Electronic Colloquium on Computational Complexity, Report No. 18 (2011)

# A Kolmogorov Complexity Proof of the Lovász Local Lemma for Satisfiability *

Jochen Messner Thomas Thierauf

Aalen University
Dep. Elektronik und Informatik
73430 Aalen
{jochen.messner,thomas.thierauf}@htw-aalen.de

**Abstract**

Recently, Moser and Tardos [MT10] came up with a *constructive* proof of the Lovász Local Lemma. In this paper, we give another constructive proof of the lemma, based on Kolmogorov complexity. Actually, we even improve the Local Lemma slightly.

## 1 Introduction

The Lovász Local Lemma applied to the satisfiability problem states that a $k$-CNF formula is satisfiable if each clause has common variables with at most $2^k/e-1$ other clauses. The original proof of the Local Lemma was *non-constructive*: it didn't give a hint on how to efficiently compute a satisfying assignment of the given formula.

Starting with a paper by Beck [Bec91] there appeared a series of papers [Alo91, Sri08, Mos08] that came up with *constructive* proofs of the lemma, with stronger bounds on the clause neighborhood however. Then Moser [Mos09] made a big step ahead and came up with a randomized algorithm that finds a satisfying assignment if the neighborhood of each clause is bounded by $2^k/8$. In his conference talk on [Mos09], Moser presented an ingeniously simple argument based on Kolmogorov complexity (see [For09]). Thereafter, Moser and Tardos [MT10] improved this to the bound claimed by the Local Lemma, $2^k/e - 1$, with a different proof however that uses involved probabilistic arguments.

*Research supported by DFG grant TH 472/4-1



ISSN 1433-8092

The main contribution of this paper is to provide again a very elegant constructive proof for the Local Lemma via Kolmogorov complexity (Section 4). Actually, for certain values of $k$ we even improve the Local Lemma slightly and show in Section 5 a bound somewhat better than $(2^k - 1)/e$. Our method also applies to the more general *conflicting neighborhood* setting, also called the *lopsided version*, with the same bounds.

In the next section we give a more detailed introduction to the Local Lemma for satisfyability. In Section 3 we introduce the concepts and tools we use, like Kolmogorov complexity, binary entropy bounds on binomial coefficients, and the number of $d$-ary trees.

## 2 The Lovász Local Lemma

Throughout this paper, $\varphi$ is a $k$-CNF formula with $n$ variables and $m$ clauses, where every clause has exactly $k \geq 2$ variables. The *variables of a clause* are the variables that occur in a clause, positive or negative. A *literal* is a variable or the negation of a variable. We define a *dependence relation* or *neighborhood relation* $\Gamma = \Gamma_\varphi$ on the clauses of $\varphi$ as follows: for clauses $C \neq D$,

$$(C, D) \in \Gamma \quad \text{if } C \text{ and } D \text{ have a common variable.}$$

For satisfiability, it often suffices to consider the *conflicting neighborhood relation* or *lopsided relation* $\Gamma' = \Gamma'_\varphi$,

$$(C, D) \in \Gamma' \quad \text{if } C \text{ has a literal that occurs negated in } D.$$

We also write $D \in \Gamma(C)$ and $D \in \Gamma'(C)$ instead of $(C, D) \in \Gamma$ and $(C, D) \in \Gamma'$, respectively.

The *neighborhood graphs* $G_\varphi$ and $G'_\varphi$ are given by the $m$ clauses as nodes and $\Gamma$ and $\Gamma'$ as edges, respectively. Crucial parameters in the following are the maximum degrees $d_\varphi = \max_C |\Gamma(C)|$ and $d'_\varphi = \max_C |\Gamma'(C)|$. Clearly we have $\Gamma' \subseteq \Gamma$, and therefore $d'_\varphi \leq d_\varphi$.

The Lovász Local Lemma shows that if $\Gamma$ (resp. $\Gamma'$) is not too dense then $\varphi$ is satisfiable. In its *symmetric* form the formulation is given by an upper bound on the maximal degree $d_\varphi$, respectively $d'_\varphi$. Moreover the symmetric form only applies to $k$-CNF formulas $\varphi$ where every clause $C$ has exactly $k \geq 2$ variables. Therefore, when we make a random assignment to the variables, every clause has the same probability of being not satisfied, namely $2^{-k}$. We will consider only the symmetric version of the Lemma in the following (see also [GMSW09] for an overview on the symmetric version).

The Local Lemma was introduced by Erdős and Lovász [EL75]. The generalization to the conflicting neighborhood setting, the *lopsided version*, is due to Erdős and Spencer [ES91]. The version presented here uses the improved bounds from [ASE92] for $d_\varphi$ and from [McD97] for $d'_\varphi$.

**Theorem 1 (Symmetric Local Lemma)** *Let $\varphi$ be a $k$-CNF formula. Then $\varphi$ is satisfiable if*

$$d'_\varphi \leq \frac{2^k}{e} - 1\,.$$

Here $e$ denote the Euler number.

Moser and Tardos [MT10] got a constructive proof of Theorem 1, i.e. they showed that one can efficiently computed a satisfying assignmend for $\varphi$ provided that $\varphi$ fulfills the conditions in the theorem. Their algorithm to compute a satisfying assignment of a $k$-CNF-formula $\varphi$ is randomized and goes as follows.

SEARCH($\varphi$)
Pick a random assignment for the variables of $\varphi$
**while** there is a clause in $\varphi$ that is not satisfied **do**
. Choose an unsatisfied clause $C$
. Reassign the variables in $C$ independently at random
**output** the satisfying assignment

We can use any deterministic way to choose an unsatisfied clause in the while-loop, for example the smallest one according to some fixed ordering of the clauses.

Clearly, there is a chance that SEARCH($\varphi$) runs forever. However if $d'_\varphi \leq 2^k/e - 1$, the expected running time is linear in $m$, and when it halts, it has computed a satisfying assignment [MT10]. A very good exposition of the result is from Spencer [Spe10]. He considers the non-lopsided version and in fact, gives a slightly better bound on $d = d_\varphi$ to guarantee satisfiability, namely

$$\frac{(d+1)^{d+1}}{d^d} \leq 2^k. \tag{1}$$

Note that

$$\frac{(d+1)^{d+1}}{d^d} = (d+1)\left(1 + \frac{1}{d}\right)^d < (d+1)e\,.$$

Therefore $d \leq \frac{2^k}{e} - 1$ implies inequality (1).

In Section 4 we show how to obtain Theorem 1, i.e. the lopsided version, but with the bound (1). In Section 5 we improve the bound and show for $d = d'_\varphi$ that algorithm SEARCH will still find a satisfying assignment for $\varphi$ if

$$\frac{d^d}{(d-1)^{d-1}} \leq 2^k - 1. \tag{2}$$

A similar calculation as above shows that inequality (2) holds if $d \leq \frac{2^k-1}{e}$.

**Theorem 2** *Let $\varphi$ be a $k$-CNF formula. If*

$$d'_\varphi \leq \frac{2^k - 1}{e} \tag{3}$$

*then $\varphi$ is satisfiable, and* SEARCH *finds a satisfying assignment for $\varphi$ in expected time $O(m)$.*

Let us mention a very weak bound shown by Gebauer *et al.* [GMSW09] with a non-constructive argument: $\varphi$ is satisfiable if

$$d'_\varphi \leq \lceil 3(k-1)/2 \rceil . \tag{4}$$

However, this bound is pretty good for small values of $k$. To give an impression for these bounds, the table below shows the maximal integral values for $d'_\varphi$ that we get from the above bounds for $k = 2, \ldots, 10$.

| $k$ | $\lfloor 2^k/e \rfloor - 1$ | Eq. (1) | $\lfloor \frac{2^k-1}{e} \rfloor$ | Eq. (2) | $\lceil 3(k-1)/2 \rceil$ |
|---|---|---|---|---|---|
| 2 | 0 | 1 | 1 | 1 | 2 |
| 3 | 1 | 2 | 2 | 3 | 3 |
| 4 | 4 | 5 | 5 | 6 | 5 |
| 5 | 10 | 11 | 11 | 11 | 6 |
| 6 | 22 | 23 | 23 | 23 | 8 |
| 7 | 46 | 46 | 46 | 47 | 9 |
| 8 | 93 | 93 | 93 | 94 | 11 |
| 9 | 187 | 187 | 187 | 188 | 12 |
| 10 | 375 | 376 | 376 | 376 | 14 |

For $k = 2$, bound (4) is better than ours and it is not known whether SEARCH efficiently finds satisfying assignments for $\varphi$ in 2-CNF with $d_\varphi = 2$. But clearly, this case is not our main focus because 2-SAT is efficiently solvable anyway.

Already for $k = 3$ we achieve the same bound as in (4), but now with a constructive argument. That is, we show that SEARCH efficiently finds satisfying assignments in this case.

Let us also note that the new bound $\lfloor \frac{2^k-1}{e} \rfloor$ is larger by one than the bound $\lfloor 2^k/e \rfloor - 1$ for infinitely many $k$. To see this, observe that $\lfloor 2^k/e + (1 - 1/e) \rfloor \geq \lfloor 2^k/e \rfloor + 1$ if a 1 appears in the binary expansion of $1/e$ at position $-(k+1)$.

Is further improvement possible? Trivially, the formula $\varphi$ in $k$-CNF with $k$ variables that contains all possible $2^k$ clauses is unsatisfiable and we have $d_\varphi = d'_\varphi = 2^k - 1$ in this case. Therefore the bound $d'_\varphi \leq 2$ is optimal for $k = 2$. For $k \geq 3$ the optimal bounds are not known. In [GST11] it is shown that for some constant $c$ there are unsatisfiable $k$-CNF formulas $\varphi$ with $d_\varphi \leq 2^k(\frac{1}{e} + \frac{c}{\sqrt{k}})$. This shows that the factor $1/e$ in inequality (3) cannot be replaced by a larger constant factor. However there is still room for additive improvements.

## 3 Preliminaries

**Kolmogorov Complexity** For an algorithm $A$ that computes a function $A : \{0,1\}^* \to \{0,1\}^*$ and a fixed pairing function $\langle \cdot, \cdot \rangle$ let

$$K_A(x \mid y) = \min(\{|w| \mid A(\langle y, w \rangle) = x\} \cup \{\infty\}) .$$

Since there are at most $2^l - 1$ strings $w \in \{0,1\}^*$ of length less than $l$, we have for any $l \in \mathbb{R}$, all $y \in \{0,1\}^*$, and any set $S \subseteq \{0,1\}^*$

$$|\{x \in S \mid K_A(x \mid y) \geq l\}| \geq |S| - 2^l + 1 .$$

This implies that for any $c \in \mathbb{R}$

$$\begin{aligned} |\{x \in S \mid K_A(x \mid y) \geq \log |S| - c\}| &\geq |S| - 2^{\log |S| - c} + 1 \\ &> (1 - 2^{-c})|S| . \end{aligned} \tag{5}$$

In particular, with $c = 0$ and $S = \{0,1\}^s$ we have some $x \in \{0,1\}^s$ with $K_A(x \mid y) \geq |x|$ .

It is known that there are (universal) algorithms $U$ such that for every $A$ there is a constant $c_A$ such that

$$K_U(x \mid y) \leq K_A(x \mid y) + c_A$$

We fix such an $U$ and denote $K_U(x \mid y)$ by $K(x \mid y)$. Let us briefly write $K(x \mid y, z)$ instead of $K(x \mid \langle y, z \rangle)$. See e.g. [LV93, Cal02] for more details on Kolmogorov complexity.

**Binomials and Entropy** By $h$ we denote the the binary entropy, $h(p) = -p\log p - (1-p)\log(1-p)$ for $0 < p < 1$. We use following well known upper bound for $\binom{ds}{s}$ (see e.g. [Sch01]). For $d \geq 2$, and $s \geq 1$

$$\binom{ds}{s} < 2^{dh(1/d)s}. \tag{6}$$

**The number of $d$-ary trees** An important tool in our argument below is an estimate on the number of $d$-ary trees with $s$ nodes, denoted by $T_d(s)$. Well known is the case of binary trees, i.e. when $d = 2$, where $T_2(s)$ are the Catalan numbers,

$$T_2(s) = \frac{1}{s+1}\binom{2s}{s}.$$

In general, we have $T_d(0) = 1$, and for $s > 0$ the numbers $T_d(s)$ obey the recursion

$$T_d(s) = \sum_{s_1+s_2+\ldots+s_d=s-1} T_d(s_1)T_d(s_2)\cdots T_d(s_d),$$

because a $d$-ary tree has one node as root and the remaining $s-1$ nodes are split into $d$ subtrees of the root, where each subtree is again a $d$-ary tree.

In closed form $T_d(s)$ is given by the Fuss-Catalan numbers (see [GKP94])

$$T_d(s) = \frac{1}{(d-1)s+1}\binom{ds}{s}.$$

Actually we consider not only trees, but forests that consists of up to $m$ $d$-ary trees. Let us call this a $(d,m)$-*forest*. The number $F_{d,m}(s)$ of such forests with $s$ nodes is given by (see [GKP94])

$$F_{d,m}(s) = \frac{m}{ds+m}\binom{ds+m}{s}.$$

Using inequality (6) we have for $d \geq 2$ and $s, m \geq 1$

$$\begin{aligned} F_{d,m}(s) &< 2^{(ds+m)h(\frac{1}{d+m/s})} \\ &< 2^{(ds+m)h(\frac{1}{d})}. \end{aligned} \tag{7}$$

# 4 The Kolmogorov Argument

The core of the Kolmogorov argument is to reconstruct the random bits used by SEARCH from an input as small as possible. The size of this input will then give a bound on the running time of SEARCH.

### 4.1 The LOG

The sequence of clauses that is chosen by algorithm SEARCH during the execution of the while-loop is called the *LOG* [MT10]. Clearly, the LOG depends on the random choices in each iteration. Also, the LOG could be an infinite sequence in the case that SEARCH doesn't halt. Therefore we cut off the LOG after a finite number of iterations: let $s \geq 1$ be some integer and assume that SEARCH makes $\geq s$ iterations, then the LOG up to this point is a sequence of clauses $(C_1, \ldots, C_s)$. Note that SEARCH consumes exactly $n + ks$ random bits up to this point: $n$ for the initial assignment and $k$ for each iteration.

We start by showing that we can reconstruct the random bits used by SEARCH from the LOG $(C_1, \ldots, C_s)$ and the assignment of the $n$ variables after iteration $s$.

**Lemma 3** *Given the* LOG $(C_1, \ldots, C_s)$ *and the assignment* $\alpha$ *after iteration* $s$*, we can reconstruct the random bits used by* SEARCH.

*Proof.* A variable $x_i$ gets reassigned by SEARCH (possibly with the same value) each time it occurs in some clause $C_j$ of the LOG. We compute $s_i$, the number of clauses in the LOG that contain $x_i$. Hence $x_i$ gets reassigned $s_i$ times. Because in every iteration precisely $k$ variables get reassigned, we have $s = \sum_{i=1}^{n} s_i/k$.

Let $\alpha_i^{(j)} \in \{0,1\}$ denote the $j$-th assignment given to $x_i$ for $1 \leq j \leq s_i$. Hence $\alpha_i^{(1)}$ is the initial assignment to $x_i$, and $\alpha_i^{(s_i)}$ is the assignment to $x_i$ after iteration $s$. That is, $(\alpha_1^{(s_1)}, \ldots, \alpha_n^{(s_n)}) = \alpha$.

We show how to obtain the random bits by going the LOG backwards, starting with $C_s$. Since SEARCH chooses $C_s$ in iteration $s$, clause $C_s$ is violated before. Moreover the assignment after iteration $s-1$ differs only in the values assigned to the variables in $C_s$. Therefore the bits $\alpha_i^{(s_i)}$ from the variables that occur in $C_s$ are the $k$ random bits used by SEARCH in the last iteration, and we obtain $\alpha_i^{(s_i-1)}$ as

$$\alpha_i^{(s_i-1)} = \begin{cases} 0, & \text{if } x_i \text{ in } C_s, \\ 1, & \text{if } \overline{x}_i \text{ in } C_s. \end{cases}$$

Now we repeat this recursively on the LOG $(C_1, \ldots, C_{s-1})$ with $s_i - 1$ instead of $s_i$ for the variables $x_i$ that occur in $C_s$, until we arrive in the first iteration where we obtain the initial assignment picked by SEARCH. ∎

If we fix an ordering of the $m$ clauses, each clause is determined by its rank $i \in [m]$ in this ordering. Hence we can code the LOG (and by Lemma 3 the random bits) with $n + s \log m$ bits, given $\varphi$. However, this encoding is too long for our purpose.

A crucial observation is that we don't need the precise LOG: we call two clauses $C$ and $D$ *independent* if they have no opposite literals in common i.e. $(C, D) \notin \Gamma'_\varphi$. We claim that when we permute the LOG by sequentially commutating independent pairs of neighboring clauses, we still can reconstruct the random bits used by SEARCH.

**Lemma 4** *Given a sequence of clauses $(D_1, \ldots, D_s)$ obtained by several commutations of independent clauses, starting with the* LOG*, and the assignment $\alpha$ after iteration $s$, we can reconstruct the random bits used by* SEARCH.

*Proof.* Consider the LOG and let us commutate two independent clauses $C_j$ and $C_{j+1}$, i.e. we consider the commutated LOG $(C_1, \ldots, C_{j-2}, C_j, C_{j-1}, C_{j+1}, \ldots, C_s)$. Then the reconstruction algorithm presented in the proof of Lemma 3 still produces the same values $\alpha_i^{(j)}$: this is obvious for the variables that occur in only one of two clauses. If $C_j$ and $C_{j+1}$ have variable $x_i$ in common, then we have

$$\alpha_i^{(l-1)} = \alpha_i^{(l-2)} = \begin{cases} 0, & \text{if } x_i \text{ in } C_j \text{ and } C_{j+1}, \\ 1, & \text{if } \overline{x}_i \text{ in } C_j \text{ and } C_{j+1}, \end{cases}$$

where $l$ is the number of times the variable $x_i$ appears in $C_1, \ldots, C_j$, i.e. $\alpha_i^{(l)}$ is the assignment to $x_i$ after iteration $j$. Note that we don't have the case $x_i$ in $C_j$ and $\overline{x}_i$ in $C_{j+1}$ or vice versa, since $(C_j, C_{j+1}) \notin \Gamma'_\varphi$.

Inductively, we can commutate arbitrarily many independent pairs of clauses and still get the same values $\alpha_i^{(j)}$ for $1 \leq i \leq n$, $1 \leq j \leq s_i$.

To reconstruct the random bit string used by SEARCH we start SEARCH using the values $\alpha_i^{(j)}$ for the successive assignments to the variables. Recall that the choice of the next clause in SEARCH is deterministic. This yields the assignment to the clauses and in turn the random bits, and also the original LOG produced by the algorithm. ∎

### 4.2 Witness forests

From the LOG we will next define a *witness forest* (cf. the witness trees in [MT10]). Our final algorithm to reconstruct the random bits used by

SEARCH will have a coding of these witness forests as input. From the witness forest we will not directly get the LOG of SEARCH, but a permutation of the LOG as described in Lemma 4. As we have just seen in the lemma, this suffices for our purpose.

We construct the witness forest of a LOG $(C_1, \ldots, C_s)$ iteratively by inserting $C_j$ for $j = 1, \ldots, n$ into the initially empty forest. To insert $C_j$ we proceed as follows:

($i$) If there is a node in the forest labeled with a clause $D \in \{C_j\} \cup \Gamma'_\varphi(C_j)$, then select a node in the forest with such a label that is at the lowest level in the forest and append to it a node with label $C_j$ as child.

($ii$) Otherwise create a new tree in the forest that consists of a node as root with label $C_j$.

The witness forest has the following important properties:

1. It is a $(d+1, m)$-forest for $d = d'_\varphi$: ($i$) there are at most $m$ roots, one for each clause of $\varphi$, and ($ii$) each node has at most $d+1$ children. The children of a node with label $C_i$ have labels from $\{C_i\} \cup \Gamma'_\varphi(C_i)$. Recall that $d'_\varphi = \max_j |\Gamma'_\varphi(C_j)|$.

2. If $C_j \in \{C_i\} \cup \Gamma'_\varphi(C_i)$ and $i < j$ then the node added for $C_j$ is lower in the tree than the node added for $C_i$.

3. If $C_i$ and $C_j$ are in the same depth of the forest, then they are independent.

Therefore, if we output the labels of the nodes by increasing depth in any order, we obtain a a sequence $(D_1, \ldots, D_s)$ of clauses that can be obtained from the LOG by commutating independent clauses.

The next observation is crucial: we don't need to store the labels of the nodes, because we can compute the labels from the precise structure of the forest! Namely, we can use the order of the clauses as they appear in $\varphi$ as an order on the clauses. This induces an order on all the sets $\Gamma'_\varphi(C_i)$. Hence, in the witness forest, we can think of every node as having $d+1$ slots reserved in a certain order, one for each child to come. That is, we can for example distinguish the tree with a node with label, say, $C_1$ with a child with label $C_2$ from the tree where the child of $C_1$ has label $C_3$. Similar, for the potential roots of the trees, we have $m$ slots reserved. When $C_i$ becomes the root of a tree, we put it at slot $j$, if $j$ is the rank of $C_i$ in the order of all clauses. Therefore, if we know the precise structure of the forest, we can reconstruct the labels of the nodes.

Since we can enumerate all $(d+1,m)$-witness forests with $s$ nodes we can encode a witness forest by its index in the enumeration which, by inequality (7), needs $\lceil \log F_{d+1,m}(s) \rceil < \lceil ((d+1)s+m)h(1/(d+1)) \rceil$ bits.

**Lemma 5** *Given $s$ and $\varphi$, and the index of a $(d+1,m)$-witness forest representing the* LOG *and the assignment after iteration $s$ we can reconstruct the random bits used by* SEARCH.

### 4.3 Putting things together

Now we have all the tools for the Kolmogorov complexity argument.

**Theorem 6** *Let $\varphi$ be a formula in $k$-CNF and $d = d'_\varphi$. If*

$$\frac{(d+1)^{d+1}}{d^d} \leq 2^k$$

*then* SEARCH *finds a satisfying assignment for $\varphi$ in expected time $O(m)$.*

*Proof.* Assume that SEARCH makes $\geq s$ iterations of the while-loop. Fix a Kolmogorov random string $w \in \{0,1\}^{n+ks}$ with

$$K(w \mid \varphi, s) \geq n + ks - c \tag{8}$$

for some constant $c > 0$, and let SEARCH use the bits of $w$ as its random bits. That is, the first $n$ bits are used for the initial assignment and $k$ bits each are used to replace the assignment of the clause variables in each iteration of the while-loop.

By Lemma 5 we can construct $w$ from the assignment after iteration $s$ and an index of the witness forest. Hence, for some constant $c_A > 0$,

$$\begin{aligned} K(w \mid \varphi, s) &\leq n + \lceil \log F_{d+1,m}(s) \rceil + c_A \\ &< n + ((d+1)s+m)\, h\left(\frac{1}{d+1}\right) + 1 + c_A\,. \end{aligned}$$

Combined with inequality (8) this implies

$$\left(k - (d+1)\, h\left(\frac{1}{d+1}\right)\right) s < m\, h\left(\frac{1}{d+1}\right) + c + c_A + 1\,.$$

An easy calculation shows that the assumption $\frac{(d+1)^{d+1}}{d^d} \leq 2^k$ is equivalent to $0 < k - (d+1)h\left(\frac{1}{d+1}\right)$ for integral $d \geq 2$. Therefore we get

$$s < \frac{mh\left(\frac{1}{d+1}\right) + c + c_A + 1}{k - (d+1)h\left(\frac{1}{d+1}\right)} = O(m). \tag{9}$$

This shows that the algorithm halts after this number of iterations and then outputs a satisfying assignment. Using inequality (5) it follows that (8) (and therefore (9)) holds for a fraction of $1 - 2^{-c}$ of the possible strings $w$.

Now assume that $w$ is choosen at random by the algorithm and let $S$ be the random variable denoting the number of iterations of SEARCH. We have just shown that for any $c \geq 0$

$$\Pr\left[S < \frac{m\,h\left(\frac{1}{d+1}\right) + c + c_A + 1}{k - (d+1)\,h\left(\frac{1}{d+1}\right)}\right] \geq 1 - 2^{-c}. \tag{10}$$

Hence, at this point we can already conclude that SEARCH finds a satisfying assignment with high probability in $O(m)$ steps. We show next that this is also the expected time, i.e. $\mathrm{E}[S] = O(m)$.

Let $K = k - (d+1)h(1/(d+1))$ be the denominator in the fraction in equation (10) and define $s_0 = \frac{mh(1/(d+1)) + c_A + 1}{K} = O(m)$.

$$\begin{aligned}
\mathrm{E}[S] &= \sum_{s \geq 0} \Pr[S \geq s] \\
&= \sum_{0 \leq s < s_0} \Pr[S \geq s] + \sum_{s \geq s_0} \Pr[S \geq s] \\
&\leq s_0 + \sum_{s \geq 0} \Pr[S \geq s_0 + s] \\
&= s_0 + \sum_{s \geq 0} (1 - \Pr[S < s_0 + s])\,.
\end{aligned}$$

For $s \geq 0$ define $c = sK$. Then $s = \frac{c}{K}$ and inequality (10) becomes

$$\Pr[S < s_0 + s] \geq 1 - 2^{-c} = 1 - 2^{-sK}\,.$$

Therefore (since $2^K > 1$)

$$\sum_{s \geq 0} (1 - \Pr[S < s_0 + s]) \leq \sum_{s \geq 0} 2^{-sK} = \frac{1}{1 - 2^{-K}} = O(1)\,,$$

and hence $\mathrm{E}[S] \leq s_0 + \frac{1}{1-2^{-K}} = O(m)$. ∎

## 5 An Improvement

In the previous section we said that the trees in the witness forest are $(d+1)$-ary trees for $d = d'_\varphi$, because the children of a node with label $C$ can be from

$\{C\} \cup \Gamma'_\varphi(C)$, a set of size $\leq d+1$. However, by the construction of the trees, no node will actually have $d+1$ children: if a node with label $C$ has a child with label $C$ this is its only child, because the labels of the other children would be dependent on $C$. Moreover, we can easily avoid that a node with label $C$ has a child with label $C$: this happens only when SEARCH picks as the next random assignment for $C$ the same assignment as it had before. We modify SEARCH such that the choosen clauses will get satisfied.

MODIFIED-SEARCH($\varphi$)
Pick a random assignment for the variables of $\varphi$
**while** there is a clause in $\varphi$ that is not satisfied **do**
. Choose an unsatisfied clause $C$
. Choose a random $i \in \{1, \ldots, 2^k - 1\}$
. Assign variables in $C$ by the $i$-th satisfying assignment for $C$
**output** the satisfying assignment

Note that SEARCH and MODIFIED-SEARCH are essentially identical algorithms: if SEARCH chooses an unsatisfied clause $C$ and accidentally reassigns the variables such that $C$ stays unsatisfied, it will choose $C$ again in the next iteration, because the selection method is deterministic. This will be repeated until the reassignment satisfies $C$. The expected number of iterations until this happens is $2^k/(2^k - 1) \leq 2$. Then both algorithm proceed again the same way.

**Lemma 7** *Let $(C_1, \ldots, C_s)$ be the* LOG *of a run of* MODIFIED-SEARCH *over $s$ iterations. Then in the witness forest constructed from $(C_1, \ldots, C_s)$ no node will have a child with the same label.*

*Proof.* If a clause $C$ is picked more than once in a run of MODIFIED-SEARCH we have $C = C_i = C_j$ for some $i < j$. Let $y$ be a literal in $C$ that is satisfied after iteration $i$ and consider the smallest $l$, such that $i < l \leq j$ and either $y$ or $\overline{y}$ is in $C_l$. Actually it can not be the case that $y$ is in $C_l$ since the assignments in the iterations $i, \ldots, l$ satisfy $y$, and the algorithm only picks unsatisfied clauses. Thus $\overline{y}$ is in $C_l$ and therefore $l < j$ and $C_l \in \Gamma'_\varphi(C)$. This shows that in the constructed witness tree, the node added for $C_j$ is lower in the tree than the one for $C_l$ which is again lower than the one for $C_i$. So $C_j$ will not be a child of $C_i$. ∎

Since there are less $(d, m)$-forests than $(d+1, m)$-forests, we can improve the bounds given in Theorem 6. On the other hand, in $s$ iterations MODIFIED-SEARCH uses random sequences $w = w_0 w_1$, where $w_0 \in \{0,1\}^n$

and $w_1 \in \{1, \ldots, 2^k - 1\}^s$. Since there are only $2^n(2^k-1)^s$ such sequences we just can guarantee the existence of a $w$ with $K(\langle w \rangle \mid \varphi, s) \geq n + s \log(2^k - 1)$, where $\langle \cdot \rangle$ is a suitable binary encoding of these sequences.

**Theorem 8** *Let $\varphi$ be a formula in $k$-CNF and $d = d'_\varphi$. If*

$$\frac{d^d}{(d-1)^{d-1}} \leq 2^k - 1.$$

*then* Search *and* ModifiedSearch *find a satisfying assignment for $\varphi$ in expected time $O(m)$.*

*Proof.* Due to linearity of expectation, the expected number of iterations of Search on $\varphi$ is $2^k/(2^k - 1)$ times the expected number of iterations of ModifiedSearch. Therefore it suffices to show that ModifiedSearch will find a satisfying assignment for $\varphi$ with $O(m)$ expected number of iterations.

Assume that ModifiedSearch makes at least $s$ iterations of the while-loop. Fix a random sequence $w = w_0 w_1$, where $w_0 \in \{0,1\}^n$ and $w_1 \in \{1, \ldots, 2^k - 1\}^s$ such that

$$K(\langle w \rangle \mid \varphi, s) \geq n + s \log(2^k - 1) - c \tag{11}$$

We can reconstruct $w$ from the $(d, m)$-witness forest defined by the LOG of Modified-Search. Hence for some $c_A \geq 0$

$$\begin{aligned} K(\langle w \rangle \mid \varphi, s) &\leq n + \lceil \log F_{d,m}(s) \rceil + c_A \\ &< n + (ds + m) h\left(\frac{1}{d}\right) + 1 + c_A \,. \end{aligned}$$

Combined with (11) this implies

$$\left(\log(2^k - 1) - d\, h\left(\frac{1}{d}\right)\right) s < m\, h\left(\frac{1}{d}\right) + c + c_A + 1 \,.$$

Note that $\frac{d^d}{(d-1)^{d-1}} \leq 2^k - 1$ is equivalent to $0 < \log(2^k - 1) - d\, h\left(\frac{1}{d}\right)$ for integral $d \geq 2$. Therefore we get

$$s < \frac{m\, h\left(\frac{1}{d}\right) + c + c_A + 1}{k - d\, h\left(\frac{1}{d}\right)} = O(m).$$

This shows that the algorithms outputs a satisfying in less than this number of iterations. Notice that a fraction of $1 - 2^{-c}$ of the possible sequences $w$ fulfills this condition (by equation (5)).

Now assume that $w$ is choosen at random by the algorithm. Let $S$ be the random variable denoting the number of iterations of ModifiedSearch. For any constant $c \geq 0$ we have

$$\Pr\left(S < \frac{m\,h\left(\frac{1}{d}\right) + c + c_A + 1}{k - d\,h\left(\frac{1}{d}\right)}\right) \geq 1 - 2^{-c}.$$

Similar to the proof of Theorem 6 we obtain $\mathrm{E}[S] = O(m)$. ■

Note that Theorem 2 follows from Theorem 8.


### Acknowledgments

We want to thank Patrick Scharpfenecker and Uwe Schöning for helpfull discussions.


## References


[Alo91] N. Alon. A parallel algorithmic version of the Local Lemma. *Random Structures and Algorithms*, 2(4):367–378, 1991.

[ASE92] N. Alon, J. Spencer, and P. Erdős. *The probabilistic method.* Wiley, 1992.

[Bec91] J. Beck. An algorithmic approach to the Lovász Local Lemma. *Random Structures and Algorithms*, 2(4):343–366, 1991.

[Cal02] C. Calude. *Information and Randomness. An Algorithmic Perspective, 2nd ed.* Springer, 2002.

[EL75] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal and V. Sós, editors, *Infinite and Finite Sets*, pages 609–627. North-Holland, 1975.

[ES91] P. Erdős and J. Spencer. Lopsided Lovász Local Lemma and latin transversals. *Discrete Applied Mathematics*, 30:151–154, 1991.

[For09] L. Fortnow. A Kolmogorov complexity proof of the Lovász Local Lemma. Computational Complexity Blog, http://blog.computationalcomplexity.org/2009/06/kolmogorov-complexity-proof-of-lov.html, 2009.

[GKP94] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd edition, 1994.

[GMSW09] H. Gebauer, R. Moser, D. Scheder, and E. Welzl. The Lovász Local Lemma and satisfiability. In *Efficient Algorithms*, LNCS 5760, pages 30–54. Springer, 2009.

[GST11] H. Gebauer, T. Szabó, and G. Tardos. The Local Lemma is tight for SAT. In *Accepted for SODA 2011, 22nd ACM-SIAM Symposium an Discrete Algorithms*, 2011.

[LV93] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 1993.

[McD97] C. McDiarmid. Hypergraph coloring and the Lovász Local Lemma. *Discrete Mathematiks*, 167/168:481–486, 1997.

[Mos08] Robin A. Moser. Derandomizing the Lovász Local Lemma more efficiently. Technical Report arXiv:0807.2120v2, arXive.org, http://arxiv.org/abs/0807.2120, 2008.

[Mos09] Robin A. Moser. A constructive proof of the Lovász Local Lemma. In *41th Symposium on Theory on Computing (STOC)*, pages 343–350, 2009.

[MT10] R. Moser and G. Tardos. A constructive proof of the general Lovász Local Lemma. *Journal of the ACM*, 57(2):11:1–11:15, 2010.

[Sch01] U. Schöning. *Algorithmik*. Spektrum, 2001.

[Spe10] J. Spencer. Robin Moser makes Lovász Local Lemma algorithmic! http://cs.nyu.edu/spencer/moserlovasz1.pdf, 2010.

[Sri08] A. Srinivasan. Improved algorithmic versions of the Lovász Local Lemma. In *9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 611–620, 2008.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de