

# An Improved Upper Bound for SAT

Evgeny Dantsin and Alexander Wolpert

Roosevelt University  
430 S. Michigan Av.  
Chicago, IL 60605, USA

{edantsin,awolpert}@roosevelt.edu

**Abstract.** We give a randomized algorithm for testing satisfiability of Boolean formulas in conjunctive normal form with no restriction on clause length. Its running time is at most  $2^{n(1-1/\alpha)}$  up to a polynomial factor, 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 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 (the satisfiability problem for Boolean formulas in conjunctive normal form with at most  $k$  literals per clause). These algorithms 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 [7, 6] and multistart random walk [10, 11]. These techniques give close upper bounds on the running time of solving  $k$ -SAT. The randomized algorithm in [10] has the  $(2 - 2/k)^n$  bound where  $n$  is the number of variables in the input formula; the randomized algorithm in [6] has a slightly better bound. The multistart-random-walk approach is derandomized using covering codes in [2], which gives the best known  $(2 - 2/(k + 1))^n$  bound for deterministic  $k$ -SAT algorithms. 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 [5] and by a deterministic algorithm with the  $1.473^n$  bound [1].

*Upper bounds for SAT (with no restriction on clause length).* The first nontrivial upper bound for SAT is given in [8]: the  $2^{n(1-1/2\sqrt{n})}$  bound for a randomized algorithm based on the satisfiability coding lemma. A close bound for a deterministic algorithm is proved in [3]. 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 [12] who gives a randomized algorithm that solves SAT using the

$k$ -SAT algorithm [7] as a subroutine. Schuler’s algorithm is derandomized in [4]. 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. Namely, we give a randomized algorithm that solves SAT with the following upper bound on the running time:

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

*Idea of the algorithm.* Our algorithm for SAT is basically 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  $p$ . A lower bound on  $p$  is given in Sections 4. 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. We divide each clause into *blocks* of length  $k$ . More exactly, for a given integer  $k \geq 1$ , a clause  $l_1, l_2, \dots, l_s$  is divided into  $b = \lceil s/k \rceil$  blocks as follows:

$$\boxed{l_1, \dots, l_k}, \quad \boxed{l_{k+1}, \dots, l_{2k}}, \quad \dots \quad \boxed{l_{k(b-1)+1}, \dots, l_s}$$

where each block (except the last one) consists of  $k$  literals. By the *first block* we mean the block consisting of  $l_1, l_2, \dots, l_k$ . We say that a block is *true* under an assignment  $A$  if at least one literal in the block is true under  $A$ ; otherwise we say that the block is *false* under  $A$ . If  $A$  is fixed, we omit the words “under  $A$ ”.

Let  $F$  consist of clauses  $C_1, \dots, C_m$ ; 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 block is false. We suppose that the number of such clauses is greater than or equal to some  $d$ . If we choose a clause  $C_i$  from  $C_1, \dots, C_m$  at random, the first block in  $C_i$  is false with probability at least  $d/m$ . Then we can simplify  $F$  by assigning “false” to all literals occurring in the first block of  $C_i$ .

**Case 2.** The input formula  $F$  has “few” clauses in which the first block is false. We suppose that the number of such clauses is less than  $d$ . Then we find  $A$  as follows. First, we guess those clauses in which the first block is false. Furthermore, we guess a true block in each such clause. Now we know a true block for each clause: it is either the first block or the block we have guessed. Let  $F'$  be the formula made up of these  $m$  true blocks. Obviously,  $F'$  is in  $k$ -CNF. Therefore we can use a  $k$ -SAT algorithm to find  $A$ .

This dichotomy suggests that  $\mathcal{P}$  is a recursive procedure that invokes a subroutine  $\mathcal{S}$  for processing Case 2. If the subroutine does not return a satisfying assignment,  $\mathcal{P}$  simplifies the input formula (Case 1) and recursively invokes itself on the simplified formula. Both  $\mathcal{P}$  and  $\mathcal{S}$  use  $k$  and  $d$  as parameters. These procedures are described in detail in Sect. 3.

Clearly, the success probability of  $\mathcal{P}$  depends on values of the parameters  $k$  and  $d$ . What values of  $k$  and  $d$  maximize the success probability? We answer

this question in Sect. 4. We show that if we take  $k \approx \log(m/n) + O(\log \log(m))$  and  $d \approx n/\log^3 m$  then we obtain the following lower bound on the success probability:

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

*Organization of the paper.* Sect. 2 gives basic definitions and notation. In Sect. 3 we describe the procedure  $\mathcal{P}$  and the subroutine  $\mathcal{S}$ . In Sect. 4 we give a lower bound on the success probability of  $\mathcal{P}$  and choose values of the parameters  $k$  and  $d$  that maximize this bound. Sect. 5 summarizes the main result: we define our algorithm for SAT and prove bound (1) on its running time.

## 2 Definitions and Notation

*Formulas and assignments.* 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 the complement literal, 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 the complement truth value. A *clause*  $C$  is a sequence of literals such that  $C$  contains no complement literals. A *formula*  $F$  is a set of clauses. If each clause in  $F$  contains at most  $k$  literals, we say that  $F$  is a *k-CNF formula*.

An *assignment* to variables  $x_1, \dots, x_n$  is a mapping from  $\{x_1, \dots, x_n\}$  to  $\{t, f\}$ . This mapping is extended to literals: each literal  $\neg x_i$  is mapped to the truth value complement to the 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  $l_1, \dots, l_s$  be literals such that their variables occur in  $F$ . We write  $F[l_1 = f, \dots, l_s = f]$  to denote the formula obtained from  $F$  by assigning the value f to all of  $l_1, \dots, l_s$ . This formula is obtained from  $F$  as follows: the clauses that contain any literal from  $\neg l_1, \dots, \neg l_s$  are deleted from  $F$ , and the literals  $l_1, \dots, l_s$  are deleted from the other clauses. Note that  $F[l_1 = f, \dots, l_s = f]$  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  $l$ .

*The SAT and k-SAT problems.* 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 consisting of at most  $k$  literals.

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

- $F$  denotes a formula;  $n$  and  $m$  denote, respectively, the number of variables and the number of clauses in  $F$ ;
- $A$  denotes a satisfying assignment to  $F$ ;
- $F[l_1=f, \dots, l_s=f]$  denotes the formula obtained from  $F$  by assigning  $f$  to all literals  $l_1, \dots, l_s$ ;
- $\log_2 x$  denotes  $\log_2 x$ ;
- $H(x)$  denotes the entropy function:  $H(x) = -x \log x - (1-x) \log(1-x)$ .

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

In this section we define the procedure  $\mathcal{P}$  and the subroutine  $\mathcal{S}$  outlined in Sect. 1. Both procedures use the parameters  $k$  and  $d$  (their values will be determined in the next section).

*Description of  $\mathcal{S}$ .* Suppose that an input formula  $F$  has a satisfying assignment such that  $F$  has only “few” ( $< d$ ) clauses in which the first block is false under this assignment. Then the subroutine  $\mathcal{S}$  finds such an assignment (with some 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  $F'$ .

At the first step,  $\mathcal{S}$  guesses all “bad” clauses for some satisfying assignment  $A$ , i.e. clauses in which the first block is false under  $A$ . More exactly, the subroutine guesses a (possibly) larger set of clauses: a set  $\{B_1, \dots, B_{d-1}\}$  such that all “bad” clauses are contained in this set. For each clause  $B_i$ , the subroutine guesses a true block in  $B_i$ . Thus, the subroutine gets a true block for each clause in  $F$  – the guessed true blocks for  $B_1, \dots, B_{d-1}$  and the first blocks for the other clauses in  $F$ . These true blocks make up  $F'$ . It is obvious that  $A$  satisfies  $F'$ .

To test satisfiability of  $k$ -CNF formulas at the second step,  $\mathcal{S}$  uses a randomized polynomial-time algorithm that finds a satisfying assignment with an exponentially small probability. We choose Schönning’s algorithm [10] to perform this testing (we could choose any algorithm that has at least the same success probability, for example the algorithm [6] based on the satisfiability coding lemma). More exactly, we use “one random walk” of Schönning’s algorithm, which has the success probability at least  $(2 - 2/k)^{-n}$  up to a constant [11].

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 blocks. Also note that the smaller  $d$ , the higher the probability of guessing a formula consisting of true blocks.

### Subroutine $\mathcal{S}$

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

**Output:** Satisfying assignment or “no”.

1. Reduce  $F$  to a  $k$ -CNF formula  $F'$  as follows:
  - (a) Choose  $d - 1$  clauses  $B_1, \dots, B_{d-1}$  in  $F$  at random.  
*Comment:* Guess a set that contains all “bad” clauses.
  - (b) For each  $B_i$ , choose a block in  $B_i$  at random and replace  $B_i$  by the chosen block.  
*Comment:* Guess a true block in each  $B_i$  and replace  $B_i$  by this true block.
  - (c) Replace each clause not belonging to  $\{B_1, \dots, B_{d-1}\}$  by its first block.  
*Comment:* The first block in each “good” clause is assumed to be true.
2. Test satisfiability of  $F'$  using one random walk of length  $3n$  (see [10] 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 stop;
    - 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 “no”.

*Description of  $\mathcal{P}$ .* The procedure first calls Subroutine  $\mathcal{S}$ . The result of  $\mathcal{S}$  depends on which case of the dichotomy holds.

1. For every satisfying assignment  $A$  (if any), the input formula  $F$  has “many” ( $\geq d$ ) clauses in which the first block is false under  $A$ . Then the subroutine never finds  $A$ .
2. For a satisfying assignment  $A$ , the input formula  $F$  has “few” ( $< d$ ) clauses in which the first block is false. Then the subroutine returns a satisfying assignment with its success probability.

The “no” answer from  $\mathcal{S}$  is treated as Case 1 of the dichotomy. Therefore, the procedure  $\mathcal{P}$  simplifies  $F$  and recursively invokes itself on the simplified formula. To simplify  $F$ , the procedure chooses a clause  $C$  at random from the “long” clauses in  $F$ , i.e. the clauses that have more than one block. Then  $\mathcal{P}$  assigns  $f$  to all literals in the first block of  $C$  and reduces  $F$  to  $F[l_1 = f, \dots, l_k = f]$ . Why can we restrict the choice to “long” clauses? Because if the input formula is satisfiable then all one-block clauses must be true.

### Procedure $\mathcal{P}$

**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  $d$ .  
*Comment:* If there are less than  $d$  “bad” clauses, the subroutine returns a satisfying assignment (with its success probability) and stops.

2. Choose clause  $C$  at random from those clauses in  $F$  that have more than one block.  
*Comment:* We guess a “long” clause in which the first block is false.
3. Simplify  $F$  by assigning  $f$  to all literals of the first block in  $C$ . Namely, if the first block of  $C$  consists of  $l_1, \dots, l_k$ , reduce  $F$  to  $F[l_1=f, \dots, l_k=f]$ .  
*Comment:* We simplify  $F$  by eliminating the guessed variables.
4. Recursively invoke  $\mathcal{P}$  on  $F[l_1=f, \dots, l_k=f]$ .
5. Return “no”.

Note that Schuler’s algorithm [12] can be viewed as a special case of Procedure  $\mathcal{P}$ . More exactly, we obtain Schuler’s algorithm from  $\mathcal{P}$  by taking  $d = 1$ ,  $k = \log(2m)$ , and using a different method for testing  $k$ -CNF formulas at step 2 in Subroutine  $\mathcal{S}$  (the algorithm based on the satisfiability coding lemma [7] instead of Schöning’s algorithm).

## 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.

Let  $s(F, k, d)$  be the success probability of Subroutine  $\mathcal{S}$ . More exactly, let  $s(F, k, d)$  be the probability that  $\mathcal{S}$  finds a satisfying assignment to a satisfiable input formula  $F$  such that  $F$  has less than  $d$  clauses in which the first block is false. Similarly, let  $p(F, k, d)$  be the success probability of Procedure  $\mathcal{P}$ , i.e. the probability that  $\mathcal{P}$  finds a satisfying assignment to a satisfiable formula  $F$ . We use the following notation:

- $s(n, m, k, d) = \min_F \{s(F, k, d)\}$ , where the minimum is taken over all satisfiable formulas  $F$  with  $n$  variables and  $m$  clauses such that  $F$  has less than  $d$  clauses in which the first block is false.
- $p(n, m, k, d) = \min_F \{p(F, k, d)\}$ , where the minimum is taken over all satisfiable formulas  $F$  with  $n$  variables and  $m$  clauses.

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

**Lemma 1.** *For any  $n, k$  and  $d$  such that  $k(d-1) \leq n$ , Subroutine  $\mathcal{S}$  has the following lower bound on the success probability:*

$$s(n, m, k, d) > \frac{\sqrt{m}}{3e} (emn)^{-(d-1)} 2^{-n(1 - \frac{\log e}{k})}$$

*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  $\{B_1, \dots, B_{d-1}\}$  of chosen clauses contains all “bad” clauses (step 1a in  $\mathcal{S}$ ). The probability  $s_2$  is the probability of guessing true blocks in all  $B_1, \dots, B_{d-1}$  (step 1b in  $\mathcal{S}$ ). The probability  $s_3$  is the probability of finding a satisfying assignment by one random walk (step 2 in  $\mathcal{S}$ ).

Since  $B_1, \dots, B_{d-1}$  are chosen at random from  $m$  input clauses, we have (using Stirling's approximation as in [9, exercise 9.42]):

$$s_1 \geq \frac{1}{\binom{m}{d-1}} > (\sqrt{m}/2) 2^{-H(\frac{d-1}{m}) m}$$

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

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

Each true block for a given  $B_i$  is chosen at random from at most  $\lceil n/k \rceil$  blocks, hence

$$s_2 \geq \left( \frac{1}{\lceil n/k \rceil} \right)^{d-1} > \left( \frac{1}{(n/k)+1} \right)^{d-1} = \left( \frac{n}{k} \right)^{-(d-1)} \left( 1 + \frac{k}{n} \right)^{-(d-1)}$$

Here we use the inequality  $1+x \leq e^x$  and our restriction on  $k$  and  $d$ :

$$s_2 > \left( \frac{n}{k} \right)^{-(d-1)} e^{-\frac{k}{n}(d-1)} \geq \left( \frac{n}{k} \right)^{-(d-1)} e^{-1}$$

A lower bound on  $s_3$  is proved in [11]:  $\frac{2}{3} \left( 2 - \frac{2}{k} \right)^{-n}$ . Using the inequality  $1-x \leq 2^{-x \log e}$ , we have

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

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

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

Finally, we relax the above bound to  $(\sqrt{m}/3e) (emn)^{-(d-1)} 2^{-n(1 - \frac{\log e}{k})}$ .  $\square$

Our goal is to choose values of the parameters  $k$  and  $d$  as functions of  $n$  and  $m$ . The values should be chosen so as to maximize the success probability of Procedure  $\mathcal{P}$ . We will choose the functions  $k_0 = k_0(n, m)$  and  $d_0 = d_0(n, m)$  in two steps. First, we will 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 will find  $d_0(n, m)$  that maximizes the bound.

**Lemma 2.** For any  $n, m, k$  and  $d$  such that

$$\frac{n}{d-1} \geq k \geq \frac{\log\left(\frac{em}{d}\right)}{1 + \frac{(d-1)\log e}{n}} \quad (2)$$

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

$$p(n, m, k, d) > \left(\frac{\sqrt{m}}{3e}\right) (emn)^{-(d-1)} 2^{-n\left(1 - \frac{\log e}{k}\right)}$$

*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  $0 \leq t \leq \lfloor n/k \rfloor$ . Then  $\mathcal{P}$  simplifies formulas  $t$  times. For each simplification, the probability of guessing a clause in which the first block is false is at least  $d/m$ . Therefore, with probability at least  $(d/m)^t$ , we get a formula  $G$  such that (i)  $G$  is still satisfied by  $A$ , (ii)  $G$  has less than  $d$  clauses in which the first clause is false; and (iii)  $G$  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:

$$\begin{aligned} p(n, m, k, d) &> \left(\frac{d}{m}\right)^t \left(\frac{\sqrt{m}}{3e}\right) (em(n-kt))^{-(d-1)} 2^{-(n-kt)\left(1 - \frac{\log e}{k}\right)} \\ &= \left(\frac{d}{m}\right)^t \left(\frac{\sqrt{m}}{3e}\right) (emn)^{-(d-1)} \left(1 - \frac{kt}{n}\right)^{-(d-1)} 2^{-n\left(1 - \frac{\log e}{k}\right) + kt\left(1 - \frac{\log e}{k}\right)} \\ &= \beta(n, m, k, d) \cdot \left(\frac{d}{m}\right)^t \left(1 - \frac{kt}{n}\right)^{-(d-1)} 2^{kt\left(1 - \frac{\log e}{k}\right)} \\ &= \beta(n, m, k, d) \cdot 2^{-t \log\left(\frac{m}{d}\right) - (d-1) \log\left(1 - \frac{kt}{n}\right) + kt\left(1 - \frac{\log e}{k}\right)} \end{aligned}$$

where  $\beta$  denotes the lower bound in the claim:

$$\beta(n, m, k, d) = \left(\frac{\sqrt{m}}{3e}\right) (emn)^{-(d-1)} 2^{-n\left(1 - \frac{\log e}{k}\right)}$$

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

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

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

$$k \left(1 + \frac{(d-1)\log e}{n}\right) - \log\left(\frac{em}{d}\right) \geq 0.$$

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:

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

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:

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

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 any  $n$  and  $m$  such that  $10 \leq n < em \leq 2\sqrt[3]{n/2}$ , take the following values of  $k$  and  $d$ :

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

$$d_0 = \left\lceil \frac{n}{\log^3(em)} \right\rceil$$

Then

$$p(n, m, k_0, d_0) > \left(\frac{\sqrt{m}}{3e}\right) 2^{-n\left(1 - \frac{\log 2}{k_0 + 2}\right)}$$

*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

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

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  $n < em$ )

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

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

$$\left(\frac{k_0}{\log(em)}\right)^2 \left(1 + \frac{2}{k_0}\right) \leq \log e \quad (4)$$

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 (4) 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  $k_0 > 3 \log \log(em) > 3 \log \log(10) > 5$ .  $\square$

## 5 Main Result

This section summarizes the contents of Sect(s). 3 and 4. We define the main algorithm (as a repetition of the procedure  $\mathcal{P}$ ) and we give an upper bound on its running time (using the lower bound on the success probability of  $\mathcal{P}$ ).

### Algorithm $\mathcal{A}$

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

**Output:** Satisfying assignment or “no”.

1. Compute  $k_0$ ,  $d_0$ , and  $r$  as follows:

$$\begin{aligned} k_0 &= \left\lceil \frac{\log\left(\frac{em}{n}\right) + 3 \log \log(em)}{1 + \frac{\log e}{\log^3(em)}} \right\rceil \\ d_0 &= \left\lceil \frac{n}{\log^3(em)} \right\rceil \\ r &= \left\lceil \left(\frac{3en}{\sqrt{m}}\right) 2^{n\left(1 - \frac{\log 2}{k_0 + 2}\right)} \right\rceil \end{aligned}$$

2. Repeat the following  $r$  times:
  - (a) Run  $\mathcal{P}(n, m, k_0, d_0)$ ;
  - (b) If a satisfying assignment is found, return it and stop.
3. Return “no”.

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

$$O(n^3 m) 2^{n\left(1 - \frac{\log e}{k_0 + 2}\right)}$$

*For any satisfiable input formula such that  $10 \leq n < em \leq 2^{\sqrt[3]{n/2}}$ , 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  $O(n)$  scans. Algorithm  $\mathcal{A}$  repeats the procedure at most  $r$  times, which gives the bound in the claim. Note that we can write this bound as

$$O(n^3 m) 2^{n\left(1 - \frac{1}{\ln\left(\frac{m}{n}\right) + O(\ln \ln m)}\right)}$$

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

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

The second claim follows from Lemma 3 and the inequality  $(1 - x)^r \leq e^{-xr}$ .  $\square$

*Remark 1.* 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 2.* Our algorithm can be viewed as a generalization of Schuler's algorithm [12]. The latter is derandomized with the same upper bound on the running time [4]. 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?

## 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, October 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'04*, 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'04*, pages 69–75, May 2004.
5. K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'04*, page 328, January 2004. A preliminary version appeared in Electronic Colloquium on Computational Complexity, Report No. 53, July 2003.
6. 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.
7. 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.
8. 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.
9. O. Patashnik R. Graham, D. Knuth. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 2nd edition, 1994.
10. 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.
11. U. Schöning. A probabilistic algorithm for  $k$ -SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.
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.