A Faster Clause-Shortening Algorithm for SAT with No Restriction on Clause Length

Evgeny Dantsin edantsin@roosevelt.edu
Alexander Wolpert awolpert@roosevelt.edu
Department of Computer Science
Roosevelt University
430 S. Michigan Av.
Chicago, IL 60605, USA

Abstract

We give a randomized algorithm for testing satisfiability of Boolean formulas in conjunctive normal form with no restriction on clause length. This algorithm uses the clause-shortening approach proposed by Schuler [14]. The running time of the algorithm is $O\left(2^{n(1-1/\alpha)}\right)$ where $\alpha = \ln(m/n) + O(\ln\ln m)$ and n, m are respectively the number of variables and the number of clauses in the input formula. This bound is asymptotically better than the previously best known $2^{n(1-1/\log(2m))}$ bound for SAT.[1]

KEYWORDS: *SAT with no restriction on clause length, upper bound, clause shortening*

Submitted July 2005; revised November 2005; published November 2005

1. Introduction

During the past few years there has been considerable progress in obtaining upper bounds on the complexity of solving the Boolean satisfiability problem. This line of research has produced new algorithms for k-SAT. They were further used to prove nontrivial upper bounds for SAT (no restriction on clause length).

Upper bounds for k-SAT. The best known upper bounds for k-SAT are based on two approaches: the satisfiability coding lemma [9, 8] and multistart random walk [12, 13]; both give close upper bounds on the time of solving k-SAT. Schöning's randomized algorithm in [12] has the $(2-2/k)^n$ bound where n is the number of variables in the input formula. The randomized algorithm in [8] has a slightly better bound (this algorithm is often referred to as the PPSZ algorithm). The multistart-random-walk approach is derandomized using covering codes in [3], which gives the best known $(2-2/(k+1))^n$ bound for deterministic k-SAT algorithms. The PPSZ algorithm is derandomized with the same bound for Unique-k-SAT [11]. For small values of k, these bounds are improved: for example, 3-SAT can be solved by a randomized algorithm with the 1.324^n bound [7] and by a deterministic algorithm with the 1.473^n bound [2].

1. The results of this paper (without proofs) appeared in the proceedings of SAT 2005 [6]

Upper bounds for SAT (no restriction on clause length). The first nontrivial upper bound for SAT is given in [10]: the $2^{n(1-1/2\sqrt{n})}$ bound for a randomized algorithm that uses the PPSZ algorithm as a subroutine. A close bound for a deterministic algorithm is proved in [4]. A much better bound for SAT is the $2^{n(1-1/\log(2m))}$ bound, where m is the number of clauses in the input formula. This bound is due to Schuler [14]. His randomized algorithm is based on an approach that can be called *clause shortening*: the algorithm repeatedly applies a k-SAT subroutine to formulas obtained by shortening input clauses to length k. Schuler's algorithm is derandomized in [5]; the derandomization gives a deterministic algorithm that solves SAT with the same bound.

In this paper we improve the $2^{n(1-1/\log(2m))}$ bound: we give a randomized algorithm that solves SAT with the following upper bound on the running time:

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

Idea of the algorithm. Our algorithm for SAT is a repetition of a polynomial-time procedure $\mathcal{P}$ that tests satisfiability of an input formula F. If F is satisfied by a truth assignment A, the procedure $\mathcal{P}$ finds A with probability at least p. As usual, repeating $\mathcal{P}$ on the input formula $O(1/p)$ times, we can find A with a constant probability.

When describing $\mathcal{P}$, we view clauses as sequences (rather than sets) of literals. Each clause $l_1, l_2, \ldots, l_s$ such that $s > k$ is divided into two segments as follows:

$$\textit{First segment: } \boxed{l_1, \ldots, l_k,} \qquad \textit{Second segment: } \boxed{l_{k+1}, \ldots, l_s}$$

We say that the first segment is *true* under an assignment A if at least one of its literals is true under A; otherwise we say that the first segment is *false* under A. If A is clear from the context, we omit the words "under A".

Let F consist of clauses $C_1, \ldots, C_m$ and let A be a fixed satisfying assignment to F. The procedure $\mathcal{P}$ is based on the following dichotomy:

Case 1. The input formula F has "many" clauses in which the first segment is false. We call such clauses *bad*. Suppose that the number of bad clauses is greater than or equal to some d. Then if we choose a clause C_i from $C_1, \ldots, C_m$ at random, the first segment in C_i is false with probability at least d/m. Assuming that C_i is bad, we can simplify F by assigning "false" to all literals occurring in the first segment of C_i.

Case 2. The input formula F has "few" (less than d) bad clauses. If we consider a formula made up of the first segments of all input clauses, we get a k-CNF formula in which "almost" all clauses contain true literals. Only the first segments of bad clauses prevent us from finding A by applying a k-SAT algorithm to this formula. To fix it, we guess all bad clauses and replace their first segments by clauses of length k that contain true literals. More exactly, for each bad clause, we replace its first segment by k literals chosen at random from this clause. The probability that the resulting k-CNF formula is satisfied by A is easily estimated.

The procedure $\mathcal{P}$ first assumes that Case 2 holds. To process this case, $\mathcal{P}$ invokes a subroutine (denoted $\mathcal{S}$) that transforms F into a k-CNF formula and applies a k-SAT algorithm. If no satisfying assignment is found, Case 1 is considered to hold. Then $\mathcal{P}$ simplifies the input formula and recursively invokes itself on the simplified formula.

The procedure $\mathcal{P}$ (and the subroutine $\mathcal{S}$) uses k and d as parameters. Clearly, the success probability of $\mathcal{P}$ depends on values of the parameters. What values of k and d maximize the success probability? In Sect. 4 we find values of the parameters ($k \approx \log(m/n) + O(\log\log(m))$ and $d \approx n/\log^3 m$) that give the following lower bound on the success probability:

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

Sect.2 contains basic definitions and notation. In Sect. 3 we describe the procedure $\mathcal{P}$ and the subroutine $\mathcal{S}$. In Sect. 4 we prove a lower bound on the success probability of $\mathcal{P}$. In Sect. 5 we define our algorithm for SAT and prove the claimed upper bound on its running time.

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 t (true) or f (false). A *literal* is a variable x or its negation $\neg x$. If l is a literal then $\neg l$ denotes its complement, i.e. if l is x then $\neg l$ denotes $\neg x$, and if l is $\neg x$ then $\neg l$ denotes x. Similarly, if v denotes one of the truth values t or f, we write $\neg v$ to denote its complement. 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 $\{\mathsf{t}, \mathsf{f}\}$. 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 (or, C is *true* under A) if A assigns t to at least one literal in C. Otherwise, we say that C is *falsified* by A (or, C is *false* under A). 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.

Let F be a formula and $l_1, \ldots, l_s$ be literals. We write $F[l_1 = \mathsf{f}, \ldots, l_s = \mathsf{f}]$ to denote the formula obtained from F by assigning the value f to all of $l_1, \ldots, l_s$. This formula is obtained from F as follows: the clauses that contain any literal from $\neg l_1, \ldots, \neg l_s$ are deleted from F, and the literals $l_1, \ldots, l_s$ are deleted from the other clauses. Note that $F[l_1 = \mathsf{f}, \ldots, l_s = \mathsf{f}]$ may contain the empty clause or may be the empty formula.

Let A and A' be two assignments that differ only in the values assigned to a literal l. Then we say that A' is obtained from A by *flipping* the value of l.

By *SAT* we mean the following computational problem: Given a formula F in CNF, decide whether F is satisfiable. The *k-SAT* problem is the restricted version of SAT that allows only clauses consisting of at most k literals.

We write $\log x$ to denote $\log_2 x$.

3. Procedure $\mathcal{P}$ and Subroutine $\mathcal{S}$

Both procedures $\mathcal{S}$ and $\mathcal{P}$ use the parameters k and d; their values are determined in the next section.

3.1 Description of $\mathcal{S}$

Suppose that an input formula F has a satisfying assignment A such that F has only "few" ($< d$) clauses in which the first segment is false under A, i.e. Case 2 of the dichotomy holds. Then the subroutine $\mathcal{S}$ finds such an assignment (with the probability estimated in Sect. 4). The subroutine takes two steps:

1. Reduction of F to a k-CNF formula F' such that any satisfying assignment to F' satisfies F.

2. Use of a k-SAT algorithm to find a satisfying assignment to F'.

First, $\mathcal{S}$ guesses all "bad" clauses, i.e. clauses in which the first segment is false under A. More exactly, the subroutine guesses a set $\{B_1, \ldots, B_{d-1}\}$ of clauses such that all "bad" clauses are contained in this set. For each clause B_i, the subroutine guesses k literals in B_i such that this "guessed" set contains a true literal. Then the subroutine replaces each clause in F by a set of k literals chosen from this clause – either the guessed set for $B_1, \ldots, B_{d-1}$ or the first segments for the other clauses in F. The resulting formula is denoted by F'. It is obvious that if we guessed right then A satisfies F'.

To test satisfiability of k-CNF formulas at the second step, $\mathcal{S}$ uses a polynomial-time randomized algorithm that finds a satisfying assignment with an exponentially small probability. We choose Schöning's algorithm [12] to perform this testing (we could use any algorithm that has at least the same success probability, for example the PPSZ algorithm [8]). More exactly, we use "one random walk" of Schöning's algorithm, which has the success probability at least $(2 - 2/k)^{-n}$ up to a constant [13].

If $\mathcal{S}$ finds a satisfying assignment then the procedure returns it and halts. Otherwise, $\mathcal{S}$ should return "no" but for technical reasons, instead of "no", we require $\mathcal{S}$ return F' which is used in the external procedure $\mathcal{P}$.

Note that if $d = 1$, i.e. there is no "bad" clause, then $\mathcal{S}$ simply finds a satisfying assignment to the k-CNF formula made up of the m first segments. Also note that the smaller d, the higher the probability of guessing F'.

```
Subroutine $\mathcal{S}$

    **Input:** Formula $F$ with $m$ clauses over $n$ variables, integers $k$ and $d$.

    **Output:** Satisfying assignment or $k$-CNF formula.

  1. Reduce $F$ to a $k$-CNF formula $F'$ as follows:

     (a) Choose $d-1$ clauses $B_1, \ldots, B_{d-1}$ in $F$ at random.
     (b) Replace each clause that does not belong to $\{B_1, \ldots, B_{d-1}\}$ by its first segment.
     (c) For each $B_i$, choose $k$ literals in $B_i$ at random and replace $B_i$ by the chosen set.
```

2. Test satisfiability of F' using one random walk of length $3n$ (see [12] for details):
   (a) Choose an initial assignment a uniformly at random;
   (b) Repeat $3n$ times:
      i. If F' is satisfied by the assignment a then return a and halt;
      ii. Pick any clause C in F' such that C is falsified by a. Choose a literal l in C uniformly at random. Modify a by flipping the value of l.
   (c) Return F'.

3.2 Description of $\mathcal{P}$

Suppose that an input formula F has a satisfyimg assignment A. The procedure first calls Subroutine $\mathcal{S}$. The result of $\mathcal{S}$ depends on which case of the dichotomy holds.

1. The input formula F has "many" ($\geq d$) bad clauses. Then $\mathcal{S}$ never finds A.

2. The input formula F has "few" ($< d$) bad clauses. Then $\mathcal{S}$ returns A (with some probability).

If $\mathcal{S}$ does not return a satisfying assignment, the procedure $\mathcal{P}$ simplifies F and recursively invokes itself on the simplified formula, i.e. $\mathcal{P}$ processes Case 1 of the dichotomy. To simplify the formula, $\mathcal{P}$ uses the fact that A falsifies at least one clause in F'. Note that such a clause has been obtained from a "long" clause (longer than k) in F because each "short" clause must contain a true literal. Therefore, $\mathcal{P}$ chooses a clause at random from those clauses in F' that are obtained from "long" clauses in F. Let $l_1, \ldots, l_k$ be all literals of the chosen clause. Then $\mathcal{P}$ assigns f to these literals and reduces F to $F[l_1 = \mathsf{f}, \ldots, l_k = \mathsf{f}]$.

Procedure $\boldsymbol{\mathcal{P}}$

Input: Formula F with m clauses over n variables, integers k and d.

Output: Satisfying assignment or "no".

1. Invoke $\mathcal{S}$ on F, k, and d.

2. Choose a clause at random from those clauses in F' that are obtained from "long" (longer than k) clauses of F. Let the chosen clause consist of literals $l_1, \ldots, l_k$.

3. Simplify F by assigning f to all $l_1, \ldots, l_k$, i.e. reduce F to $F[l_1 = \mathsf{f}, \ldots, l_k = \mathsf{f}]$.

4. Recursively invoke $\mathcal{P}$ on $F[l_1 = \mathsf{f}, \ldots, l_k = \mathsf{f}]$.

5. Return "no".

Note that Schuler's algorithm [14] is a special case of $\mathcal{P}$: take $d = 1$, $k = \lceil \log(2m) \rceil$, and use the PPSZ algorithm instead of Schöning's algorithm at step 2 in $\mathcal{S}$.

4. Success Probability of Procedure $\mathcal{P}$

Given an input formula F and some values of the parameters k and d, Procedure $\mathcal{P}$ finds a fixed satisfying assignment A or returns "no". What is the probability of finding A? In this section, we give a lower bound on the success probability of $\mathcal{P}$ and choose values of k and d that maximize this bound.

We define the minimum success probabilities for $\mathcal{S}$ and $\mathcal{P}$ as follows:

- $s(n,m,k,d)$ is the minimum success probability of $\mathcal{S}$ where the minimum is taken over all pairs F, A such that (i) A satisfies F; (ii) F has n variables and m clauses; (iii) F has less than d bad clauses.

- $p(n,m,k,d)$ is the minimum success probability of $\mathcal{P}$ where the minimum is taken over all pairs F, A such that (i) A satisfies F; (ii) F has n variables and m clauses.

In all next lemmas we assume that $m > d \geq 2$.

Lemma 1. Subroutine $\mathcal{S}$ has the following lower bound on the success probability:

$$s(n,m,k,d) \quad > \quad (emn)^{-(d-1)}\; 2^{-n\left(1-\frac{\log e}{k}\right)}$$

Proof. Let A be a fixed satisfying assignment to F. The probability of finding A is the product of three probabilities s_1, s_2, and s_3. The probability s_1 is the probability that the set $\{B_1, \ldots, B_{d-1}\}$ of chosen clauses contains all bad clauses (step 1a in $\mathcal{S}$). The probability s_2 is the probability of guessing sets with true literals for all $B_1, \ldots, B_{d-1}$ (step 1b in $\mathcal{S}$). The probability s_3 is the probability of finding A by one random walk (step 2 in $\mathcal{S}$).

Since $B_1, \ldots, B_{d-1}$ are chosen at random from m input clauses, we have (using Stirling's approximation as in [1, page 4]):

$$s_1 \;\geq\; \frac{1}{\binom{m}{d-1}} \;\geq\; \sqrt{2\pi m \left(\frac{d-1}{m}\right)\left(1-\frac{d-1}{m}\right)} \cdot 2^{-H\left(\frac{d-1}{m}\right)m} \;>\; \frac{3}{2} \cdot 2^{-H\left(\frac{d-1}{m}\right)m}$$

For further estimation we use the inequality $\log(1+x) < x \log e$:

$$\begin{aligned} s_1 &> \frac{3}{2} \cdot 2^{-H\left(\frac{d-1}{m}\right)m} \\ &= \frac{3}{2} \cdot 2^{-(d-1)\log\left(\frac{m}{d-1}\right)-(m-d+1)\log\left(1+\frac{d-1}{m-d+1}\right)} \\ &> \frac{3}{2} \cdot 2^{-(d-1)\log\left(\frac{m}{d-1}\right)-(m-d+1)\left(\frac{d-1}{m-d+1}\right)\log e} \\ &= \frac{3}{2}\left(\frac{em}{d-1}\right)^{-(d-1)} \end{aligned}$$

To estimate s_2, we estimate the probability of guessing a set with a true literal for each B_i. Obviously, B_i has at most $\binom{n}{k}$ subsets of literals of size k. For a fixed true literal in B_i, there are at most $\binom{n-1}{k-1}$ of such subsets that contain this literal. Hence

$$s_2 \;\geq\; \left(\frac{\binom{n-1}{k-1}}{\binom{n}{k}}\right)^{d-1} \;=\; \left(\frac{k}{n}\right)^{d-1}$$

A lower bound on s_3 is proved in [13]: $\frac{2}{3}\left(2-\frac{2}{k}\right)^{-n}$. Using the inequality $1-x \le 2^{-x\log e}$, we have

$$\begin{aligned} s_3 &\ge \tfrac{2}{3}\left(2-\tfrac{2}{k}\right)^{-n} &=& \tfrac{2}{3}\cdot 2^{-n}\left(1-\tfrac{1}{k}\right)^{-n} \\ &\ge \tfrac{2}{3}\cdot 2^{-n}\, 2^{\frac{n\log e}{k}} &=& \tfrac{2}{3}\cdot 2^{-n\left(1-\frac{\log e}{k}\right)} \end{aligned}$$

The product of the above bounds for s_1, s_2, and s_3 gives the following bound:

$$\begin{aligned} s_1\cdot s_2\cdot s_3 &> \tfrac{3}{2}\left(\tfrac{em}{d-1}\right)^{-(d-1)}\cdot\left(\tfrac{n}{k}\right)^{-(d-1)}\cdot(2/3)\; 2^{-n\left(1-\frac{\log e}{k}\right)} \\ &= \left(\tfrac{emn}{k(d-1)}\right)^{-(d-1)}\; 2^{-n\left(1-\frac{\log e}{k}\right)} \end{aligned}$$

Finally, we relax the above bound to $(emn)^{-(d-1)}\; 2^{-n\left(1-\frac{\log e}{k}\right)}$.

□

Our goal is to choose values of the parameters k and d (viewed as functions of n and m) so as to maximize the success probability of Procedure $\mathcal{P}$. We choose the functions $k_0 = k_0(n,m)$ and $d_0 = d_0(n,m)$ in two steps. First, we find a lower bound on the success probability of $\mathcal{P}$. The maximum of this bound is attained when k is a certain function of n, m, and d. Then, substituting this function in the bound, we find $d_0(n,m)$ that maximizes the bound.

Lemma 2. For all k such that

$$k \;\ge\; \frac{\log\left(\frac{em}{d}\right)}{1+\frac{(d-1)\log e}{n}} \tag{2}$$

Procedure $\mathcal{P}$ has the following lower bound on the success probability:

$$p(n,m,k,d) \;>\; (emn)^{-(d-1)}\; 2^{-n\left(1-\frac{\log e}{k}\right)}$$

Proof. Let A be a fixed satisfying assignment to F. Suppose that Procedure $\mathcal{P}$ finds A with t recursive calls, where t is some integer in the interval $0 \le t \le \lfloor n/k \rfloor$. Then $\mathcal{P}$ simplifies formulas t times. For each simplification, the probability of guessing a clause in F' that consists of false literals is at least d/m. Therefore, with probability at least $(d/m)^t$, we get a formula G such that

- G is still satisfied by A;
- G has less than d bad clauses;
- G has at most $n - kt$ variables.

Using Lemma 1, we get the following lower bound on the probability of finding A with t recursive calls:

$$\begin{aligned} p(n,m,k,d) &> \left(\tfrac{d}{m}\right)^t (em(n-kt))^{-(d-1)}\; 2^{-(n-kt)\left(1-\frac{\log e}{k}\right)} \\ &= \left(\tfrac{d}{m}\right)^t (emn)^{-(d-1)}\left(1-\tfrac{kt}{n}\right)^{-(d-1)} 2^{-n\left(1-\frac{\log e}{k}\right)+kt\left(1-\frac{\log e}{k}\right)} \\ &= \beta(n,m,k,d)\cdot\left(\tfrac{d}{m}\right)^t\left(1-\tfrac{kt}{n}\right)^{-(d-1)}\; 2^{kt\left(1-\frac{\log e}{k}\right)} \\ &= \beta(n,m,k,d)\cdot 2^{-t\,\log\left(\frac{m}{d}\right)\,-\,(d-1)\,\log\left(1-\frac{kt}{n}\right)\,+\,kt\left(1-\frac{\log e}{k}\right)} \end{aligned}$$

where β denotes the lower bound in the claim:

$$\beta(n,m,k,d) \quad = \quad (emn)^{-(d-1)}\, 2^{-n\left(1-\frac{\log e}{k}\right)}$$

Note that the β bound is the same as the bound given by Lemma 1. Since $\log(1-x) < -x\log e$ for all $0 \le x < 1$, we have

$$\begin{aligned} p(n,m,k,d) \quad &> \quad \beta(n,m,k,d)\cdot 2^{t\left(-\log(\frac{m}{d})+\frac{k(d-1)}{n}\log e + k - \log e\right)} \\ &= \quad \beta(n,m,k,d)\cdot 2^{t\left(k\left(1+\frac{(d-1)\log e}{n}\right)-\log(\frac{em}{d})\right)} \\ &\ge \quad \min_t \left[\beta(n,m,k,d)\cdot 2^{t\left(k\left(1+\frac{(d-1)\log e}{n}\right)-\log(\frac{em}{d})\right)}\right] \end{aligned}$$

where the minimum is taken over all t such that $0 \le t \le \lfloor n/k \rfloor$. The second inequality in (2) is equivalent to

$$k\left(1+\tfrac{(d-1)\log e}{n}\right) - \log\left(\tfrac{em}{d}\right) \quad \ge \quad 0.$$

Therefore, the minimum is attained when $t = 0$, which proves the claim.

$\square$

It follows from Lemma 2 that Procedure $\mathcal{P}$ has the following lower bound on the success probability:

$$p(n,m,k,d) \quad > \quad \min_t \left[\beta(n,m,k,d)\cdot 2^{t\left(k\left(1+\frac{(d-1)\log e}{n}\right)-\log(\frac{em}{d})\right)}\right] \tag{3}$$

What value of k maximizes this bound for given n, m, and d? A more thorough analysis shows that this bound is maximum when the inequality on k is replaced by the equality:

$$k \quad = \quad \frac{\log\left(\frac{em}{d}\right)}{1+\frac{(d-1)\log e}{n}}$$

Consider the bound obtained from (3) by substituting this value for k. Differentiation by d shows that this bound is maximum when d is close to $n/\log^3(em)$. However, k and d must be integers, so we take the ceilings of those values of k and d.

Lemma 3. For all n and m such that $10 \le n < em \le 2^{\sqrt[3]{n/2}}$, take the following values of k and d:

$$k_0 = \left\lceil \frac{\log\left(\frac{em}{n}\right)+3\log\log(em)}{1+\frac{\log e}{\log^3(em)}} \right\rceil \qquad d_0 = \left\lceil \frac{n}{\log^3(em)} \right\rceil \tag{4}$$

Then

$$p(n,m,k_0,d_0) \quad > \quad 2^{-n\left(1-\frac{\log 2}{k_0+2}\right)}$$

Proof. First, we show that Lemma 2 can be applied for k_0 and d_0, i.e. we show that (2) holds. It is straightforward to check the first inequality in (2). For the second inequality we have

$$\begin{aligned}
&k_0\left(1+\tfrac{(d_0-1)\log e}{n}\right)-\log\left(\tfrac{em}{d_0}\right) \;=\\
&k_0\left(1+\tfrac{(d_0-1)\log e}{n}\right)-\log(em)+\log d_0 \;\geq\\
&\log\left(\tfrac{em}{n}\right)+3\log\log(em)-\log(em)+\log\left(\tfrac{n}{\log^3(em)}\right) \;=\; 0.
\end{aligned}$$

Next, we substitute k_0 and d_0 in the bound given by Lemma 2. Note that $d_0 = n/\log^3(em)+\delta$ where $0 \leq \delta < 1$. Then we have (using $n < em$)

$$\begin{aligned}
p(n,m,k_0,d_0) &> (emn)^{-(d_0-1)}\, 2^{-n\left(1-\frac{\log e}{k_0}\right)}\\
&= (emn)^{-\delta+1}\,(emn)^{-\frac{n}{\log^3(em)}}\, 2^{-n\left(1-\frac{\log e}{k_0}\right)}\\
&> (emn)^{-\frac{n}{\log^3(em)}}\, 2^{-n\left(1-\frac{\log e}{k_0}\right)}\\
&= 2^{-\left(\frac{n}{\log^2(em)}\right)\left(\frac{\log(emn)}{\log(em)}\right)-n\left(1-\frac{\log e}{k_0}\right)}\\
&> 2^{-\left(\frac{n}{\log^2(em)}\right)\left(\frac{\log(em\cdot em)}{\log(em)}\right)-n\left(1-\frac{\log e}{k_0}\right)}\\
&= 2^{-n\left(\frac{2}{\log^2(em)}\right)-n\left(1-\frac{\log e}{k_0}\right)}
\end{aligned}$$

It remains to prove that

$$-\frac{2n}{\log^2(em)}-n+\frac{n\log e}{k_0} \;\geq\; -n\left(1-\frac{\log e}{k_0+2}\right)$$

This inequality is equivalent to

$$\frac{\log e}{k_0}-\frac{\log e}{k_0+2} \;\geq\; \frac{2}{\log^2(em)}$$

which, in turn, is equivalent to

$$\left(\frac{k_0}{\log(em)}\right)^2\left(1+\frac{2}{k_0}\right) \;\leq\; \log e \tag{5}$$

Our assumption $em \leq 2^{\sqrt[3]{n/2}}$ implies $k_0 < \log(em)$. Indeed,

$$\begin{aligned}
k_0 &< \log\left(\frac{em}{n}\right)+3\log\log(em)+1\\
&= \log(em)-\left[\log\left(\frac{n}{2}\right)-\log(\log^3(em))\right]\\
&\leq \log(em)
\end{aligned}$$

Therefore, the first factor in (5) is less than 1 under the assumption. To make sure that the second factor is less than $\log e \approx 1.44$, we notice that $k_0 \geq 5$. Indeed, if $n \geq 10$ then $k_0 > 3\log\log(em) > 3\log\log(10) > 5$.

□

5. Main Result

The main algorithm (Algorithm $\mathcal{A}$) is a repetition of Procedure $\mathcal{P}$. Lemma 3 shows that if we substitute k_0 and d_0 defined in (4) for the parameters of $\mathcal{P}$ then the success probability of $\mathcal{P}$ is at least

$$2^{-n\left(1-\frac{\log 2}{k_0+2}\right)}$$

The number r of repetitions is taken to be the inverse to this probability:

$$r = \left\lceil 2^{n\left(1-\frac{\log e}{k_0+2}\right)} \right\rceil \tag{6}$$

which gives a constant success probability of Algorithm $\mathcal{A}$.

Algorithm $\boldsymbol{\mathcal{A}}$

Input: Formula F in CNF with m clauses over n variables.

Output: Satisfying assignment or "no".

```
1. Compute $k_0$ and $d_0$ as in (4) and $r$ as in (6).
2. Repeat the following $r$ times:
    (a) Run $\mathcal{P}(F, k_0, d_0)$;
    (b) If a satisfying assignment is found, return it and halt.
3. Return "no".
```

Theorem 1. Algorithm $\mathcal{A}$ runs in time

$$O(n^2m)\ 2^{n\left(1-\frac{\log e}{k_0+2}\right)}$$

For any satisfiable input formula such that $10 \le n < em \le 2^{\sqrt[3]{n/2}}$, Algorithm $\mathcal{A}$ finds a satisfying assignment with probability greater than 1/2.

Proof. To prove the first claim, we need to estimate the running time of Procedure $\mathcal{P}$. The procedure recursively invokes itself at most $\lfloor n/k_0 \rfloor$ times. Each call scans the input formula and eliminates at most k_0 variables. Therefore, the procedure performs less than n scans. Algorithm $\mathcal{A}$ repeats the procedure at most r times, which gives the bound in the claim.

Let $p_{\mathcal{A}}(n, m)$ be the success probability of Algorithm $\mathcal{A}$. Then

$$p_{\mathcal{A}}(n, m) \ \ge\ 1 - (1 - p(n, m, k_0, d_0))^r$$

Using Lemma 3 and the inequality $(1 - x)^r \le e^{-xr}$, we have $p_{\mathcal{A}}(n, m) \ge 1 - e^{-1}$.

□

Remark 1. Note that we can write the bound in Theorem 1 as

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

Remark 2. For formulas with a constant clause density, i.e. if m is linear in n, our bound is better than in the general case:

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

Remark 3. Our algorithm is based on the clause-shortening approach proposed by Schuler [14]. His algorithm is derandomized with the same upper bound on the running time [5]. It would be natural to try to apply the same method of derandomization to our algorithm. However, the direct application gives a deterministic algorithm with a much worse upper bound. Is it possible to derandomize Algorithm $\mathcal{A}$ with the same or a slightly worse upper bound?

Acknowledgments

We thank Edward A. Hirsch and Daniel Rolf for helpful discussions. We also thank the anonymous reviewers for their help in improving the paper.

References

[1] B. Bollobás. *Random Graphs.* Cambridge University Press, 2nd edition, 2001.

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

[3] 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.

[4] 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.

[5] 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*, pages 69–75, May 2004.

[6] 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.

[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-Verlag, 1998.

[11] D. Rolf. Derandomization of PPSZ for Unique-k-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 216–225. Springer, June 2005.

[12] 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.

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

[14] 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.