The Local Lemma is Tight for SAT

H. Gebauer * T. Szabó † G. Tardos ‡

Abstract

We construct unsatisfiable k-CNF formulas where every clause has k distinct literals and every variable appears in at most $\left(\frac{2}{e}+o(1)\right)\frac{2^k}{k}$ clauses. The lopsided Local Lemma shows that our result is asymptotically best possible: every k-CNF formula where every variable appears in at most $\frac{2^{k+1}}{e(k+1)}-1$ clauses is satisfiable. The determination of this extremal function is particularly important as it represents the value where the k-SAT problem exhibits its complexity hardness jump: from having every instance being a YES-instance it becomes NP-hard just by allowing each variable to occur in one more clause.

The asymptotics of other related extremal functions are also determined. Let $l(k)$ denote the maximum number, such that every k-CNF formula with each clause containing k distinct literals and each clause having a common variable with at most $l(k)$ other clauses, is satisfiable. We establish that the bound on $l(k)$ obtained from the Local Lemma is asymptotically optimal, i.e., $l(k)=\left(\frac{1}{e}+o(1)\right)2^k$.

The constructed formulas are all in the class MU(1) of minimal unsatisfiable formulas having one more clause than variables and thus they resolve these asymptotic questions within that class as well.

The SAT-formulas are constructed via the binary trees of [10]. In order to construct the trees a continuous setting of the problem is defined, giving rise to a differential equation. The solution of the equation diverges at 0, which in turn implies that the binary tree obtained from the discretization of this solution has the required properties.

1 Introduction

The satisfiability of Boolean formulas is the archetypical NP-hard problem. Somewhat unusually we define a k-CNF formula as the conjunction of clauses that are the disjunction of exactly k distinct literals. (Note that most texts allow shorter clauses in a k-CNF formula, but fixing the exact length will be important for us later on.) The problem of deciding whether a k-CNF formula is satisfiable is denoted by k-SAT, it is solvable in polynomial time for $k=2$, and is NP-complete for every $k\geq 3$ as shown by Cook [3].

*Institute of Theoretical Computer Science, ETH Zurich, CH-8092, Switzerland; mail: gebauerh@inf.ethz.ch.

†Department of Mathematics and Computer Science, Freie Universität Berlin, 14195 Berlin, Germany; mail: szabo@zedat.fu-berlin.de.

‡Simon Fraser University, Burnaby BC, Canada and Rényi Institute, Budapest, Hungary; mail: tardos@cs.sfu.ca.

Papadimitriou and Yannakakis [18] have shown that k-SAT is even MAX-SNP-complete for every $k\geq 2$.

The first level of difficulty in satisfying a CNF formula arises when two clauses share variables. For a finer view into the transition to NP-hardness, a grading of the class of k-CNF formulas can be introduced, that limits how much clauses interact locally. A k-CNF formula is called a (k,s)-CNF formula if every variable appears in at most s clauses. The problem of satisfiability of (k,s)-CNF formulas is denoted by (k,s)-SAT.

Tovey [24] proved that while every $(3,3)$-CNF formula is satisfiable (due to Hall's theorem), the problem of deciding whether a $(3,4)$-CNF formula is satisfiable is already NP-hard. Dubois [5] showed that $(4,6)$-SAT and $(5,11)$-SAT are also NP-complete.

Kratochvíl, Savický, and Tuza [15] defined the value $f(k)$ to be the largest integer s such that every (k,s)-CNF is satisfiable. They also generalized Tovey's result by showing that for every $k\geq 3$ $(k,f(k)+1)$-SAT is already NP-complete. In other words, for every $k\geq 3$ the (k,s)-SAT problem goes through a kind of "complexity phase transition" at the value $s=f(k)$. On the one hand the $(k,f(k))$-SAT problem is trivial by definition in the sense that every instance of the problem is a "YES"-instance. On the other hand the $(k,f(k)+1)$-SAT problem is already NP-hard, so the problem becomes hard from being trivial just by allowing one more occurrence of each variable. For large values of k this might seem astonishing, as the value of the transition is exponential in k: one might think that the change of just one in the parameter should have hardly any effect.

The complexity hardness jump is even greater: the problem of (k,s)-SAT is also MAX-SNP-complete for every $s>f(k)$ as was shown by Berman, Karpinski, and Scott [2] (generalizing a result of Feige [7] who showed that $(3,5)$-SAT is hard to approximate within a certain constant factor).

The determination of where this complexity hardness jump occurs is the topic of the current paper.

For a lower bound the best tool available is the Lovász Local Lemma. The lemma does not deal directly with number of occurrences of variables, but rather with pairs of clauses that share at least one variable.

We call such a pair an *intersecting pair* of clauses. A straightforward consequence of the lemma states that if every clause of a k-CNF formula intersects at most $2^k/e - 1$ other clauses, then the formula is satisfiable. It is natural to ask how tight this bound is and for that Gebauer et al. [9] define $l(k)$ to be the largest integer number satisfying that whenever all clauses of a k-CNF formula intersect at most $l(k)$ other clauses the formula is satisfiable. With this notation the Lovász Local Lemma implies that

$$l(k) \geq \left\lfloor \frac{2^k}{e} \right\rfloor - 1. \tag{1.1}$$

The order of magnitude of this bound is trivially optimal: $l(k) < 2^k - 1$ follows from the unsatisfiable k-CNF consisting of all possible k-clauses on only k variables.

In [9] a hardness jump is proved for the function l: deciding the satisfiability of k-CNF formulas with maximum neighborhood size at most $\max\{l(k)+2, k+3\}$ is NP-complete.

As observed by Kratochvíl, Savický and Tuza [15] the bound (1.1) immediately implies

$$f(k) \geq \left\lfloor \frac{l(k)}{k} \right\rfloor + 1 \geq \left\lfloor \frac{2^k}{ek} \right\rfloor. \tag{1.2}$$

From the other side Savický and Sgall [21] showed that $f(k) = O\left(k^{0.74} \cdot \frac{2^k}{k}\right)$. This was improved by Hoory and Szeider [12] who came within a logarithmic factor: $f(k) = O\left(\log k \cdot \frac{2^k}{k}\right)$. Recently, Gebauer [10] showed that the order of magnitude of the lower bound is correct and $f(k) = \Theta(\frac{2^k}{k})$.

More precisely, the construction of [10] gave $f(k) \leq \frac{63}{64} \cdot \frac{2^k}{k}$. The constant factor $\frac{63}{64}$ was clearly an artefact of the proof and seemed to be pushing the humanly possible boundaries of the approach of [10] in the sense that the technicalities of the proof became more and more intricate and unmanageable with no hope of converging to any conclusion. Determining $f(k)$ asymptotically remained an outstanding open problem and there was no clear consensus about where the correct asymptotics should fall between the constants $1/e$ of [15] and $63/64$ of [10]. In fact several of the open problems of the recent survey of Gebauer, Moser, Scheder, and Welzl [9] is centered around the understanding of this question.

In our main theorem we settle these questions from [9] and determine the asymptotics of $f(k)$. We show that the lower bound (1.2) can be strengthened by a factor of 2 and that this bound is tight.

THEOREM 1.1.

$$f(k) = \left(\frac{2}{e} + O\left(\frac{1}{\sqrt{k}}\right)\right)\frac{2^k}{k}.$$

For the upper bound we use the fundamental binary tree approach of [10]. We define a suitable continuous setting for the construction of the appropriate binary trees, which allows us to study the problem via a differential equation. The solution of this differential equation corresponds to our construction of the binary trees, which then can be given completely discretely.
The lower bound is achieved via the lopsided version of the Lovász Local Lemma. The key of the proof is to assign the random values of the variables counter-intuitively: each variable is *more* probable to satisfy those clauses where it appears as a literal with its *less* frequent sign. The lower bound can also be derived from a theorem of Berman, Karpinski and Scott [2] tailored to give good lower bounds on $f(k)$ for small values of k. In [2] the asymptotic behavior of the bound is not calculated, since the authors do not believe in its optimality. In the Appendix we reproduce a simple argument giving this asymptotics, because the proof of [2] contains a couple of errors, so the unusual choice of the probabilities is not apparent.

Here we only give the intuition of where the factor two improvement is coming from and how to achieve it. The lopsided version of the local lemma [6] allows for a more restricted definition of "intersecting" clauses in a CNF formula. Namely, one can consider two clauses intersect only if they contain a common variable with *different sign* and this still allows the same conclusion as in the original Local Lemma. If all variables in a (k, s)-CNF are *balanced*, that is they appear an equal number of times with either sign, then each clause intersects only at most $ks/2$ other clauses in this restricted sense, instead of the at most $k(s-1)$ other clauses it may intersect in the original sense and the factor two improvement is immediate. To handle the unbalanced case we consider a distribution on assignments where the variables are assigned true or false values with some bias. It would be natural to favor the assignment that satisfies more clauses, but the opposite turns out to be the distribution that works. This is because the clauses with some variables receiving the *less frequent sign* are those that intersect more than average other clauses, so those are the ones whose satisfiability should be boosted with the bias put on the assignments.

Since the (lopsided) Lovász Local Lemma was fully algorithmized by Moser and Tardos [17] we now have that not only every (k, s)-CNF formula for $s = \lfloor 2^{k+1}/(e(k+1)) \rfloor$ has a satisfying assignment but there

is also an algorithm that *finds* such an assignment in probabilistic polynomial time. Moreover, for just a little bit larger value of the parameter s one cannot find a satisfying assignment efficiently simply because already the decision problem is NP-hard.

Our construction also shows that the lower bound (1.1) on $l(k)$ is asymptotically tight.

THEOREM 1.2.

$$l(k) = \left(\frac{1}{e} + O\left(\frac{1}{\sqrt{k}}\right)\right) 2^k.$$

One may wonder how the lower bound on $l(k)$ implied by the (original) Local Lemma is tight and no improvement can be achieved using the lopsided version. This is no surprise when we see that in the formulas we construct every pair of clauses that intersect in the original sense also do in the more restricted sense.

2 Formal Definitions and the Informal Continuous Construction

The proof of the upper bounds of Theorem 1.1 and 1.2 contain several technically involved arguments. In this section we give most of the formal definitions and sketch the main informal ideas behind the construction. Even though the final construction can be formulated without mentioning the underlying continuous context, we feel that an informal description greatly helps in motivating it.

2.1 Binary Trees We start by collecting the necessary ingredients from [10]. We consider only binary trees where every node has either two or no children. For every such binary tree T with all leaves having depth at least k one can construct a k-CNF formula F as follows. For every non-leaf node $v \in V(T)$ we create a variable x_v and label one of its children with the literal x_v and the other with $\bar{x}_v$. We do not label the root. With every leaf $w \in V(T)$ we associate a clause C_w by walking along the path from w towards the root and collecting the first k labels encountered (including the one at w). The set of clauses C_w, over all leaves w of T, constitutes the formula F.

Observation F is unsatisfiable.

Indeed, any assignment α of the variables defines a path from the root to some leaf w, by always proceeding to the unique child whose label is mapped to 0 by α; thus C_w is violated by α.

Hence our task of constructing unsatisfiable formulas with low variable-degree can possibly be achieved via constructing binary trees where each node has low number of leaf descendants within distance k. Formally, we say that a leaf w is *l-close* to a vertex v if v is an ancestor of w, at distance at most l from v. We call a binary tree T a *(k,d)-tree* if

(i) every leaf has depth at least k and

(ii) for every node u of T there are at most d leaves that are k-close to u.

Note that this definition is slightly different from the one in [10].

The next lemma states the connections between (k,d)-trees and unsatisfiable k-CNF formulas with bounded number of conflicts. It is a slight modification of Lemma 1.6 in [10].

LEMMA 2.1. *Let T be a (k,d)-tree. Then there are unsatisfiable CNF formulas $F = F(T), F' = F'(T)$ with the following properties.*

(a) *F is a (k,d)-CNF formula. In particular $f(k) \leq d-1$.*

(b) *F' is a $(k+1,2d)$-CNF formula where every clause intersects at most $(k+1)d$ other clauses. In particular $l(k+1) \leq (k+1)d - 1$.*

Note that Lemma 2.1 reduces the proof of the upper bounds in Theorem 1.1 and Theorem 1.2 to the construction of a $\left(k, \left(\frac{2}{e} + O(1/\sqrt{k})\right)\frac{2^k}{k}\right)$-tree and this is what we concentrate on in the remaining of this section.

2.2 Vectors and constructibility Given a tree T, we assign a *leaf-vector* $\vec{d}_w = (x_0, x_1, \ldots, x_k)$ to each node $w \in V(T)$, where x_i denotes the number of leaf-descendants of w having distance i to w. E.g., for a leaf w we have $\vec{d}_w = (1, 0, \ldots, 0)$, for the root w of a full binary tree of depth $l \leq k$ we have $\vec{d}_w = (0, 0, \ldots, 0, 2^l, 0, \ldots, 0)$. By definition, for every node w of a (k,d)-tree we have $\vec{d}_w = (x_0, x_1, \ldots, x_k)$ with $|\vec{d}_w| := \sum_{i=0}^{k} x_i \leq d$.

For some vector $\vec{x}$ with $|\vec{x}| \leq d$ we define a *$(k,d,\vec{x})$-tree* as a tree where (i) the root has a leaf-vector $\vec{y}$ which is coordinate-wise dominated by $\vec{x}$ (i.e., $y_i \leq x_i$ for every i), and (ii) each vertex has at most d leaf-descendants that are k-close. E.g., a tree consisting of a parent with two children is a $(k,d,\vec{x})$-tree, for every $\vec{x}$ with $x_1 \geq 2$ and $\sum_{i=0}^{k} x_i \leq d$. A vector $\vec{x}$ is *(k,d)-constructible* (or *constructible* if k and d are clear from the context), if a $(k,d,\vec{x})$-tree exists. E.g., $(1, 0, \ldots, 0)$, or more generally $(\underbrace{0, 0, \ldots, 0}_{l}, 2^l, 0, \ldots, 0)$ are constructible as long as $2^l \leq d$. Observe that the constructibility of the vector $(0, \ldots, 0, r)$ for some $r \leq d$ readily implies the existence of a (k,d)-tree.

If $|\vec{x}| \leq d$ then $\vec{x}$ is a (k,d)-*vector*. For a vector $\vec{x} = (x_0, \ldots, x_k)$ the *weight* $w(\vec{x})$ is $\sum_{i=0}^{k} x_i/2^i$. The next lemma, which is a dual version of Kraft's inequality, gives a useful necessary condition for the constructibility of a vector.

LEMMA 2.2. *Let $\vec{x}$ be a (k,d)-vector. If $w(\vec{x}) \geq 1$ then $\vec{x}$ is constructible.*

In this terminology the main result of [10] (giving a weaker constant factor) established the constructibility of the vector $\vec{v} = (0, \ldots, 0, 1, 2, \ldots, 2^s)$, provided $s = k - \log k + 1$ and $d \geq 2^{s+1}$. This is an immediate consequence of Lemma 2.2. By considering a full binary tree T with all leaves at depth s and attaching a $(k,d,\vec{v})$-tree to every leaf l of T (such that l is the root) one can obtain the constructibility of $(0, \ldots, 0, 2^s)$, which directly implies the existence of a (k,d)-tree for infinitely many k and $d = \frac{2^{k+2}}{k}$.

A non-leaf vertex v of a tree "distributes" its k-close leaf descendants between its children w' and w''. That is, if $\vec{d}_{w'} = (x'_0, x'_1, \ldots, x'_k)$ and $\vec{d}_{w''} = (x''_0, x''_1, \ldots, x''_k)$, then we have

$$\vec{d}_v = (0, x'_0 + x''_0, x'_1 + x''_1, \ldots, x'_{k-1} + x''_{k-1}) \tag{2.3}$$

We will consider two fundamentally different kinds of way a parent vertex v with leaf-vector $\vec{d}_v = (x_0, x_1, \ldots, x_k)$ can distribute its k-close leaf-descendants between its children. First, a distribution is *even* if $x'_i = x''_i = \frac{x_{i+1}}{2}$ for every $i \in \{1, \ldots, k\}$. (Assume for the moment that the coordinates of $\vec{d}_v$ are even.)

OBSERVATION 2.3. *Let m be an integer and let $\vec{x} = (x_0, x_1, \ldots, x_k)$ be a (k,d)-vector where x_i is divisible by 2^m, for every $i \geq m$. Then $\vec{x}^{(m)} = (\frac{x_m}{2^m}, \frac{x_{m+1}}{2^m}, \ldots, \frac{x_k}{2^m}, \frac{d}{2^m}, \frac{d}{2^{m-1}}, \ldots, \frac{d}{2})$ is a (k,d)-vector. Moreover, if $\vec{x}^{(m)}$ is constructible then $\vec{x}$ is constructible.*

The first statement is immediate. For the second statement, attach a copy of a $(k,d,\vec{x}^{(m)})$-tree to each leaf of a full binary tree of depth m. This gives a $(k,d,\vec{x})$-tree.

Secondly, a distribution is *piecewise* if there is some threshold index t such that for $i \leq t-1$, $x'_i = x_{i+1}$ and $x''_i = 0$, whereas for $t \leq i \leq k-1$, $x'_i = 0$ and $x''_i = x_{i+1}$. If the last coordinates $x'_k = x''_k = 0$ then $\vec{x}'$ and $\vec{x}''$ are (k,d)-vectors provided $\vec{x}$ is a (k,d)-vector. Furthermore the constructibility of $\vec{x}'$ and the constructibility of $\vec{x}''$ imply the constructibility of $\vec{x}$.

We build our (k,d)-tree from the root down to the leaves, starting with the vector $(0, \ldots, 0, 1, 2, \ldots, \frac{d}{2})$. Here we assume that d is a large power of 2, and as d is of the order $2^k/k$ the number of 0s at the beginning is about $\log k$. The typical sub-routine we use to define the leaf-vectors of the children of some node is the one we call "*cut at t and split*". This involves first a piecewise distribution at t, such that Lemma 2.2 is applicable for w', that is $\sum_{i=0}^{t-1} x_{i+1}/2^i \geq 1$. Then the even distribution is applied to the child w'' $m = \log x_{t+1}$-times (c.f. Observation 2.3) to produce a leaf vector $(0, \ldots, 0, 1, \frac{x_{t+2}}{2^m}, \ldots, \frac{x_k}{2^m}, 0, \frac{d}{2^m}, \frac{d}{2^{m-1}}, \ldots, \frac{d}{2})$ (assuming, as we will have it, that all x_is are large enough powers of 2). In fact we produce 2^m copies of this vector, but since its constructibility implies the constructibility of $\vec{x}$ by Observation 2.3 we concentrate on producing one.

In the following we give an indication how, using only the simple sub-routine of cut and split, can produce a (k,d)-tree with $\frac{1}{2-\epsilon} \cdot \frac{2^{k+1}}{k}$ for every $\epsilon > 0$. Then we introduce a biased version of cuts which allows us to push the bound on d to the limit.

2.3 Passing to continuous The goal of this subsection is to give motivation behind the formal proof of the next section. Recall that our goal is to obtain a (k,d)-tree for

$$d = \frac{1}{T}\frac{2^{k+1}}{k} \tag{2.4}$$

where T should be as large as possible. By our lower bound we know that we can not achieve $T \geq e$, our goal is to have $T \geq e - \epsilon$ for every $\epsilon > 0$.

After fixing a target constant T, it will be helpful to consider the leaf-vectors in a normalized form, which then will enable us to interpret them as continuous functions on the $[0,1]$ interval. For a given target d and leaf-vector $x = (x_0, \ldots, x_k)$ the *normalized vector*, $\vec{y} = \vec{y}(\vec{x})$ is defined as $(y_0, \ldots, y_k)$ with $y_j = \frac{1}{2^j}\frac{2^{k+1}}{d}x_j$, i.e., $\bar{x} = (2^{k+1}\frac{x_0}{d}, 2^k\frac{x_1}{d}, \ldots, x_{k+1-\log d}, 2x_{k+2-\log d}, \ldots, 4\frac{x_{k-1}}{d}, 2\frac{x_k}{d})$. We say that a normalized vector $\vec{y}(\vec{x})$ is constructible if $\vec{x}$ is constructible. The next observation follows directly from Lemma 2.2 and (2.4).

OBSERVATION 2.4. *Let $\vec{y} = (y_0, y_1, \ldots, y_k)$ be a normalized vector. If $\sum_{i=0}^{k} y_i \geq Tk$ then $\vec{y}$ is constructible.*

The relationship between the normalized vectors of a parent and its children is also described by an equation similar to (2.3). The normalized vector of the parent is

$$(0, \frac{y'_0 + y''_0}{2}, \frac{y'_1 + y''_1}{2}, \ldots, \frac{y'_{k-1} + y''_{k-1}}{2}) \tag{2.5}$$

We use this normalizing operation to help us see the leaf-vectors more and more as functions, defined on the $[0,1]$ interval, possibly in a continuous manner. Indeed, if k is large enough then normalized vectors can be represented as step-functions.

For example the normalizing operation transforms the leaf-vector
$(0,\ldots,0,1,2,\ldots,2^s)$ with $2^s = d/2$, into the normalized $(0,\ldots,0,1,1,\ldots,1)$. If $s = k - \log k$, then only $o(k)$ entries are 0 and most entries are 1. We want to disregard this small error and consider this normalized vector as the constant 1 function defined on the interval $[0,1]$.

In general we will talk about real functions on the interval $[0,1]$ and will routinely ignore the little $o(1)$ sized pieces — by choosing k large enough the normalized vectors are approximated well by the real functions and the $o(1)$ errors will be insignificant.

The following is a reformulation of Observation 2.4 into the continuous setting. (The $\star$ denotes that this is not a formal statement.)

LEMMA* 2.5. *If $\int_{x=0}^{1} f(x)\,dx \geq T$ then f is constructible.*

We say that f is *easy constructible* if Lemma* 2.5 applies.

First we take a look at how the sub-routine "cut and split" of the previous subsection can be pushed to give an easily constructible function with target $T = 2 - \epsilon$.

In the continuous the subroutine will be called "cut at v and split" which, given a function f on $[0,1]$ and a value v between 0 and 1, creates the following two functions (corresponding to the two leaf-vectors in the previous section):

$$f_{\text{right}}(x) = \begin{cases} 2f(x+v) & x \in [0, 1-v) \\ 1 & x \in [1-v, 1] \end{cases}$$

$$f_{\text{left}}(x) = \begin{cases} 2f(x) & x \in [0, v) \\ 0 & x \in [v, 1] \end{cases}$$

As in the cut and split subroutine of the previous section, where the child w' is expected to use Lemma 2.2, here we want that the function f_{left} is able to use Lemma* 2.5 and hence is easily constructible.

Starting with the constant 1 function f and performing a cut at $1-\delta$ with any constant $\delta < \epsilon/4$, we obtain that the integral of f_{left} is at least $2-\epsilon$, hence Lemma* 2.5 applies. The other child, f_{right} on the other hand is significantly improved compared to his parent (as it should be!): its value on the interval $[0,\delta)$ is not 1, but 2. Hence we are able to cut f_{right} at $1-2\delta$ and have the left child large enough integral. It turns out that repeatedly cutting at $v_i = 1 - i\delta$ the right-child functions for $i = 1, 2\ldots$ the left-functions can always immediately use Lemma* 2.5 with target $T = 2-\epsilon$ and be easily constructible. At the same time it is easy to see that the integral of the right-functions grows in every step and by the time cut point v_i reaches $1/3$ it gets larger than 2 and thus Lemma* 2.5 applies to it.

In order to reach target $T = e - \epsilon$ we need a generalization of the cut and split subroutine of above.

Our goal is to create a subroutine which for a given function f creates two functions $f_{\text{left}}, f_{\text{right}}$ in which f_{left} is easily constructible. The idea is to cut very close to 0, and distribute the possibly very little gain in the integral of f_{right} more evenly on the whole interval $[0,1]$. In order to be able to cut very close to 0 and still being able to use Lemma* 2.5 we need to multiply the value of the function left of the cut by a large number (we will use 2^l) to compensate for the shortness of the cut. We do that by actually distributing the function f into 2^l functions averaging f, out of one will be the left part multiplied by 2^l and all the others will be the right part multiplied by $1 + 1/(2^l - 1)$. This can be done by a depth l full binary tree connecting the corresponding trees. At the end such an "*l-deep cut at v and split*" subroutine will create from a given function f on $[0,1]$ two functions:

$$f_{\text{right}}(x) = \begin{cases} \frac{2^l}{2^l-1} f(x+v) & x \in [0, 1-v) \\ 1 & x \in [1-v, 1] \end{cases}$$

$$f_{\text{left}}(x) = \begin{cases} 2^l f(x) & x \in [0, v) \\ 0 & x \in [v, 1] \end{cases}$$

We can cut very close to 0 and this is what we will do. On the other hand, the improvement in the value of the right-function is then hardly visible, but is very evenly distributed and is tailored for continuous analysis.

We define a function that approximates well how the right-functions develop if we repeatedly cut very close to 0. For this we will define a two variable function $F(t,x)$ where t represents the time that has elapsed since we started our process and $x \in [0,1]$ is the variable of our current right-function. We will have the initial conditions $F(0,x) = 1$ (we start with the constant 1 function) and $F(t,1) = 1$ (after a cut, always 1 enters from the right side during a split). For each t, $F(t,x)$ should be a good approximation of the right-function that we get by starting with the constant 1 function and cut always at infinitesimally small values until our cumulative cut is t.

Let $F(t,x)$ be our current right-function (defined on the $[0,1]$ interval with variable x) for some fixed t. What happens if we cut at some infinitesimally small $\delta \in [0,1]$ with a cut of sufficiently large depth? In order to have the left child use Lemma* 2.5 immediately the integral of the function on the interval $[0,\delta]$, times 2^l should be at least T: approximating the integral with $2^l \delta F(t,0) \geq T$, so $l = \log_2(T/(\delta F(t,0)))$. After this cut the right-function $F(t+\delta, x)$ is 1 on the infinitesimally small interval $[1-\delta, 1]$. On the long interval our old right-function is multiplied by a factor depending on δ,

which in turn depends on the depth of the cut giving the factor $(1+1/(2^l-1)) \approx 1+(\delta F(t,0))/T$. Imagining that our function changes with unit speed over time

$$F(t+\delta, x-\delta) = F(t,x)(1+\frac{\delta F(t,0)}{T}).$$

This gives us an equation on the derivative of $F(t,x)$ (in direction $(-1,1)$) which describes how fast the function values tend to change

$$\frac{F'(t,x)}{F(t,x)} = \frac{F(t,0)}{T}.$$

Integrating on the segment from $(s-1,1)$ to $(s,0)$ we obtain

$$\int (\ln F)' = \frac{1}{T}\int_{s-1}^{s} F(t,0)dt.$$

The left side evaluates to $\ln F(s,0)$ by the initial condition. Estimating $F(s,0)$ gives that $F(s,0) \geq \exp(\frac{1}{T}F(s-1,0))$. Hence the function $F(s,0)$ tends to infinity if the constant T is less than e, showing the right-function can also use Lemma* 2.5 after a while.

The above continuous heuristic is the underlying idea of the construction described in the Appendix. It provides a good approximation to what happens in the discrete case, even though a large number of small errors must be introduced and the handling of all of them is quite technical. Crucially the number of these errors depends only on the given ϵ and hence one can plan for them in advance.

3 The Class MU(1) and Outlook

3.1 The class MU(1) The function $f(k)$ is not known to be computable. In order to still be able to upper bound its value, one tries to restrict to a smaller/simpler class of formulas. When looking for unsatisfiable (k,s)-CNF formulas it is naturally enough to consider *minimal unsatisfiable formulas*, i.e., unsatisfiable CNF formulas that become satisfiable if we delete any one of their clauses. The set of minimal unsatisfiable CNF formulas is denoted by MU. As observed by Tarsi (c.f. [1]), all formulas in MU have more clauses then variables, but some have only one more. The class of these MU formulas, having one more clauses than variables is denoted by MU(1). This class has been widely studied (see, e.g., [1], [4], [13], [16], [23]). Hoory and Szeider [11] considered the function $f_1(k)$, denoting the largest integer such that no $(k, f_1(k))$-CNF formula is in MU(1), and showed that $f_1(k)$ is computable. Their computer search determined the values of $f_1(k)$ for small k: $f_1(5) = 7$, $f_1(6) = 11$, $f_1(7) = 17$, $f_1(8) = 29$, $f_1(9) = 51$. Via the trivial inequality $f(k) \leq f_1(k)$, these are the best known upper bounds on $f(k)$ in this range. In contrast, even the value of $f(5)$ is not known.

It is an interesting open problem whether $f(k) = f_1(k)$ for every k. Our upper bound construction from Theorem 1.1 does reside in the class MU(1) and hence shows that $f(k)$ and $f_1(k)$ are equal asymptotically: $f(k) = (1+o(1))f_1(k)$.

Scheder [22] showed that for almost disjoint k-CNF formulas (i.e. CNF-formulas where any two clauses have at most one variable in common) the two functions are not the same. That is, if $\tilde{f}(k)$ denotes the maximum s such that every almost disjoint (k,s)-CNF formula is satisfiable, for k large enough every unsatisfiable almost disjoint $(k, \tilde{f}(k)+1)$-CNF formula is outside of MU(1).

3.2 Constructing binary trees and MU(1) formulas The structure of MU(1) formulas is well understood and it is closely related to binary trees. In particular, given any finite rooted binary tree T with the property that each non-leaf vertex has exactly two children we associate with it certain CNF formulas. Similarly to the previous section we start with assigning distinct literals to the vertices, assigning the negated and non-negated form of the same variable to the two children of any non-leaf vertex. We do not assign any literal to the root. For each leaf of T select a clause that is the disjunction of *some* literals along the path from the root to that leaf and consider the CNF formula F that is the conjunction of one clause for each leaf. Clearly, F is unsatisfiable and it has one more clauses than the number of variables associated with T. Note that in our construction for proving the upper bound in Theorem 1.1 we restrict to formulas obtained from binary trees by selecting on every root-leaf path the k vertices farthest from the root.

As proved in [4] F is a MU(1) formula if and only if all literals associated to vertices of T *do* appear in F, furthermore every formula in MU(1) can be obtained from a suitable binary tree this way. Similarly, if a (k,d)-tree T is minimal with respect to the number of leaves then the corresponding k-CNF formula $F = F(T)$ we consider in our construction is in MU(1).

Let f_2 be the largest integer d such that no (k,d)-tree exist. Then, clearly, $f(k) \leq f_1(k) \leq f_2(k)$ and we show $f(k) = (1+o(1))f_2(k)$.

3.3 On the size of unsatisfiable formulas By the *size* of a tree we mean the number of its leaves and by the *size* of a CNF formula we mean the number of its clauses. With this notation the size of a (k,d)-tree and the size of the corresponding (k,d)-CNF in MU(1) are the same.

When proving the upper bound of Theorem 1.1 we

constructed (k,d)-trees for $d \approx \frac{2^{k+1}}{ek}$. Their size and therefore the size of the corresponding (k,d)-CNF in MU(1) is at most 2^h, where h is the *height* of the tree: the largest root-leaf distance. In fact, the size of the trees we constructed are very close to this upper bound. Therefore it makes sense to take a closer look at the height.

Recall that we associated with a vertex v of a (k,d)-tree the (k,d)-vector $(x_0,\ldots,x_k)$, where x_j is the number of leaf-descendants of v of distance j from v. A minimal (k,d)-tree has no two vertices along the same branch with identical vectors, so the height of a minimal (k,d)-tree is limited by the number of (k,d)-vectors, less than $(d+1)^{k+1}$. For $d = f(k)+1$ this is $2^{\Theta(k^2)}$. The same bound for minimal (k,d)-CNF formulas in MU(1) is implicit in [11]. There is numerical evidence that the size of the minimal $(k,f(k)+1)$-tree and the minimal (k,d)-CNF in MU(1) might indeed be doubly exponential in k (consider the size of the minimal $(7,18)$-tree and the minimal $(7,18)$-CNF in MU(1) mentioned below).

A closer analysis of the proof of Theorem 1.1 shows that the height of the (k,d)-tree constructed in it is at most kd. While this is better than the general upper bound above it still allows for trees with sizes that are doubly exponential in k.

This height can, however, be substantially decreased if we allow the error term in d to slightly grow. If we allow $d = (1+\epsilon)\frac{2^{k+1}}{ek}$ for a fixed $\epsilon > 0$, then a more careful analysis shows that the height of the tree created becomes $O(k)$. This bounds the size of the tree and the corresponding formula by a *polynomial in d*.

Let us define $f_1(k,d)$ for $d > f_1(k)$ to be the size of the smallest (k,d)-CNF in MU(1) and let $f_2(k,d)$ stand for the size of the smallest (k,d)-tree, assuming $d > f_2(k)$. While $f_1(k,f_1(k)+1)$ and similarly $f_2(k,f_2(k)+1))$ are probably doubly exponential in k, for slightly larger values of d $f_1(k,d)$ and $f_2(k,d)$ are polynomial in d (and thus simply exponential in k).

Finally we mention the question whether $f_1(k) = f_2(k)$ for all k. In other words, we ask whether we lose by selecting the k vertices farthest from the root when making a MU(1) k-CNF formula from a binary tree. As mentioned above, $f(k) = f_1(k)$ is also open, but $f_1(k) = f_2(k)$ seems to be a simpler question as both functions are computable. Computing their values up to $k = 8$ we found these values agreed. To gain more insight we computed the corresponding size functions too and found that $f_1(k,d) = f_2(k,d)$ for $k \leq 7$ and all $d > f_1(k)$ with just a single exception. We have $f_1(7) = 17$ and $f_1(7,18) = 10,197,246,480,846$, while $f_2(7,18) = 10,262,519,933,858$. Does this indicate that all other equalities are coincidences and f_1 and f_2 will eventually diverge?

A related algorithmic question is whether the somewhat simpler structure of (k,d)-trees can be used to find an algorithm computing $f_2(k)$ substantially faster than the algorithm of Hoory and Szeider [11] for computing $f_1(k)$. Such an algorithm would give useful estimates for $f_1(k)$ and also $f(k)$. At present we use a similar (and similarly slow) algorithm for either function.

Appendix

A Proof of the Lower Bound of Theorem 1.1

For our proof we use the Lopsided Local Lemma of Erdős and Spencer [6].

LEMMA A.1. *(Lopsided Local Lemma) Let $\{A_C\}_{C\in I}$ be a finite set of events in some probability space. Let $\Gamma(C)$ be a subset of I for each $C \in I$ such that for every subset $J \subseteq I \setminus (\Gamma(C) \cup \{C\})$ we have*

$$Pr(A_C | \wedge_{D\in J} \bar{A}_D) \leq Pr(A_C).$$

Suppose there are real numbers $0 < x_C < 1$ for $C \in I$ such that for every $C \in I$ we have

$$Pr(A_C) \leq x_C \prod_{D\in\Gamma(C)} (1-x_D).$$

Then

$$Pr(\wedge_{C\in I}\bar{A}_C) > 0.$$

Let F be a (k,s)-CNF formula with $s = \left\lfloor \frac{2^{k+1}}{e(k+1)} \right\rfloor$.

We set the values of the variables randomly and independently, but not according to the uniform distribution. This seems reasonable to do as the number of appearances of a variable x_i in F as a non-negated literal could be significantly different from the number of clauses where x_i appears negated. It is even possible that a variable x_i appears negated in only a few, maybe even in a single clause, in which case one tends to think that it is reasonable to set this variable to true with much larger probability than setting it to false. In fact it is exactly the opposite we will do. The *more* a variable appears in the clauses of F as non-negated, *the less likely* we will set it to true. The intuition behind this is explained in the introduction.

For a literal v we denote by d_v the number of occurrences of v in F. We set a variable x to true with probability $P_x = \frac{1}{2} + \frac{2d_{\bar{x}}-s}{2sk}$. This makes the negated version $\bar{x}$ satisfied with probability $P_{\bar{x}} = \frac{1}{2} - \frac{2d_{\bar{x}}-s}{2sk} \geq \frac{1}{2} + \frac{2d_x-s}{2sk}$ as we have $d_x + d_{\bar{x}} \leq s$. So any literal v is satisfied with probability at least $\frac{1}{2} + \frac{2d_{\bar{v}}-s}{2sk}$.

For each clause $C \in F$, we define the "bad event" A_C to be that C is not satisfied.

For every $C \in F$ we define $\Gamma(C)$ to be the family of clauses D in F that have at least one such variable in common with C whose sign is different in C and D.

Finally we set the value of each x_C to be $x = \frac{e}{2^k}$.

We need to check that for every subset $J \subseteq I \setminus (\Gamma(C) \cup \{C\})$ we have

$$Pr(A_C | \wedge_{D \in J} \bar{A}_D) \leq Pr(A_C).$$

This is true because of the FKG inequality [8].

We need to check also the other condition of the lemma. Let C be an arbitrary clause and let us denote the literals it contains by $v_1, \ldots, v_k$. For C to not to be satisfied we must not set any of the independent literals in C to true and therefore we have

$$\begin{aligned} Pr(A_C) &= \prod_{i=1}^{k}(1 - P_{v_i}) \\ &\leq \prod_{i=1}^{k}\left(\frac{1}{2} - \frac{2d_{\bar{v}_i} - s}{2sk}\right) \\ &\leq \frac{1}{2^k}\prod_{i=1}^{k}\left(\left(1 + \frac{1}{k}\right)\left(1 - \frac{ed_{\bar{v}_i}}{2^k}\right)\right) \\ &\leq \frac{\left(1 + \frac{1}{k}\right)^k}{2^k}\prod_{i=1}^{k}(1 - x)^{d_{\bar{v}_i}} \\ &< \frac{e}{2^k}(1 - x)^{|\Gamma(C)|} \\ &= x \prod_{D \in \Gamma(C)}(1 - x). \end{aligned}$$

As the conditions of the lopsided local lemma are satisfied, its conclusion must also hold. It states that the random evaluation of the variables we consider satisfies the (k, s)-CNF F with positive probability. Thus F must be satisfiable and we have $f(k) \geq s = \left\lfloor \frac{2^{k+1}}{e(k+1)} \right\rfloor$.

B Formal Proof of the Existence of Suitable Trees

Let us fix the positive integers k, d and l. We will not show the dependence on these parameters in the next definitions to simplify the notation, although d', E, C_r and C_r^* depend on them. We let $d' = d(1 - 1/(2^l - 1))$. For a (k, d)-vector $x = (x_0, \ldots, x_k)$ we define $E(x) = (\lfloor x_1/2 \rfloor, \lfloor x_2/2 \rfloor, \ldots, \lfloor x_k/2 \rfloor, \lfloor d'/2 \rfloor)$. We denote by $E^m(x)$ the vector obtained from x by m applications of the operation E. For $l \leq r \leq k$ we define $C_r(x)$ to be the $(k + 1)$-tuple starting with $r + 1 - l$ zero entries followed by $\lfloor x_j/(2^l - 1) \rfloor$ for $j = r + 1, \ldots, k$, followed by $\lfloor d'/2^{l-j} \rfloor$ for $j = 0, 1, \ldots, l - 1$ and let $C_r^*(x)$ be the $(k + 1)$-tuple starting with x_j for $j = l, l + 1, \ldots, r$ followed by $k - r + l$ zeros.

Note that for the following lemma to hold we could use d instead of d' in the definition of E and also in most places in the definition of C. The one place where we cannot do this is the entry $\lfloor d'/2^l \rfloor$ of $C_r(x)$ right after $\lfloor x_k/(2^l - 1) \rfloor$. If we used a higher value there, then one of the children of the root of the tree constructed in the proof below would have more than d leaves among its descendants in distance at most k. We use d' everywhere to be consistent and provide for the monotonicity as used in the proof of Theorem B.2.

LEMMA B.1. *Let k, d and l be positive integers and x a (k, d)-vector.*

(a) $E(x)$ is a (k, d)-vector. If $E(x)$ is constructible, then so is x.

(b) For $l \leq r \leq k$ both $C_r(x)$ and $C_r^*(x)$ are (k, d)-vectors. If both of these vectors are constructible and $|C_r^*(x)| < d/2^l$, then x is also constructible.

Proof. (a) We have $|E(x)| \leq |x|/2 + d'/2 < d$, so $E(x)$ is a (k, d)-vector. If there exists a $(k, d, E(x))$-tree, take two disjoint copies of such a tree and connect them with a new root vertex, whose children are the roots of these trees. The new binary tree so obtained is a (k, d, x)-tree.

(b) The sum of the first $k+1-l$ entries of $C_r(x)$ is at most $|x|/(2^l - 1) \leq d/(2^l - 1)$, the remaining fixed terms sum to less than $d' = d(1 - 1/(2^l - 1))$, so $|C_r(x)| \leq d$. We trivially have $|C_r^*(x)| \leq |x| \leq d$, so both $C_r(x)$ and $C_r^*(x)$ are (k, d)-vectors.

Let T be a $(k, d, C_r(x))$-tree and T^* a $(k, d, C_r^*(x))$-tree. Consider a full binary tree of depth l and attach T^* to one of the 2^l leaves of this tree and attach a separate copy of T to all remaining $2^l - 1$ leaves. This way we obtain a finite binary tree T' with all non-leaf vertices having exactly two children. To check condition (i) of the definition notice that no leaf of T' is in distance less than l from the root, leaves in distance $l \leq j \leq r$ are all in T^* and leaves in distance $r < j \leq k$ are all in the $2^l - 1$ copies of T. Condition (ii) we have to check only for vertices in distance $1 \leq j \leq l$ from the root. There are two types of vertices in distance j. One of them has 2^{l-j} copies of T below it, the other has one less and also T^*. It is a straight forward calculation to bound the number of leaves below and in distance at most k in either case. Instead of giving all the details of the computation we mention that the vertex of the first type in distance $j = 1$ of the root makes us use the specific value of $d' < d$ we use and the vertices of the second type make it necessary to assume a bound on $|C_r^*(x)|$.

Armed with the last two lemmas we are ready to prove the main result of this paper.

THEOREM B.2.

$$f_2(k) \leq \frac{2^{k+1}}{ek} + O\left(\frac{2^{k+1}}{k^{3/2}}\right)$$

Proof. We show that (k,d)-trees exist for large enough k and with $d = \lfloor 2^{k+1}/(ek) + 100 \cdot 2^{k+1}/k^{3/2} \rfloor$.

We set $l = \lfloor \log k/2 \rfloor$ and $s = 2l$. Here log denotes the binary logarithm, so $2^l \approx \sqrt{k}$. We define the (k,d)-vectors $x^{(t)} = (x_0^{(t)}, \ldots, x_k^{(t)})$ recursively. We start with $x^{(0)} = E^{k-s}(z)$, where z denotes the all zero (k,d)-vector. For $t \geq 0$ we define $x^{(t+1)} = E^{r_t - s - l}(C_{r_t}(x^{(t)}))$, where r_t is the is the smallest index in the range $s + l \leq r_t \leq k$ with $\sum_{j=0}^{r_t} x_j^{(t)}/2^j \geq 2^{-l}$. At this point we may consider the sequence of the (k,d)-vectors $x^{(t)}$ end whenever the weight of one of them falls below 2^{-l} and thus the definition of r_t does not make sense. But we will establish below that this never happens and the sequence is infinite.

Notice first, that we have $x_j^{(t)} = 0$ for all t and $0 \leq j \leq s$, while the entries $x_j^{(t)}$ for $s < j \leq k$ are all obtained from d' by repeated application of dividing by an integer (namely by $2^l - 1$ or by a power of 2) and taking lower integer part. Using the simple observation that $\lfloor \lfloor a \rfloor / j \rfloor = \lfloor a/j \rfloor$ if a is real and j is a positive integer we can ignore all roundings but the last. This way we can write each of these entries in the form $\left\lfloor \frac{d'}{2^i (2^l-1)^j} \right\rfloor = \left\lfloor \frac{d'}{2^{i+lj}} \left(1 + \frac{1}{2^l - 1}\right)^j \right\rfloor$ for some non-negative integers i and j. Using the values $\alpha = 1 + 1/(2^l - 1)$ and $q_t = r_t - s$ (this is the amount of "left shift" between x^t and x^{t+1}) we can give the exponents explicitly.

$$x_j^{(t)} = \left\lfloor \frac{d'}{2^{k+1-j}} \alpha^{c(t,j)} \right\rfloor \tag{B.1}$$

for all $s < j \leq k$ and all t, where $c(t,j)$ is the largest integer $0 \leq c \leq t$ satisfying $\sum_{i=t-c}^{t-1} q_i \leq k - j$. We define $c(t,j) = 0$ if $q_{t-1} > k - j$.

The formal inductive proof of this formula is a straight forward calculation. What really happens here (ignoring the rounding) is that each entry enters at the right end of the vector as $d'/2$ and is divided by 2 every time it moves one place to the left (application of E) but when it moves l places to the left through an application of C_{r_t} it is divided by $2^l - 1$ instead of 2^l so it gains a factor of α. The exponent $c(t,j)$ counts how many such factors are accumulated. If the "ancestor" of the entry $x_j^{(t)}$ was first introduced in $x^{(t')}$, then $c(t,j) = t - t'$.

We claim next that $c(t,j)$ and $x_j^{(t)}$ increases monotonously in t for each fixed $s < j \leq k$, while q_t decreases monotonously with t. We prove these statements by induction on t. We have $c(0,j) = 0$ for all j, so $c(1,j) \geq c(0,j)$. If $c(t+1,j) \geq c(t,j)$ for all j, then all entries of $x^{(t+1)}$ dominate the corresponding entries of $x^{(t)}$ by Equation (B.1). If $x_j^{(t+1)} \geq x_j^{(t)}$ for all j, then we have $r_{t+1} \leq r_t$ by the definition of these numbers, so we also have $q_{t+1} \leq q_t$. Finally, if q_i is decreasing for $i \leq t+1$, then by the definition of $c(i,j)$ we have $c(i+1,j) \geq c(i,j)$ for $i \leq t+1$.

The monotonicity just established also implies that the weight of $x^{(t)}$ is also increasing, so if the weight of $x^{(0)}$ is at least 2^{-l}, then so is the weight of all the other $x^{(t)}$, and thus the sequence is infinite. The weight of $x^{(0)}$ is

$$\begin{aligned}
&= \sum_{j=s+1}^{k} \frac{\left\lfloor \frac{d'}{2^{k+1-j}} \right\rfloor}{2^j} \\
&> \sum_{j=s+1}^{k} \frac{\frac{d'}{2^{k+1-j}} - 1}{2^j} \\
&> (k-s)\frac{d'}{2^{k+1}} - 2^{-s} \\
&> (k-s)\frac{1 - \frac{1}{2^l - 1}}{ek} - 2^{-s},
\end{aligned}$$

where the last inequality follows from $d > 2^{k+1}/(ek)$ and the last term tends to e^{-1} as k tends to infinity, so it is larger than 2^{-l} for large enough k.

We have just established that the sequence $x^{(t)}$ of (k,d)-vectors is infinite and coordinate-wise increasing. Since $|x^{(t)}| \leq d$ and it must strictly increase before the sequence stabilizes, the sequence must stabilize in at most d steps. So $x^{(t)} = x = (x_0, \ldots, x_k)$ for $t \geq d$. This implies that q_t also stabilizes with $q_t = q$ for $t \geq d$. Equation (B.1) as applied to $t > d + k$ simplifies to

$$x_j = \left\lfloor \frac{d'}{2^{k+1-j}} \alpha^{\lfloor (k-j)/q \rfloor} \right\rfloor. \tag{B.2}$$

Recall that $q = r_{t_0} - s$, and r_{t_0} is defined as the smallest index in the range $s + l \leq r \leq k$ with $\sum_{j=0}^{r} x_j/2^j \geq 2^{-l}$. Thus we have $q \geq l$. We claim that equality holds. Assume for contradiction that $q > l$ Then by the minimality of r_{t_0} we must have

$$\begin{aligned}
2^{-l} &> \sum_{j=0}^{s+q-1} \frac{x_j}{2^j} \\
&= \sum_{j=s+1}^{s+q-1} \frac{\left\lfloor \frac{d'}{2^{k+1-j}} \alpha^{\lfloor (k-j)/q \rfloor} \right\rfloor}{2^j} \\
&> \sum_{j=s+1}^{s+q-1} \frac{\frac{d'}{2^{k+1-j}} \alpha^{(k-j)/q-1} - 1}{2^j} \\
&> (q-1)\frac{d'}{2^{k+1}} \alpha^{k/q-4} - 2^{-2l}.
\end{aligned}$$

In the last inequality we used $s = 2l$ and that $\frac{j}{q} \leq \frac{s+q}{q} = 1 + \frac{s}{q} \leq 1 + \frac{2l}{l} = 3$. This inequality simplifies to

$$2^{-l}(1 + 2^{-l})\alpha^4 \frac{2^{k+1}}{d'} > (q-1)\alpha^{k/q}. \tag{B.3}$$

Simple calculus gives that the right hand side takes its minimum for $q \geq 2$ between $c-1$ and $c-2$ for $c = k \ln \alpha$ and this minimum is more than $(c-3)e$. Using $\alpha = 1 + 1/(2^l - 1) > e^{2^{-l}}$ we have $c \geq k/2^l$. So Inequality (B.3) yields

$$2^{-l}(1 + 2^{-l})\alpha^4 \frac{2^{k+1}}{d'} > \frac{ke}{2^l} - 3e.$$

Substituting our choice for d, d', α and l (as functions of k) shows that this last formula does not hold for large k. The contradiction proves $q = l$ as claimed.

We finish the proof of the theorem by establishing that the (k,d)-vectors $x^{(t)}$ are constructible. We prove this statement by downward induction for t. We start with $t = d$, where $x^{(d)} = x$ and by Equation (B.2) its weight is readily computable. Here we omit the details of this straight forward calculation but state that the weight is above 1. So by Lemma 2.2 x is constructible.

Now assume that $x^{(t+1)}$ is constructible. Recall that $x^{(t+1)} = E^{r_t - s - l}(C_{r_t}(x^{(t)}))$, so by (the repeated use of) Lemma B.1 part (a) $C_{r_t}(x^{(t)})$ is constructible. By part (b) of the same lemma $x^{(t)}$ is also constructible (and thus the inductive step is complete) if we can (i) show that $C^*_{r_t}(x^{(t)})$ is constructible and (ii) establish that $|C^*_{r_t}(x^{(t)})| \leq d/2^l$. For (i) we use the definition of r_t: $\sum_{j=0}^{r_t} x_j^{(t)}/2^j \geq 2^{-l}$. But the weight of $C^*_{r_t}(x^{(t)})$ is $\sum_{j=l}^{r_t} x_j^{(t)}/2^{j-l}$, so the contribution of each term with $j \geq l$ is multiplied by 2^l, while the missing terms $j < l$ contributed zero anyway as $l < s$. This shows that the weight of $C^*_{r_t}$ is at least 1 and therefore Lemma 2.2 proves (i). For (ii) we use monotonicity to see $r_t \leq r_0$ and Equation (B.1) to see that $r_0 \leq 5k/2^l$ for large enough k. Then we use monotonicity again to see $|C^*_{r_t}(x^{(t)})| = \sum_{j=s+1}^{r_t} x_j^{(t)} \leq \sum_{j=s+1}^{r_t} x_j$. Finally we use Equation (B.2) to see that this number is well below $d/2^l$ for large enough k. This finishes the proof of (ii) and hence the inductive proof that $x^{(t)}$ is constructible for every t.

As $x^{(0)} = E^{k-s}(z)$ is constructible Lemma B.1 (a) implies that z is also constructible, so there exists (k,d,z)-tree T. As z is the all zero vector, T must also be a (k,d)-tree.

References

[1] R. Aharoni and N. Linial, Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas, *J. Combin. Theory Ser. A* **43**, (1986), 196–204.

[2] P. Berman, M. Karpinski, and A. D. Scott, Approximation hardness and satisfiability of bounded occurrence instances of SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, **10** (022), 2003.

[3] S.A. Cook, The complexity of theorem-proving procedures, *Proc. 3rd Ann. ACM Symp. on Theory of Computing (STOC)* (1971), 151–158.

[4] G. Davydov, I. Davydova, and H. Kleine Büning, An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF, *Artif. Intell.* **23**, (1998), 229–245.

[5] O. Dubois, On the r,s-SAT satisfiability problem and a conjecture of Tovey, *Discrete Appl. Math.* **26** (1990), 51-60.

[6] P. Erdős and J.Spencer, Lopsided Lovász local lemma and Latin transversals *Discrete Appl. Math.* **30**, (1991), 151–154.

[7] U. Feige, A threshold of $\ln n$ for approximating set cover, *J. ACM* **45**(4), (1998), 634–652.

[8] C.M. Fortuin, P.N. Kasteleyn, J. Ginibre, Correlation inequalities for some partially ordered sets, *Comm. Math. Phys.* **22** (1971), 89–103.

[9] H. Gebauer, R. A. Moser, D. Scheder and E. Welzl, The Lovász Local Lemma and Satisfiability *Efficient Algorithms*, (2009), 30–54.

[10] H. Gebauer, Disproof of the Neighborhood Conjecture with Implications to SAT, *Proc. 17th Annual European Symposium on Algorithms (ESA)* (2009), LNCS 5757, 764–775.

[11] S. Hoory and S. Szeider. Computing unsatisfiable k-SAT instances with few occurrences per variable, *Theoretical Computer Science* **337**(1–3) (2005), 347–359,

[12] S. Hoory and S. Szeider, A note on unsatisfiable k-CNF formulas with few occurrences per variable, *SIAM J. Discrete Math* **20** (2), (2006), 523–528.

[13] H. Kleine Büning and X. Zhao, On the structure of some classes of minimal unsatisfiable formulas, *Discr. Appl. Math.* **130**(2), (2003), 185–207

[14] L. G. Kraft, A device for quantizing, grouping, coding amplitude modulated pulses, *M.S. thesis, Electrical Engineering Department, MIT* (1949).

[15] J. Kratochvíl, P. Savický and Z. Tuza, One more occurrence of variables makes satisfiability jump from trivial to NP-complete, *SIAM J. Comput.* **22** (1), (1993), 203–210.

[16] O. Kullmann, An application of matroid theory to the SAT problem, *Fifteenth Annual IEEE Conference on Computational Complexity* (2000), 116-124

[17] R.A. Moser, G. Tardos, A constructive proof of the general lovász local lemma, *J. ACM* **57**(2), (2010)

[18] C.H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *J. Comput. System Sci.* **43** (3), (1991), 425–440

[19] J. Radhakrishnan, A. Srinivasan, Improved bounds and algorithms for hypergraph 2-coloring, *Random Structures Algorithms* **16** (3), (2000), 4–32

[20] S. Roman, Coding and Information Theory, *Springer, New York* (1992).

[21] P. Savický and J. Sgall, DNF tautologies with a limited number of occurrences of every variable, *Theoret. Comput. Sci.* **238** (1–2), (2000), 495–498.

[22] D. Scheder, Unsatisfiable Linear CNF Formulas Are Large and Complex, *27th International Symposium on Theoretical Aspects of Computer Science (STACS)* (2010), 621–631

[23] S. Szeider, Homomorphisms of conjunctive normal forms, *Discr. Appl. Math.* **130**(2), (2003), 351–365

[24] C.A. Tovey, A simplified NP-complete satisfiability problem, *Discr. Appl. Math.* **8** (1), (1984), 85–89.