathcal{S}$, we define $T_{\mathcal{S}}(n,m)$ as the maximum running time on a different set of formulas, namely let $T_{\mathcal{S}}(n,m)$ be the maximum of the running time of $\mathcal{S}$ on the set of formulas F such that each F has m clauses over n variables and the maximum length of clauses is not greater than $\log(2m)$. Let L denote $\log(2m)$. Then for any n and m, the inequality (1) implies the following recurrence relation:

$$T_{\mathcal{M}}(n,m) \leq T_{\mathcal{S}}(n,m) + m \cdot T_{\mathcal{M}}(n-L,m)$$

Iterating this recurrence and using the bound on $t_{\mathcal{S}}(F)$ with $k \leq L$, we get (again up to a polynomial factor)

$$\begin{aligned} T_{\mathcal{M}}(n,m) &\leq \sum_{i=0}^{n/L} m^i \cdot T_{\mathcal{S}}(n-iL,m) \\ &\leq \sum_{i=0}^{n/L} m^i \cdot 2^{(n-iL)(1-1/L)} = 2^{n(1-1/L)} \sum_{i=0}^{n/L} \left(m \cdot 2^{1-L}\right)^i \end{aligned}$$

Since $L = \log(2m)$, we have $m\,2^{1-L} = 1$. Therefore,

$$t_{\mathcal{M}}(F) \leq T_{\mathcal{M}}(n,m) \leq 2^{n(1-1/L)}$$

up to a polynomial factor. □

Theorem 2 (based on [DGH$^+$02]). There exists a deterministic algorithm that tests satisfiability of an input formula F in time at most $2^{n(1-1/k)}$ up to a polynomial factor, where n is the number of variables in F, and k is the maximum length of clauses in F.

Proof. Paper [DGH$^+$02] defines two algorithms that can be applied to any formula. Their running times are estimated in terms of the maximum length of clauses in the input formula (thus, they can be viewed as algorithms for k-SAT). Both algorithms cover the Boolean cube $\{0,1\}^n$ by Hamming balls and search for a satisfying assignment inside these balls. The first algorithm runs in time at most $2^{n(1-\log(1+1/k))}$ up to a polynomial factor (Theorem 1 in [DGH$^+$02]). Since

$$\log(1+1/k) = (\log e)/k + o(1/k),$$

this algorithm meets the claim. The second algorithm has a parameter δ; its running time is at most

$$2^{n(1-\log(1+1/k)+\delta)}$$

up to a polynomial factor (Theorem 2 in [DGH$^+$02]). Taking $\delta \le \frac{\log e/2}{k}$, we have

$$2^{n\left(1-\log\left(1+\frac{1}{k}\right)+\delta\right)} \;\le\; 2^{n\left(1-\frac{\log e}{k}+\frac{\log e/2}{k}\right)} \;\le\; 2^{n(1-1/k)}.$$

Hence, the second algorithm also meets the claim.

The algorithms differ in the construction of the covering of $\{0,1\}^n$ by Hamming balls. The first algorithm uses a greedy method to constuct the covering that is minimal up to a polynomial factor. The construction requires an exponential space (approximately $2^{n/6}$). The second algorithm constructs a "nearly minimal" covering, i.e., a covering that is minimal up to a factor of $2^{\delta n}$, where δ can be chosen arbitrary small.

To estimate the space used by the second algorithm, we have to consider details of how it constructs the covering of $\{0,1\}^n$. Each ball center is the concatenation of n/b blocks of length b (Lemma 7 in [DGH$^+$02]). The algorithm constructs a covering code $\mathcal{C}$ of length b for blocks. Then, keeping this code in memory, the algorithm generates code words of length n (centers of balls) one by one. An upper bound on the space can be estimated as the cardinality of the covering code $\mathcal{C}$ for blocks. Using Lemma 4 in [DGH$^+$02], we can estimate the cadrinality $|\mathcal{C}|$ as follows:

$$|\mathcal{C}| \;\le\; b\sqrt{b}\; 2^{b\left(1-H\left(\frac{1}{k+1}\right)\right)} \tag{2}$$

where $H(x) = -x\log x - (1-x)\log(1-x)$ is the binary entropy function. To obtain the desired upper bound on the running time, we should choose b so that

$$|\mathcal{C}|^{n/b} \;\le\; 2^{n\left(1-H\left(\frac{1}{k+1}\right)+\delta\right)}. \tag{3}$$

Using the bound (2) and the inequality (3), we get the following constraint on b:

$$\left(b\sqrt{b}\; 2^{b\left(1-H\left(\frac{1}{k+1}\right)\right)}\right)^{n/b} \;\le\; 2^{n\left(1-H\left(\frac{1}{k+1}\right)+\delta\right)} \tag{4}$$

which is equivalent to $(b\sqrt{b})^{1/b} \le 2^{\delta}$. Now we substitute $\delta = \frac{\log(e/2)}{k}$ and take $b = 4k\log k$. Then (4) holds for all sufficiently large k. Therefore, we can use blocks of length $4k\log k$. In fact, the algorithm will be applied to formulas with $k = \log(2m)$, which gives the upper bound $(2m)^{4\log\log(2m)}$ on the space. □

Theorem 3. Suppose that the algorithm $\mathcal{M}$ uses the algorithm from Theorem 2 as the subroutine $\mathcal{S}$. Then $\mathcal{M}$ tests satisfiability of an input formula F with m clauses over n variables in time at most $2^{n(1-1/(\log(2m)))}$ up to a polynomial factor.

Proof. Immediately follows from Theorems 1 and 2. □

References

[BS03] S. Baumer and R. Schuler. Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. Electronic Colloquium on Computational Complexity, Report No. 10, February 2003.

[DGH+02] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $\left(2 - \frac{2}{k+1}\right)^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, October 2002.

[DGHS00] E. Dantsin, A. Goerdt, E. A. Hirsch, and U. Schöning. Deterministic algorithms for k-SAT based on covering codes and local search. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Proceedings of the 27th International Colloquium on Automata, Languages and Programming, ICALP'2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 236–247. Springer, July 2000.

[DHW03] E. Dantsin, E. A. Hirsch, and A. Wolpert. Algorithms for SAT based on search in Hamming balls. *Electronic Colloquium on Computational Complexity*, Report TR03-072, October 2003.

[Hir00] E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[HSSW02] T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe. A probabilistic 3-SAT algorithm further improved. In H. Alt and A. Ferreira, editors, *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Scienceg, STACS'02*, volume 2285 of *Lecture Notes in Computer Science*, pages 192–202. Springer, March 2002.

[IT03] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. Electronic Colloquium on Computational Complexity, Report No. 53, July 2003.

[PPSZ98] 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.

[PPZ97] 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.

[Pud98] P. Pudlák. Satisfiability — algorithms and logic. In L. Brim, J. Gruska, and J. Zlatuska, editors, *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-Verlag, 1998.

[Rol03] D. Rolf. 3-SAT in $RTIME(O(1.32793^n))$ — improving randomized local search by initializing strings of 3-clauses. Electronic Colloquium on Computational Complexity, Report No. 54, July 2003.

[Sch99] 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.

[Sch02] U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.

[Sch03] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. To appear in *Journal of Algorithms.*, 2003.

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'