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, applied with assignement of random values according to counterintuitive probabilities, shows that our result is asymptotically best possible. 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 lower 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 construction of our unsatisfiable CNF-formulas is based on the binary tree approach of [16] and thus the constructed formulas are in the class MU(1) of minimal unsatisfiable formulas having one more clauses than variables. To obtain the asymptotically optimal binary trees we consider a continuous approximation of the problem, set up a differential equation and estimate its solution. The trees are then obtained through a discretization of this solution.

The binary trees constructed also give asymptotically precise answers for seemingly unrelated problems like the European Tenure Game introduced by Doerr [9] and the search problem with bounded number of consecutive lies, considered in a problem of the 2012 IMO contest. As yet another consequence we slightly improve the best known bounds on the maximum degree and maximum edge-degree of a k-uniform Maker's win hypergraph in the Neighborhood Conjecture of Beck.

*A preliminary version of this paper has appeared as an extended abstract in the Proceedings of SODA 2011

†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@math.fu-berlin.de. Research supported by DFG, project SZ 261/1-1.

§Simon Fraser University, Burnaby BC, Canada and Rényi Institute, Budapest, Hungary; mail: tardos@cs.sfu.ca. Supported by an NSERC grant, the Lendület project of the Hungarian Academy of Sciences and the Hungarian OTKA grants T-046234, AT-048826 and NK-62321

1 Introduction

The satisfiability of Boolean formulas is the archetypal 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 [6].

Papadimitriou and Yannakakis [27] have shown that MAX-k-SAT (finding the maximum number of simultaneously satisfiable clauses in an input k-CNF formula) 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 deciding satisfiability of a (k, s)-CNF formula is denoted by (k, s)-SAT, while finding the maximum number of simultaneously satisfiable clauses in such a formula is called MAX-(k, s)-SAT.

Tovey [35] 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 [10] showed that $(4, 6)$-SAT and $(5, 11)$-SAT are also NP-complete.

Kratochvíl, Savický, and Tuza [22] 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: MAX-(k, s)-SAT is also MAX-SNP-complete for every $s > f(k)$, $k \geq 2$ as was shown by Berman, Karpinski, and Scott [4, 5] (generalizing a result of Feige [13] who showed that MAX-$(3, 5)$-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. [15] 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 formula consisting of all possible k-clauses on only k variables.

In [15] a hardness jump is proved for the function l: for $k \geq 3$, deciding the satisfiability of k-CNF formulas with maximum neighborhood size at most $l(k) + 2$ is NP-complete.[1]

As observed by Kratochvíl, Savický and Tuza [22] 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 [30] showed that $f(k) = O\left(k^{0.74} \cdot \frac{2^k}{k}\right)$. This was improved by Hoory and Szeider [18] who came within a logarithmic factor: $f(k) = O\left(\log k \cdot \frac{2^k}{k}\right)$. Recently, Gebauer [16] 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 [16] gave $f(k) \leq \frac{63}{64} \cdot \frac{2^k}{k}$ for infinitely many k. The constant factor $\frac{63}{64}$ was clearly not the optimum rather the technical limit of the approach of [16]. 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 [22] and 63/64 of [16]. In fact several of the open problems of the survey of Gebauer, Moser, Scheder, and Welzl [15] are centered around the understanding of this question.

In our main theorem we settle these questions from [15] 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.

[1] In [15] the slightly more complicated formula $\max\{l(k) + 2, k + 3\}$ appeared for the maximum neighborhood size but this can be simplified to just its first term. For $k \geq 5$ this was already observed in [15] to follow from Equation 1.1. The statement follows from a slightly stronger form of the Local Lemma for $k = 4$ and from a case analysis for $k = 3$.

Theorem 1.1.

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

For the upper bound we use the fundamental binary tree approach of [16]. 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 counterintuitively: 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 [5] tailored to give good lower bounds on $f(k)$ for small values of k. In [5] the asymptotic behavior of the bound is not calculated, since the authors did not believe in its optimality. In Section 6 we reproduce a simple argument giving the asymptotics, because the proof of [5] contains a couple of inaccuracies obscuring the required unusual choice of the probabilities.

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 [11] 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 many variables receiving the *less frequent sign* are those that intersect more than the average number of other clauses, so for the use of the Lopsided Local Lemma 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 [24] we now have that not only every (k, s)-CNF formula for $s = \left\lfloor \frac{2^{k+1}}{e(k+1)} \right\rfloor$ 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 is not likely to be able to 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.

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

Theorem 1.1 and Theorem 1.2 are another instances which show the tightness of the Lovász Local Lemma. The first such example was given by Shearer [32].

1.1 (k, d)-trees

The substantial part of the proofs of Theorems 1.1 and 1.2—the upper bounds—as well as all our further results depend on the construction of certain binary trees.

Throughout the paper, whenever mentioning *binary tree*s we always mean *proper binary trees*, that is, rooted trees where every non-leaf node has exactly two children. We say that a leaf w of a tree T is *k-close* from a vertex $v \in V(T)$ if v is an ancestor of w and w has distance at most k from v. When k is clear from the context we say *w is visible from v* or *v sees w*.

The concept of (k, d)-trees, introduced by Gebauer [16], will be our main tool in this paper. We call a (proper) binary tree T a *(k, d)-tree*[2] if

(i) every leaf has depth at least k and

(ii) for every node u of T the number of k-close leaves from u is at most d.

For a fixed k, we are interested in how low one can make d in a (k, d)-tree. Essentially all our main results will be consequences of the construction of (k, d)-trees with relatively small d. We introduce $f_{\text{tree}}(k)$ to stand for the smallest integer such that a $(k, f_{\text{tree}}(k))$-tree exists and determine $f_{\text{tree}}(k)$ asymptotically.[3]

Theorem 1.3.

$$\left\lfloor \frac{2^{k+1}}{e(k+1)} \right\rfloor \leq f_{\text{tree}}(k) = \left(\frac{2}{e} + O\left(\frac{1}{\sqrt{k}}\right)\right) \frac{2^k}{k}.$$

The construction of the trees providing the upper bound constitutes a large portion of our paper. We devote quite a bit of effort (the entire Section 4) to describe the key informal ideas of the proof. That is, we formulate the construction process in a continuous setting and study its progress with the help of a continuous two-variable function $F(t, x)$ defined by a certain differential equation. It can be shown

[2]To simplify our statements we shifted the parameter k in the definition compared to [16]: what is called a (k, d)-tree there is a $(k-1, d)$-tree in our new terminology.

[3]Note that in [15] the function f_{tree} is defined to be one less than our definition (the maximum d such that *no* (k, d)-tree exist). This might be more appropriate if f_{tree} is only considered for its implications to SAT, but in view of the many other applications we feel that our definition is more natural.

that the integral $\int F(t,x)dx$ being large for some t corresponds to the construction process terminating with the desired (k,d)-tree. Even though our treatment in Section 4 will be highly informal (with simplifying assumptions and approximations), we find it important for a couple of reasons. On the one hand, it provides the true motivation behind our formal discrete construction and illustrates convincingly *why* this construction (treated formally in Section 5) *should* work. Furthermore the continuous function $F(t,x)$, defined via the differential equation, is also helpful in studying the *size* of the constructed formulas. This connection will be indicated in Section 7.

1.2 Formulas and 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 (cf. [1]), all formulas in MU have more clauses than 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], [7], [19], [23], [34]). Hoory and Szeider [17] 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. (For $k \leq 4$ it is a doable exercise to see that $f(k) = f_1(k) = k$.)

In [16] (k,d)-trees were introduced to construct unsatisfiable CNF formulas and upper bound the functions f and l. Since these formulas also reside in the class MU(1), we can also use them to upper bound f_1. Most of the content of the following statement appears already in Lemma 1.6 of [16].

Theorem 1.4 ([16]). (a) $f(k) \leq f_1(k) < f_{\text{tree}}(k)$

(b) $l(k) < k f_{\text{tree}}(k-1)$

For completeness and since part (a) is stated in a slightly weaker form in [16], we include the short proof in Section 2.1.

It is an interesting open problem whether $f(k) = f_1(k)$ for every k. Theorems 1.1, 1.3, and 1.4 imply that $f(k)$ and $f_1(k)$ are equal asymptotically: $f(k) = (1 + o(1))f_1(k)$. More precisely, we have the following.

Corollary 1.5. $f_1(k) = \left(\frac{2}{e} + O\left(\frac{1}{\sqrt{k}}\right)\right)\frac{2^k}{k}$.

Scheder [31] 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).

1.3 The Neighborhood Conjecture

A *hypergraph* is a pair $(V,\mathcal{F})$, where V is a finite set whose elements are called *vertices* and $\mathcal{F}$ is a family of subsets of V, called *hyperedges*. A hypergraph is *k-uniform* if every hyperedge contains exactly k vertices.

A hypergraph is 2-colorable if there is a coloring of the vertices with red and blue such that no edge is *monochromatic*.

A standard application of the first moment method says that for any k-uniform hypergraph $\mathcal{F}$, we have

$$|\mathcal{F}| < 2^{k-1} \Rightarrow \mathcal{F} \text{ is 2-colorable.}$$

An important generalization of this implication was given by Erdős and Selfridge [12], which also initiated the derandomization method of conditional expectations. Erdős and Selfridge formulated their result in the context of positional games. Given a k-uniform hypergraph $\mathcal{F}$ on vertex set V, players Maker and Breaker take turns in claiming one previously unclaimed element of V, with Maker going first. Maker wins if he claims all vertices of some hyperedge of $\mathcal{F}$, otherwise Breaker wins.

Since this is a finite perfect information game and the players have complementary goals, either Maker has a winning strategy (that is, the description of the vertex to be claimed next by Maker in any imaginable game scenario, such that at the end he wins, no matter how Breaker plays) or Breaker has a winning strategy. Which of this is the case depends solely on $\mathcal{F}$, hence it makes sense to call the hypergraph $\mathcal{F}$ *Maker's win* or *Breaker's win*, respectively.

The crucial connection between Maker/Breaker games to 2-colorability is the following:

$$\mathcal{F} \text{ is Breaker's win} \Rightarrow \mathcal{F} \text{ is 2-colorable.}$$

Indeed, if both players use Breaker's winning strategy,[4] by the end of the game they both win as Breakers and hence create a 2-colored vertex set V, where both colors are represented in each hyperedge — a proper 2-coloring of $\mathcal{F}$. Hence, the following theorem of Erdős and Selfridge [12] is a generalization of the first moment result

[4]The second player can use this strategy directly, but for the player going first one has to slightly modify it: start with an arbitrary move and then use the strategy, always maintaining to own one extra element of the board. It is easy to see that owning an extra element cannot be disadvantageous, so there exists a Breaker-win strategy for the first player, too.

above.

$$|\mathcal{F}| < 2^{k-1} \Rightarrow \mathcal{F} \text{ is a Breaker's win.}$$

As the Erdős-Selfridge Theorem can be considered the game-theoretic first moment method, the Neighborhood Conjecture of József Beck (to be stated below) would be the game-theoretic Local Lemma. Unlike the first moment method, the Local Lemma guarantees the 2-colorability of hypergraphs based one some local condition like the maximum degree of a vertex or an edge of the hypergraph. The *degree* $d(v)$ of a vertex $v \in V(\mathcal{F})$ is the number of hyperedges of $\mathcal{F}$ containing v and the *maximum degree* $\Delta(\mathcal{F})$ of $\mathcal{F}$ is the maximum degree of its vertices. The *neighborhood* $N(e)$ of a hyperedge e is the set of hyperedges of $\mathcal{F}$ which intersect e, excluding e itself, and the *maximum neighborhood size* $\Delta(L(\mathcal{F}))$ of $\mathcal{F}$ is the maximum of $|N(e)|$ where e runs over all hyperedges of $\mathcal{F}$. ($L(\mathcal{F})$ denotes the *line-graph* of $\mathcal{F}$). Simple applications of the Local Lemma show that,

$$\Delta(L(\mathcal{F})) \leq \frac{2^{k-1}}{e} - 1 \Rightarrow \mathcal{F} \text{ is 2-colorable,} \tag{1.3}$$

$$\Delta(\mathcal{F}) \leq \frac{2^{k-1}}{ek} \Rightarrow \mathcal{F} \text{ is 2-colorable.} \tag{1.4}$$

The Neighborhood Conjecture in its strongest form [3, Open Problem 9.1] was suggesting the far-reaching generalization that already when $\Delta(L(\mathcal{F})) < 2^{k-1} - 1$, $\mathcal{F}$ should be a Breaker's win. This was motivated by the construction of Erdős and Selfridge of a k-uniform Maker's win hypergraph $\mathcal{G}$ with $|\mathcal{G}| = 2^{k-1}$, showing the tightness of their theorem. The maximum neighborhood size of $\mathcal{G}$ is $2^{k-1} - 1$ (every pair of edges intersect) and no better construction was known until Gebauer [16] disproved the conjecture using her (k, d)-tree approach. She constructed Maker's win hypergraphs $\mathcal{F}$ and $\mathcal{H}$ with $\Delta(L(\mathcal{F})) = 0.75 \cdot 2^{k-1}$ and $\Delta(\mathcal{H}) \leq \frac{63}{128}\frac{2^k}{k}$, respectively. Our (k, d)-trees from Theorem 1.1 will imply the following somewhat improved bounds.

Theorem 1.6. *For every integer $k \geq 3$ there exists a Maker's win k-uniform hypergraph $\mathcal{H}$ such that*

(*i*) $\Delta(\mathcal{H}) \leq 2f_{\text{tree}}(k-2) = \left(1 + O\left(\frac{1}{\sqrt{k}}\right)\right)\frac{2^k}{ek}$ *and*

(*ii*) $\Delta(L(\mathcal{H})) = (k-1)f_{\text{tree}}(k-2) = \left(1 + O\left(\frac{1}{\sqrt{k}}\right)\right)\frac{2^{k-1}}{e}$.

Note that the bound in part (*ii*) asymptotically coincides with the one given by the Local Lemma in (1.3) for 2-colorability, while part (*i*) is still a factor 2 away from (1.4). Note furthermore that the bounds in (1.3) and (1.4) are not optimal. More elaborate methods of Radhakrishnan and Srinivasan [28] show that for a small enough constant $c > 0$ any k-uniform hypergraph $\mathcal{F}$ with $\Delta(L(\mathcal{F})) \leq$

$c2^k\sqrt{k/\log k}$) is, in fact, 2-colorable. Part (*ii*) of Theorem 1.6 establishes that no game-theoretic generalization of such a stronger 2-coloring condition exists (*beyond* the Local Lemma bound).

Determining whether the bounds in (1.3) and (1.4) have game-theoretic generalizations is still open. In [3] various weaker versions of the Neighborhood Conjecture are stated, maybe the most fundamental one is whether there exists a constant $\epsilon > 0$, such that every k-uniform hypergraph with $\Delta(\mathcal{F}) < (1+\epsilon)^k$ is a Breaker's win. However, the futility to give lower bounds makes even the following question interesting.

Problem 1.7. *Is every k-uniform hypergraph $\mathcal{F}$ with $\Delta(\mathcal{F}) = \lfloor \frac{k}{2} \rfloor + 1$ a Breaker's win for large enough k?*

It is known that Hall's Theorem does give Breaker a winning *pairing* strategy when the bound on the maximum degree is just $\lfloor \frac{k}{2} \rfloor$. The hypergraph dual of the Petersen graph is a Maker's win 3-uniform hypergraph of maximum degree 2 and Knox [20] constructed 4-uniform Maker's win hypergraphs with maximum degree 3. However Problem 1.7 is open for $k > 4$. It seems likely that solving it for general k will require a new idea and hence might lead to a more significant progress on the Neighborhood Conjecture itself.

1.4 European Tenure Game

The (usual) Tenure Game (introduced by Joel Spencer [33]) is a perfect information game between two players: the (good) chairman of the department, and the (vicious) dean of the school. The department has d non-tenured faculty and the goal of the chairman is to promote (at least) one of them to tenure, the dean tries to prevent this. Each non-tenured faculty is at one of k pre-tenured rungs, denoted by the integers $1, \ldots, k$. A non-tenured faculty becomes tenured if she has rung k and is promoted. The procedure of the game is the following. Once each year, the chairman proposes to the dean a subset S of the non-tenured faculty to be promoted by one rung. The dean has two choices: either he accepts the suggestion of the chairman, promotes everybody in S by one rung and fires everybody else, or he does the complete opposite of the chairman's proposal (also typical dean-behavior): fires everybody in S and promotes everybody else by one rung. This game obviously ends after at most k years. The game analysis is very simple, see [33].

In the European Tenure Game (introduced by Benjamin Doerr [9]) the rules are modified, so the non-promoted part of the non-tenured faculty is not fired, rather demoted back to rung 1. An equivalent, but perhaps more realistic scenario is that the non-promoted faculty is fired but the department hires new people at the lowest rung to fill the tenure-track positions vacated by those fired. For simplicity we assume that all non-tenured faculty are at the lowest rung in the beginning of the

game and would like to know what combinations of k and d allow for the chairman to eventually give tenure for somebody when playing against any (vicious and clever) dean. For fixed d let v_d stand for the largest number k of rungs such that this is possible.

Doerr [9] showed that

$$\lfloor \log d + \log\log d + o(1) \rfloor \leq v_d \leq \lfloor \log d + \log\log d + 1.73 + o(1) \rfloor.$$

It turns out that the game is equivalent to (k, d)-trees, hence using Theorem 1.3 we can give a precise answer, even in the additive constant, which turns out be $\log_2 e - 1 \approx 0.442695$.

Theorem 1.8. *The chairman wins the European Tenure Game with d faculty and k rungs if and only if there exists a (k, d)-tree. In particular, $v_d = \max\{k \mid f_{\text{tree}}(k) \leq d\}$ and we have*

$$v_d = \lfloor \log d + \log\log d + \log e - 1 + o(1) \rfloor.$$

1.5 Searching with lies

In a liar game Player A thinks of a member x of an agreed upon N element set H and Player B tries to figure it out by Yes/No questions of the sort "Is $x \in S$?", where S is a subset of H picked by B. This is not difficult if A is always required to tell the truth, but usually A is allowed to lie. However for B to have a chance to be successful, the lies of A have to come in some controlled fashion. The most prominent of these restrictions allows A to lie at most k times and asks for the smallest number $q(N, k)$ of questions that allows B to figure out the answer. This is also called Ulam's problem for binary search with k lies. For an exhaustive description of various other lie-controls, see the survey of Pelc [25].

One of the problems in the 2012 International Mathematics Olympiad was a variant of the liar game. Instead of limiting the total number of lies, in the IMO problem the number of consecutive lies was limited. This fits into the framework of Section 5.1.3 in Pelc's survey [25]. This restriction on the lies is not enough for B to find the value x with certainty, but he is able to narrow down the set of possibilities. The IMO problem asks for certain estimates on how small B can eventually make this set. This problem was also the topic of the Minipolymath4 project research thread [26].

It turns out that this question can also be expressed in terms of existence of (k, d)-trees.

Theorem 1.9. *Let $N > d$ and k be positive integers. Assume A and B play the guessing game in which A thinks of an element x of an agreed upon set H of size N and then answers an arbitrary number of B's questions of the form "Is $x \in S$?". Assume further that A is allowed to lie, but never to k consecutive questions. Then*

B can guarantee to narrow the number of possibilities for x with his questions to at most d distinct values if and only if a $(k, d+1)$-tree exists, that is, if $d < f_{\text{tree}}(k)$.

1.6 Notation

Throughout this paper, log denotes the binary logarithm. We use $\mathbb{N}$ to denote the set of natural numbers including 0.

As we have mentioned, by a *binary tree* we always mean a rooted tree where every node has either two or no children. The *height* of a binary tree is defined as the largest root-leaf distance. The *depth* of a vertex is its distance from the root.

1.7 Organization of this paper

In Section 2 we derive all applications of the upper bound in Theorem 1.3. This includes the upper bounds of Theorems 1.1 and 1.2 through proving Theorem 1.4, as well as the proofs of Theorems 1.6, 1.8 and 1.9. In Section 3 we give the basic definitions and simple propositions required for the proof of Theorem 1.3. In Section 4 we sketch the main informal ideas behind our tree-construction including a rough description of our approach, and how it translates to solving a differential equation. This is in the background of our actual formal constructions for the proof of Theorems 1.3, which then can be given completely discretely. The formal construction is the subject of Section 5. The lower bound of Theorem 1.1 (implying the lower bound for Theorem 1.3) is shown in Section 6. In Section 7 we give an outlook and pose some open problems.

2 Applying (k, d)-trees

In this section we apply (k, d)-trees and the upper bound in Theorem 1.3 to prove the upper bounds of Theorem 1.1 and 1.2, as well as Theorems 1.4, 1.6, 1.8, and 1.9. Some of these connections, sometimes in disguise, were already pointed out in [16].

2.1 Formulas

In this subsection we give a proof of Theorem 1.4, which, together with the upper bound in Theorem 1.3, readily implies the upper bounds in Theorems 1.1 and 1.2.

For every binary tree T (recall that we only consider proper binary trees) which has all of its leaves at depth at least k, one can construct a k-CNF formula $F_k(T)$ 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, which is the conjunction of the first k labels encountered when walking along the path from w towards the root

(including the one at w). The disjunction of the clauses C_w for all leaves w of T constitutes the formula $F_k(T)$.

Observation 2.1. *$F_k(T)$ is unsatisfiable.*

Proof. 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 `false` by α. Then C_w is violated by α. □

Proof of Theorem 1.4. Consider a $(k, f_{\text{tree}}(k))$-tree T and the corresponding k-CNF formula $F = F_k(T)$. F is unsatisfiable by Observation 2.1. The variables of this formula are the vertex-labels of T. The variable x_v corresponding to a vertex v appears in the clause C_w if and only if w is k-close from v. Thus each variable appears at most $f_{\text{tree}}(k)$ times. This makes F an unsatisfiable $(k, f_{\text{tree}}(k))$-CNF, proving $f(k) < f_{\text{tree}}(k)$. If F is in MU(1), then we further have $f_1(k) < f_{\text{tree}}(k)$. The number of clauses in F is the number of leaves in T, the number of variables in F is the number of non-leaf vertices in T, so we have exactly one more clauses than variables. But F is not a *minimal* unsatisfiable formula in general. Fortunately it is one, if each variable appears in F both in negated and in non-negated forms, see [7]. This will be the case if we pick T to be a $(k, f_{\text{tree}}(k))$-tree which is minimal with respect to containment. Indeed, if a literal associated to a vertex v does not appear in any of the clauses, then the subtree of T rooted at v is a $(k, f_{\text{tree}}(k))$-tree. This finishes the proof of part (a) of the theorem.

Clearly, the neighborhood of any clause in a (k, d)-CNF formula is of size at most $k(d-1)$, but this bound is too rough to prove part (b) with. We start by picking a $(k-1, f_{\text{tree}}(k-1))$-tree T' with each leaf having depth at least k. Such a tree can be constructed by taking two copies of an arbitrary $(k-1, f_{\text{tree}}(k-1))$-tree and connecting their roots to a new root vertex. Now $F' = F_k(T')$ is an unsatisfiable k-CNF by Observation 2.1. The advantage of this construction is that now each *literal* appears at most $f_{\text{tree}}(k-1)$ times (as opposed to only having a bound on the multiplicity of each *variable* in T). Note that if two distinct clauses in F' share a common variable, then there is a variable that appears negated in one of them and non-negated in the other. This implies that every clause intersects at most $kf_{\text{tree}}(k-1)$ other clauses as needed. □

Note that F' in the proof above can also be chosen to be in MU(1). This is the case if T' is obtained from a minimal $(k-1, f_{\text{tree}}(k-1))$-tree by doubling it.

Now Theorem 1.3 together with part (a) of Theorem 1.4 implies the upper bound in Theorem 1.1, while together with part (b) of Theorem 1.4 it implies the upper bound in Theorem 1.2. We also have an implication in the reverse direction: the lower bound of Theorem 1.1 implies the lower bound of Theorem 1.3 using Theorem 1.4(a).

2.2 The Neighborhood Conjecture

In this subsection we prove Theorem 1.6.

We start with a few simple observations. One can associate a hypergraph $\mathcal{H}(F)$ to any CNF formula F by taking all the literals in F as vertices, and considering the clauses of F (or rather the set of literals which the clause is the disjunction of) as the hyperedges.

Observation 2.2. *If F is unsatisfiable, then $\mathcal{H}(F)$ is Maker's win.*

Proof. The pairing strategy where Maker picks one of the two literals corresponding to any variable guarantees a victory for Maker. Maker can even let Breaker start the game (as we have previously argued, going second cannot help a player) and then pick the pair of any literal picked by Breaker. At the end, consider the evaluation of the variables setting all the literals that Breaker has to `true`. As F is unsatisfiable this evaluation falsifies F and therefore it violates one of the clauses, giving Maker his win at the hyperedge corresponding to this clause. □

As we have seen in the proof of Theorem 1.4 there exist a $(k-1, f_{\text{tree}}(k-1))$-tree T' with all leaves at depth at least k and $F_k(T')$ is an unsatisfiable k-CNF where all literals appear in at most $f_{\text{tree}}(k-1)$ clauses. By Observation 2.2 this makes $\mathcal{H}(F_k(T'))$ a Maker's win k-uniform hypergraph with maximum degree at most $f_{\text{tree}}(k-1)$. It is easy to see that $f_{\text{tree}}(k-1) \leq 2f_{\text{tree}}(k-2)$, so this bound is better than the one claimed in part (i) of Corollary 1.6. But in order to get part (ii) as well we have to take a slightly different approach.

Proof of Theorem 1.6. Let T be a $(k-2, f_{\text{tree}}(k-2))$-tree, with all leaves in depth at least k. One can build such a tree from four copies of an arbitrary $(k-2, f_{\text{tree}}(k-2))$-tree. In each clause of $F = F_k(T)$ the literals are associated to vertices in T. We distinguish the *leading* literal in each clause to be the one associated to the vertex closest to the root. A clause C_w contains the literal associated to a vertex v in *non-leading* position exactly when the leaf w is $(k-2)$-close from v. Thus any literal appears in at most $f_{\text{tree}}(k-2)$ clauses in non-leading position. A literal associated to a leaf of T appears in a single clause of F. Any other literal ℓ appears in at most $2f_{\text{tree}}(k-2)$ clauses as any clause containing ℓ must also contain the literal associated to one of the two children of the vertex of ℓ and these are not in leading position. This makes $\mathcal{H} = \mathcal{H}(F)$ a k-uniform hypergraph with maximum degree $\Delta(\mathcal{H}) \leq 2f_{\text{tree}}(k-2)$. Observations 2.1 and 2.2 ensure that $\mathcal{H}$ is a Maker's win hypergraph and this concludes the proof of part (i).

As we have observed earlier each pair of distinct intersecting clauses of F share a variable that appears negated in one of them and in non-negated form in the other. Observe further that when two distinct clauses share a literal they also share a variable that appears in opposite form as non-leading literals in the two clauses.

A clause C has $k-1$ non-leading literals and the opposite form of each is contained in non-leading position in at most $f_{\text{tree}}(k-2)$ clauses. This gives us the bound stated in part (ii) on the number of clauses sharing a literal with C. □

2.3 The European Tenure Game

In this subsection we prove Theorem 1.8, namely the connection to (k,d)-trees. The formula estimating v_d will then follow from Theorem 1.3. We start with a special labeling of (k,d)-trees.

Proposition 2.3. *Let T be a (k,d)-tree and let L be its set of leaves. There exists a labeling $i : L \to \{1,\dots,d\}$ such that for every vertex $v \in V(T)$ all the k-close leaves from v have distinct labels.*

Proof. We define the labels of the leaves one by one. We process the vertices of T according to a Breadth First Search, starting at the root. When processing a vertex v we label the still unlabeled leaves that are k-close from v making sure they all receive different labels. This is possible, because the ones already labeled are all k-close from the parent of v, so must have received different labels. We have enough labels left because the total number of leaves visible from v is at most d. After processing all vertices of T our labeling is complete and satisfies the requirement. □

Proof of Theorem 1.8. Suppose first that there is a (k,d)-tree T and let us give a winning strategy to the chairman. We start by labeling the leaves of T with the d non-tenured faculty, according to Proposition 2.3. The chairman is placed on the root of T and during the game he will move along a path from the root to one of the leaves, in each round proceeding to one of the children of its current position. To which leaf he arrives at depends on the answers of the dean. When standing at a non-leaf vertex v with left-child v_1 and right-child v_2, the chairman proposes to promote the subset S of the faculty containing the labels of the leaves that are $(k-1)$-close from v_1. If the dean accepts his proposal he moves to v_1, otherwise he moves to v_2. The game stops when a leaf is reached. We claim that the label P of this leaf is promoted to tenure, hence the chairman has won.

Note that by part (i) of the definition of a (k,d)-tree the game lasts for at least k rounds. We will show that in each of the last k rounds P was promoted. Indeed, if the chairman moved to the left in one of these rounds then P was proposed for promotion and the dean accepted it. However, if the chairman moved to the right, then P could not be proposed for promotion by the condition of the labeling of Proposition 2.3, but the dean reversed the proposal and hence P was promoted in these rounds as well.

For the other direction, we are given a winning strategy for the chairman. This strategy specifies at any point of the game which subset of the faculty the chairman

should propose for promotion unless a member of the faculty is already tenured, at which time the game stops. In building the game tree we disregard the subsets but pay close attention to when the game stops. In particular, each vertex corresponds to a position of the game with the root corresponding to the initial position. If a vertex v corresponds to a position where the game stops we make v a leaf and label it with one of the faculty members that has just been tenured. Otherwise v has two children, one corresponding to each of the two possible answers of the dean.

Clearly, this is a (proper) binary tree. We claim that it is a (k, d)-tree. Note that in order for somebody get tenured she has to be promoted in k consecutive rounds, so all leaves are at depth at least k, as required.

To prove that no vertex sees more than d leaves we prove that all leaves k-close from the same vertex have distinct labels. Indeed, if a leaf w is k-close from a vertex v and w is labeled by faculty Frank, then Frank had to be promoted in all rounds in the game from the position corresponding to v till the position corresponding to w. But Frank is promoted in exactly one of the two cases depending on the dean's answer, so this condition determines a unique path in our tree from v making w the unique leaf labeled Frank that is k-close from v. □

2.4 Searching with lies

Proof of Theorem 1.9. The first observation is that the game with parameters $N > d$ and k is won by B if and only if B wins with parameters $N' = d + 1$, d and k.

One direction is trivial, if N is decreased it cannot hurt B's chances. For the other direction assume B has a winning strategy for the $N' = d + 1$ case. This means that given $d + 1$ possible values for x he can eliminate one of them. Now if he has more possibilities for the value of x he can concentrate on $d + 1$ of them and ask questions till he eliminates one of these possibilities. He can repeat this process, always reducing the number of possible values to x till this number goes below $d + 1$ — but by then he has won.

We can therefore concentrate to the case $N = d + 1$. We claim that this is equivalent to the European tenure game with k non-tenured rungs and $d + 1$ non-tenured faculty, thus Theorem 1.9 follows from Theorem 1.8. Indeed, A corresponds to the dean, B corresponds to the chairman, the $d+1$ possible values of x correspond to the non-tenured faculty members. B asking if $x \in S$ holds corresponds to the chairman proposing the set S for promotion, a "no" answer corresponds to the dean accepting the proposal, while a "yes" answer corresponds to the dean reversing it. At any given time the rung of the faculty member corresponding to value $v \in [d+1]$ in the liar game is $i + 1$, where i is the largest value such that the last i questions of B were answered by A in a way that would be false if $x = v$. Thus a win for the chairman (tenuring a faculty member, i.e. promoting him k consecutive times) exactly corresponds to A answering k consecutive times in such a way that would

be false if $x = v$. This makes $x = v$ impossible according to the rules of the liar game, so B can eliminate v and win. □

3 Formal Definitions and Basic Statements

3.1 Vectors and constructibility

Given a node w in a tree T, it is important to count the leaf-descendants of w in distance i for $i \leq d$. We say that a non-negative integer vector $(x_0, x_1, \ldots, x_k)$ is a *leaf-vector* for w if w has at most x_i leaf-descendants in distance i for each $0 \leq i \leq k$. E.g., the vector $(1, 0, \ldots, 0)$ is a leaf-vector for any leaf, while for the root w of a full binary tree of height $l \leq k$ we have $(0, 0, \ldots, 0, 2^l, 0, \ldots, 0)$ as its smallest leaf-vector. We set $|\vec{x}| := \sum_{i=0}^{k} x_i$. By definition, every node w of a (k, d)-tree has a leaf-vector $\vec{x}$ with $|\vec{x}| \leq d$.

For some vector $\vec{x} \in \mathbb{N}^{k+1}$ we define a $(k, d, \vec{x})$-*tree* to be a tree where

(i) $\vec{x}$ is a leaf-vector for the root, and

(ii) each vertex has at most d leaves that are k-close.

For example, a tree consisting of a parent with two children is a $(k, d, (0, 2, 0 \ldots, 0))$-tree for any $k \geq 1$ and $d \geq 2$.

We say that a vector $\vec{x} \in \mathbb{N}^{k+1}$ 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 (k, d)-constructible as long as $2^l \leq d$.

Observation 3.1. *There exists a (k, d)-tree if and only if the vector $(0, \ldots, 0, d)$ is (k, d)-constructible.*

Proof. The vector $(0, \ldots, 0, d)$ is (k, d)-constructible if and only if a $(k, d, (0, \ldots, 0, d))$-tree exists. It is easy to see that the definitions of (k, d)-tree and $(k, d, (0, \ldots, 0, d))$-tree are equivalent. Indeed, condition (ii) is literally the same for both, while condition (i) states in both cases that there is no leaf $k - 1$-close to the root. The only difference is that condition (i) for a $(k, d, (0, \ldots, 0, d)$-tree also states that there are at most d leaves k-close to the root, but this also follows from (ii). □

The next observation will be our main tool to capture how leaf-vectors change as we pass from a parent to its children.

Observation 3.2. *If $x' = (x'_0, x'_1, \ldots, x'_k)$ and $x'' = (x''_0, x''_1, \ldots, x''_k)$ are (k, d)-constructible and $|x| \leq d$ for $x = (0, x'_0 + x''_0, x'_1 + x''_1, \ldots, x'_{k-1} + x''_{k-1})$, then x is also (k, d)-constructible.*

Proof. Let T' be a (k, d, x')-tree with root r' and T'' a (k, d, x'')-tree with root r''. We create the tree T by taking a new root vertex r and attach it to r' and r''. This tree is a (k, d, x)-tree. Indeed, the leaf-descendants of r at distance i are exactly leaf-descendants of either r' or r'' at distance $i-1$, hence x is a leaf-vector for r. We also have to check that no vertex has more than d leaves k-close. This holds for the vertices of T' and T'' and is ensured by our assumption $|x| \leq d$ for the root r. □

For a vector $\vec{x} = (x_0, \ldots, x_k)$ we define its *weight* $w(\vec{x})$ to be $\sum_{i=0}^{k} x_i/2^i$. The next lemma gives a useful sufficient condition for the constructibility of a vector.

Lemma 3.3. *Let $\vec{x} \in \mathbb{N}^{k+1}$ with $|\vec{x}| \leq d$. If $w(\vec{x}) \geq 1$ then $\vec{x}$ is (k, d)-constructible.*

We note that Lemma 3.3 is a reformulation of Kraft's inequality. For completeness we give a direct proof here.

Proof of Lemma 3.3: We build a binary tree starting with the root and adding the levels one by one. As long as $\sum_{j=0}^{i} \frac{x_j}{2^j} < 1$, we select a set of x_i vertices from the vertices on level i and let them be leaves. We construct the $(i+1)$th level by adding two children to the remaining $2^i(1 - \sum_{j=0}^{i} \frac{x_j}{2^j})$ vertices on level i. At the first level $\ell \leq k$ where $\sum_{j=0}^{\ell} \frac{x_j}{2^j} \geq 1$ we mark all vertices as leaves and stop the construction of the tree. The total number of leaves is at most $\sum_{j=0}^{\ell} x_j \leq |\vec{x}| \leq d$ and the number of leaves at distance j from the root is at most x_j, so the constructed tree is a $(k, d, \vec{x})$-tree. □

The main result of [16] is the construction of (k, d)-trees with $d = \Theta(\frac{2^k}{k})$. This argument is now streamlined via Lemma 3.3. Indeed, the (k, d)-constructibility of the vector $\vec{v} = (0, \ldots, 0, 1, 2, 4, \ldots, 2^s)$ is an immediate consequence of Lemma 3.3, provided $1 \leq w(\vec{v}) = \sum_{i=k-s}^{k} 2^{i-k+s}/2^i = (s+1)2^{s-k}$ and $d \geq |\vec{v}| = \sum_{i=0}^{k} v_i = 2^{s+1} - 1$. Setting $s = k - \lfloor \log(k - \log k) \rfloor$ and $d = 2^{s+1}$ allows both inequalities to hold. Then by repeated application of Observation 3.2 with $x' = x''$ we obtain the constructibility of $(0, \ldots, 0, 2, 4 \ldots, 2^s)$, $(0, \ldots, 0, 4, \ldots, 2^s)$, etc., and finally the constructibility of $(0, \ldots, 0, 2^s)$. This directly implies the existence of a (k, d)-tree by Observation 3.1, where $d = 2^{s+1}$. Note that $d = (2 + o(1))\frac{2^k}{k}$ for infinitely many k including $k = 2^t + t + 1$ for any t. Figure 1 shows an illustration.

Proof strategy. In Section 5 we will construct our (k, d)-tree starting with the root, from top to bottom. When considering some vertex w it will be assigned a leaf-vector $\vec{\ell}_w$. At this moment w itself is a leaf in the partly constructed tree, so one should consider $\vec{\ell}_w$ just as a promise: for each $i = 0, 1 \ldots, k$ the vertex w promises to have at most $(\vec{\ell}_w)_i$ leaf-descendants at distance i when the (k, d)-tree is fully constructed.

We start with the root with a leaf-vector $(0, \ldots, 0, d)$. At each step we have to consider the vertices v that are currently leaves but promise not to be leaves: i.e.,

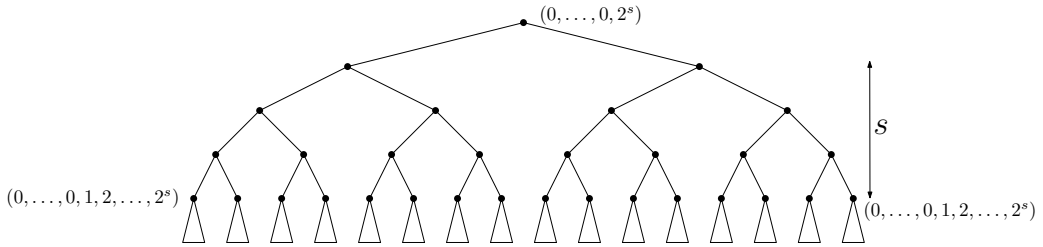

Figure 1: Construction in [16]: attaching a $(k, d, (0, \ldots, 0, 1, 2, \ldots, 2^s))$-tree to every leaf of a full binary tree of height s gives a (k, d)-tree.

having a leaf-vector $\vec{x}$ with $(\vec{x})_0 = 0$. For such a vertex v we add two children and associate leaf-vectors $\vec{x}'$ and $\vec{x}''$ to them. According to Observation 3.2 we have to split the coordinates of $\vec{x}$ observing $x'_{i-1} + x''_{i-1} = x_i$ for $1 \leq i \leq k$ and then we can decide about the last coordinates x'_k and x''_k almost freely, though we must respect the bounds $|\vec{x}'| \leq d$ and $|\vec{x}''| \leq d$.

We do not have to worry about nodes v with a leaf-vector $\vec{x}$ satisfying $w(\vec{x}) \geq 1$: Lemma 3.3 ensures that $\vec{x}$ is constructible. Making v the root of a $(k, d, \vec{x})$-tree we ensure that v keeps its promise.

The proof of Theorem 1.3 contains several technically involved arguments. In the next section we 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. A reader in a hurry for a formal argument is encouraged to skip right ahead to Section 5.

4 Informal Continuous Construction

4.1 Operations on leaf-vectors

By the argument at the end of the last section all we care about from now on are leaf-vectors and how we split them up between the two children, such that eventually all leaf-vectors have weight at least 1.

We will consider two fundamentally different ways a parent vertex v with leaf-vector $(x_0, x_1, \ldots, x_k)$ can split up its leaf-vector (in effect: its allotted number of k-close leaves) between its children. In the *fair split* both children get the same vector. In this case the children can even get a last coordinate of at most $d/2$ and their coordinate sum would still be at most d. In the simplest case of the *piecewise split* the left child gets all the leaves that are t-close and the right child gets the k-close leaves whose distance is more than t. In other words all the non-zero coordinates in the leaf-vector of the left child will be on the left of the non-zero

coordinates of the leaf-vector of the right child. For simplicity we keep the last coordinate of the leaf-vectors of the children 0. In the general case of the piecewise split we split a leaf-vector to many vectors, one inheriting all t-close leaves, while the others split the farther leaf-descendants evenly.

In the following informal description we will split leaf-vectors and divide their coordinates freely, not caring about divisibility. Dealing with rounding is one of the issues we leave to the formal argument.

Fair Split. The leaf-vector x of the parent node v is split evenly between its children v' and v''. Furthermore their last coordinate is $d/2$. That is,

$$x' = x'' = E(x) := (x_1/2, x_2/2, \ldots, x_k/2, d/2).$$

By m repeated applications of the fair split we obtain the leaf-vector

$$E^m(x) := \left(\frac{x_m}{2^m}, \frac{x_{m+1}}{2^m}, \ldots, \frac{x_k}{2^m}, \frac{d}{2^m}, \frac{d}{2^{m-1}}, \ldots, \frac{d}{2}\right).$$

After the m-times iterated fair split the leaf-vectors of all 2^m leaves of the full binary tree so obtained are equal. After this operation it is sufficient to ensure the constructibility of this single leaf-vector.

In the previous section the iterated fair split was used on the leaf-vector $(0, \ldots, 2^s)$ to obtain the leaf-vector $(0, \ldots, 0, 1, 2, 4, \ldots, 2^s)$. The constructibility of the latter vector was ensured by Lemma 3.3. The result obtained there is the best one can do using only fair splits and Lemma 3.3 and is a factor $\frac{1}{e}$ away from our goal. In order to improve we will also use the piecewise splitting of the leaf-vectors, where the $l = 1$ case of the piecewise split can be thought of as a sort of complete opposite of the fair split.

Piecewise Split This split has two parameters, r and l with $1 \leq l \leq r \leq k$. Piecewise split of a node v with leaf-vector $\vec{x} = (x_0, \ldots, x_k)$ is similar to the l-times iterated fair split in that we insert a depth l full binary tree under v. But instead of assigning the same leaf-vector to all 2^l leaves of this binary tree we assign a different leaf-vector $\vec{x}'$ to one leaf and the same leaf-vector $\vec{x}''$ to the remaining $2^l - 1$ leaves of this full binary tree. We call the node with leaf-vector x' the left-descendant and the ones with leaf-vector $\vec{x}''$ the right-descendants of v. In particular, we make the left-descendant of v inherit all the r-close leaves by setting

$$\vec{x}' = (x_l, \ldots, x_r, \underbrace{0, \ldots, 0}_{k-r+l})$$

and let the right-descendants evenly split the remaining ones by setting

$$\vec{x}'' = (\underbrace{0, \ldots, 0}_{r-l+1}, \frac{x_{r+1}}{2^l - 1}, \ldots, \frac{x_k}{2^l - 1}, \underbrace{0, \ldots, 0}_{l}).$$

To make a piecewise split useful we have to show that whenever $\vec{x}'$ and $\vec{x}''$ are constructible, so is $\vec{x}$. This follows from iterated application of Observation 3.2 if the intermediate vectors $\vec{x}^*$ (the leaf-vectors assigned to the intermediate vertices in the binary tree of depth l) satisfy the requirement $|\vec{x}^*| \leq d$. This condition will be satisfied in the cases where we apply piecewise split.

The advantage of a piecewise split is that since the coordinates with a small index are fully given to the left-descendant, their weight is multiplied by 2^l. We will set the parameters such that this makes the weight of $\vec{x}'$ reach 1, ensuring its constructibility by Lemma 3.3. For the right-descendants the weight-gain on the non-zero coordinates of the assigned leaf-vector is uniformly distributed, but tiny: only a factor $1 + \frac{1}{2^l-1}$. Furthermore the leaf-vector starts with many zeros, so we can perform a large number of fair splits and hope that the resulting leaf-vector is "better" in some way than $\vec{x}$, for example its weight increases. This will not always be the case in reality, because the behaviour of the weight in the optimal process is more subtle and can oscillate. This represents yet another, more serious technicality to handle in Section 5.

The cut subroutine in the next paragraph describes more formally the above combination of the piecewise split and the fair splits on the right-descendants.

The Cut Subroutine. The cut subroutine has a single parameter $l \in \{1, \ldots, k\}$. It can be applied to a leaf-vector $\vec{x}$ with $x_i = 0$ for $i < l$ and of weight $w(x) \geq 2^{-l}$. It consists of a piecewise split with parameters l and $r = r(l, \vec{x})$, where r, $l \leq r \leq k$, is the smallest index such that $\sum_{i=l}^{r} x_i/2^i \geq 2^{-l}$. The choice of r ensures that the leaf-vector $\vec{x}'$ of the left-descendant is constructible by Lemma 3.3. Then we apply an $(r-l)$-times iterated fair split to the leaf-vector $\vec{x}''$ of the right-descendants to obtain a leaf-vector

$$C^l(\vec{x}) = \left(0, \frac{x_{r+1}}{2^{r-l}(2^l-1)}, \ldots, \frac{x_k}{2^{r-l}(2^l-1)}, \underbrace{0, \ldots, 0}_{l}, \frac{d}{2^{r-l}} \ldots, \frac{d}{2}\right).$$

As we ensured the constructibility of $\vec{x}'$ and because we observed similar implications for the piecewise split and fair split operations we have that the constructibility of $C^l(\vec{x})$ implies the constructibility of $\vec{x}$.

In the rest of this subsection we give an illustration of how the cut subroutine works by considering the simplest case $l = 1$. This already gives a factor 2 improvement over the bound on d which we obtained in Section 3 using only the fair split. For the understanding of the general case of the cut subroutine, treated in the next subsection, this discussion is not crucial.

We saw earlier using repeated applications of the fair split and Observation 3.1 that in order to prove the existence of (k, d)-trees it is enough to see that the vector $\vec{x}^0 = (0, \ldots, 0, 1, 2, \ldots d/2)$ is constructible. Our plan is to establish this

through the repeated application of cuts with parameter $l = 1$: we recursively define $\vec{x}^i = C^1(\vec{x}^{i-1})$ and hope that we eventually obtain a vector with $w(\vec{x}^i) \geq 1$. By Lemma 3.3 this would establish the constructibility of $\vec{x}^i$, and through that, also the constructibility of $\vec{x}^0$ and the existence of (k, d)-trees.

In order to just get started with the first cut we need $w(\vec{x}^0) = \log d \cdot \frac{d}{2^{k+1}} \geq 2^{-l} = 1/2$ and thus $d > 2^k/k$. It turns out that if d is chosen slightly larger at $d = (1+\epsilon)2^k/k$ and k is large enough for ϵ, our plan of repeated cut subroutines with parameter $l = 1$ can indeed be carried through. Note that this bound on d is a factor of 2 smaller than the bound obtained from fair split and Lemma 3.3 alone in the previous section, but it is still larger by a factor of $e/2$ than the bound we need to prove Theorem 1.3. For the stronger bound we will need cuts with parameter $l > 1$, but as an illustration we give a very rough sketch here why $l = 1$ is enough if $d = (1+\epsilon)2^k/k$.

Let us start with examining the first cut producing $\vec{x}^1$. Except the first $\approx \log k$ coordinates, each coordinate contributes the same $\frac{d}{2^{k+1}} = (1+\epsilon)/(2k)$ to the weight of $\vec{x}^0$. Thus the first piecewise split will have parameter $r_1 \approx k/(1+\epsilon) \approx (1-\epsilon)k$. After the piecewise split the leaf-vector of the right descendant will have only $\approx \epsilon k$ non-zero entries, but the contribution to the weight of each of these entries is doubled to $\approx (1+\epsilon)/k$. This contribution is not changing during the repeated fair splits, but $r_1 - 1$ new non-zero entries show up, each with the "standard" $\approx (1+\epsilon)/(2k)$ contribution to the weight. In total we will have $w(\vec{x}^1) \approx (1+2\epsilon)/2$, a noticeable $\frac{\epsilon}{2}$ improvement over $w(\vec{x}^0)$.

This improvement in the weight of the coordinates towards the beginning of the leaf-vector makes the parameter of the second cut slightly smaller at $r_2 \approx (1-2\epsilon)k$, further improving the weight of $\vec{x}^2$. In general we will have $r_i \approx (1-i\epsilon)k$ and $w(\vec{x}^i) \approx (1+(i+1)\epsilon)/2$. This works till $r_i > k/2$. After that threshold (at around $i = 1/(2\epsilon)$) the rate by which the weight increases slows a little, but we will still have an index $i < 2/\epsilon$ with $w(\vec{x}^i) > 1$ as needed to finish this argument.

4.2 Passing to continuous

The goal of this subsection is to give the continous motivation behind the formal discrete proof of the next section as well as to elucidate *why* the discrete construction should work. The continuous function F, defined in this subsection via a differential equation, is also helpful in studying the *size* of the constructed formulas (see Section 7).

Recall that our goal is to obtain a (k, d)-tree for

$$d = \frac{1}{T}\frac{2^{k+1}}{k} \tag{4.1}$$

where T should be as large as possible and k suitably large for a given T. To

establish the upper bound of Theorem 1.3 we need this for any $T < e$. A similar result for a value $T > e$ would contradict to the lower bound in the same theorem.

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.

First we normalize the leaf-vector $\vec{z} = (z_0, \ldots, z_k)$ to get $(y_0, \ldots, y_k)$ with $y_i = 2^{k+1-i} z_i/d$. Note that we have $w(\vec{z}) = \sum_i y_i/(kT)$, so, in particular, $\vec{z}$ is constructible whenever $\sum_i y_i \geq kT$. We associate a real function $f : [0, 1] \to \mathbb{R}$ to the leaf-vector $\vec{z}$ by the formula $f(x) = y_{\lfloor kx \rfloor}$. Although this is a step function we will treat it as a continuous function. Clearly, as k increases, this is more and more justified.

Our Lemma 3.3 translates to our new setting as follows.

Lemma 4.1. *If the real function f associated to the leaf-vector $\vec{z}$ satisfies*

$$\int_{x=0}^{1} f(x)dx \geq T$$

then $\vec{z}$ is constructible.

We now illustrate how the cut subroutine (with sufficiently high parameter l chosen for $\epsilon > 0$) can be applied to achieve the target $T = e - \epsilon$. This is an informal analysis (as everything in this section) and hence we will allow ourselves to ignore what happens to the function $f(x)$ on $o(1)$-long subintervals of $[0, 1]$.

Our first job is to see how a single application of the cut subroutine with parameter l transforms the real function f associated to a leaf-vector. The cut starts with a piecewise split with parameters l and r where r is chosen minimal with respect to the condition that the leaf-vector of the left-descendant has weight at least 1. Let us set $v = r/k$ and consider the function

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

Notice that the true transformation of the leaf-vectors involves a left shift of l places for the entries up to x_r followed by zeros. This left shift explains why the normalized vector and thus the values of the associated real function are multiplied by 2^l. By the same left shift the values of the real function should also be shifted to the left by l/k, but as $l/k = o(1)$ this small effect is ignored here.

Notice that by Lemma 4.1 the choice of r translates to a choice of v that makes

$$T = \int_0^1 f_{\text{left}}(x)\ dx = 2^l \int_0^v f(x)\ dx. \tag{4.2}$$

The real function f_{right} associated to the leaf-vector of the right-descendants of the piecewise split with parameters l and r can similarly be approximated by

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

Recall that the cut subroutine further applies the $(r-l)$-times iterated fair split to the leaf-vector. Because of our normalization fair splits do not change the values of the associated real function, but they move these values to the left in the domain of the function. In our case the amount of this translation is $(r-l)/k \approx v$. The fair splits also introduce constant 1 values in the right end of the $[0,1]$ interval freed up by the shift. This explains why the following function approximates well the resulting real function:

$$C_f^l(x) = \begin{cases} \frac{2^l}{2^l-1} f(x+v) & x \in [0, 1-v) \\ 1 & x \in [1-v, 1] \end{cases} \tag{4.3}$$

In the following we analyze how the repeated application of these cut subroutines changes the associated real function. For this analysis we define a two-variable function $F(t,x)$ that approximates well how the function develops. Here $t \geq 0$ represents the "time" that has elapsed since we started our process, that is the cumulative length of the shifts v we made. The value $x \in [0,1]$ stands for the single variable of our current real function. In other words, for each fixed t, $F(t,x)$ should be a good approximation of the real function associated to the leaf-vector after t/v_{avg} infinitesimally small cuts were made (where v_{avg} is the length of the average cut). We have the initial condition $F(0,x) = 1$ as the constant 1 function approximates the real function associated to our original leaf-vector of $(0, \ldots, 0, 1, 2, \ldots, d/2)$ (in fact, the two functions coincide except for the $o(1)$ length subinterval of their domain where the latter function is 0). We also have $F(t,1) = 1$ for every $t \geq 0$ by (4.3).

We make yet another simplification: we assume that l is chosen large enough making v so small, that we can treat it as infinitesimal. We will denote the cut parameter v at time t by v_t. Recall that by (4.2) v_t can be defined by the formula $\int_0^{v_t} F(t,x)\ dx = T/2^l$, which simplifies to $v_t \approx T/(2^l F(t,0))$ after approximating the integral.

For the "right-descendant" we have by (4.3) that

$$F(t+v_t, x) = C^l_{F(t,\cdot)}(x) = \frac{2^l}{2^l-1} F(t, x+v_t)$$

if $x < 1 - v_t$ and $F(t+v_t, x) = 1$ for every $1 - v_t \leq x \leq 1$. Using the approximation $\frac{2^l}{2^l-1} \approx 1 + \frac{1}{2^l}$ (justified as l is considered "large") we approximate the above equation with

$$F(t+v_t, x) \approx \left(1 + \frac{1}{2^l}\right) F(t, x+v_t) \approx \left(1 + \frac{v_t F(t,0)}{T}\right) F(t, x+v_t).$$

This gives us an equation on the derivative of $F(t,x)$ in direction $(1,-1)$. For any $s>1$, define the function $F_s(t):[s-1,s]\to\mathbb{R}$ by $F_s(t)=F(t,s-t)$. Then rewriting the above with $s=x+t+v_t$, we have

$$F_s(t+v_t)\approx\left(1+\frac{v_tF(t,0)}{T}\right)F_s(t).$$

$$\frac{F_s'(t)}{F_s(t)}\approx\frac{F_s(t+v_t)-F_s(t)}{v_tF_s(t)}\approx\frac{F(t,0)}{T}. \tag{4.4}$$

Integrating we obtain

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

The left hand side evaluates to $\ln F_s(s)=\ln F(s,0)$ by the boundary condition $F_s(s-1)=1$.

As our last simplifying assumption we assume that the function $F(t,0)$ increases monotonically. Therefore the right hand side is at least $F(s-1,0)/T$, which implies that

$$F(s,0)\geq e^{\frac{F(s-1,0)}{T}}. \tag{4.5}$$

Now the increasing function $F(t,0)$ either goes to infinity or tends to a finite limit a. If the latter happens we have $a\geq e^{a/T}$. But classic calculus shows that $a\leq e^{a/e}$ for all real a, so this implies $T\geq e$. In the case $T<e$, that we study here, $F(t,0)$ must then tend to infinity. From the assumed monotonicity of $F(t,0)$ and (4.4) we obtain $F(t,x)>\sqrt{F(t,0)}$ if $1/2\leq x\leq 1$. Since $F(t,0)$ tends to infinity, so does $\int_0^1 F(t,x)\ dx$. Whenever this integral grows above T, the current leaf-vector is constructible by Lemma 4.1 finishing our highly informal proof of the existence of (k,d)-trees.

The above continuous heuristic is the underlying idea of the construction described in the next subsection. It provides a good approximation to what happens in the discrete case. Instead of dealing with all the introduced approximation errors in a precise manner, we give a direct discretized proof where we explicitely make sure that our many simplifying assumptions are satisfied and the approximations are correct.

5 Formal Construction of (k,d)-Trees

In this section we complete the proof of Theorem 1.3. To simplify notation we will omit vector arrows throughout.

Before proving Theorem 1.3 we first set up two of the main ingredients of our construction. Let us fix the positive integers k, d and l. To simplify the notation, we

will not show the dependence on these parameters in the next definitions, although d', E, C_r and C_r^* all depend on them. We let

$$d' = d\left(1 - \frac{1}{2^l - 1}\right).$$

For a vector $x = (x_0, \ldots, x_k)$ we define

$$E(x) = \left(\lfloor x_1/2 \rfloor, \lfloor x_2/2 \rfloor, \ldots, \lfloor x_k/2 \rfloor, \lfloor d'/2 \rfloor\right).$$

We denote by $E^m(x)$ the vector obtained from x by m applications of the operation E. 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.

$$E^m(x) = \left(\left\lfloor \frac{x_m}{2^m} \right\rfloor, \left\lfloor \frac{x_{m+1}}{2^m} \right\rfloor, \ldots, \left\lfloor \frac{x_k}{2^m} \right\rfloor, \left\lfloor \frac{d'}{2^m} \right\rfloor, \left\lfloor \frac{d'}{2^{m-1}} \right\rfloor, \ldots, \left\lfloor \frac{d'}{2} \right\rfloor\right).$$

For $l \le r \le k$ and the vector x as above we define the $(k+1)$-tuples $C_r(x)$ and $C_r^*(x)$ by the following formulas:

$$C_r(x) = \Big(\underbrace{0, \ldots, 0}_{r+1-l}, \underbrace{\left\lfloor \frac{x_{r+1}}{2^l - 1} \right\rfloor, \ldots, \left\lfloor \frac{x_k}{2^l - 1} \right\rfloor}_{k-r}, \underbrace{\left\lfloor \frac{d'}{2^l} \right\rfloor, \ldots, \left\lfloor \frac{d'}{2} \right\rfloor}_{l}\Big)$$

$$C_r^*(x) = (\underbrace{x_l, x_{l+1}, \ldots, x_r}_{r+1-l}, \underbrace{0, 0, \ldots, 0}_{k-r+l}).$$

Note that for the following lemma to hold we could use d instead of d' in the definition of E, and we could also raise most of the constant terms in the definition of C_r. The one term we cannot raise 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 k-close. We use d' everywhere to be consistent and provide for the monotonicity necessary in the proof of Theorem 1.3.

The first part of the next lemma states the properties of our fair split procedure (which is somewhat modified compared to the informal treatment of Section 4), the second part does the same for the cut subroutine.

Lemma 5.1. *Let k, d and l be positive integers and $x \in \mathbb{N}^{k+1}$ with $|x| \le d$.*

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

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

Proof. (a) We have $|E(x)| \leq |x|/2 + d'/2 < d$. 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 $|C_r^*(x)| \leq |x| \leq d$.

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 height 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'. We claim that T' is a (k, d, x)-tree showing the constructibility of x and finishing the proof of the lemma. To check condition (i) of the definition of a (k, d, x)-tree 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. Hence

$$\left(0, \ldots, 0, x_l, \ldots, x_r, (2^l - 1)\left\lfloor \frac{x_{r+1}}{2^l - 1} \right\rfloor, \ldots, (2^l - 1)\left\lfloor \frac{x_k}{2^l - 1} \right\rfloor\right)$$

is a leaf-vector of the root of T', thus x is also a leaf-vector.

Condition (ii) is satisfied for the root, because it has at most $|x| \leq d$ leaves that are k-close. Notice that the nodes of T' of distance at least l from the root are also nodes of T^* or a copy of T, so they satisfy condition (ii). There are two types of vertices in distance $0 < j < l$ from the root. One of them has 2^{l-j} copies of T below it, the other has one less and also T^*. In the first case we can bound the number of k-close leaves by

$$\begin{aligned} & 2^{l-j}\left(\frac{x_{r+1} + \ldots + x_k}{2^l - 1} + \frac{d'}{2^l} + \ldots + \frac{d'}{2^{l-j+1}}\right) \leq \frac{2^{l-j}}{2^l - 1} \cdot d + d'\left(1 - \frac{1}{2^j}\right) \\ = \; & \frac{d}{2^l - 1}\left(2^{l-j} + (2^l - 2)\left(1 - \frac{1}{2^j}\right)\right) \leq \frac{d}{2^l - 1}(2^l - 1) \\ = \; & d, \end{aligned} \tag{5.1}$$

where the last inequality holds since $j \geq 1$. We also point out that here it is crucial that we use the specific value of $d' < d$.

In the second case, the number of k-close leaves is bounded by

$$\begin{aligned} & (2^{l-j} - 1)\left(\frac{x_{r+1} + \ldots + x_k}{2^l - 1} + \frac{d'}{2^l} + \ldots + \frac{d'}{2^{l-j+1}}\right) + |C_r^*(x)| \\ \leq \; & (2^{l-j} - 1)\frac{d}{2^{l-j}} + \frac{d}{2^l} \leq d, \end{aligned}$$

where we used the result of the calculation in (5.1). □

Armed with the last lemma we are ready to prove Theorem 1.3.

Proof of Theorem 1.3. . We will show that (k,d)-trees exist for large enough k and $d = \lfloor 2^{k+1}/(ek) + 100 \cdot 2^{k+1}/k^{3/2} \rfloor$.

We set $l = \lfloor \log k/2 \rfloor$, so $2^l \sim \sqrt{k}$. We introduce the notation $s = 2l$. We define the vectors $x^{(t)} = (x_0^{(t)}, \ldots, x_k^{(t)}) \in \mathbb{N}^{k+1}$ recursively. We start with $x^{(0)} = E^{k-s}(z)$, where z denotes the vector consisting of $k+1$ zeros. 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 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. As we have observed earlier in this section, we can ignore all roundings but the last. This way we can write each of these entries in the form $\left\lfloor \frac{d'}{2^i(2^l-1)^j} \right\rfloor = \left\lfloor \frac{d'}{2^{i+lj}} \alpha^j \right\rfloor$ for some non-negative integers i and j and with $\alpha = 1 + 1/(2^l - 1)$. Using the values $q_t = r_t - s$ (this is the amount of "left shift" between $x^{(t)}$ and $x^{(t+1)}$) we can give the exponents explicitly.

$$x_j^{(t)} = \left\lfloor \frac{d'}{2^{k+1-j}} \alpha^{c(t,j)} \right\rfloor \tag{5.2}$$

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 $c(0,j) = 0$ for all j and $c(t,j) = 0$ if $q_{t-1} > k - j$.

The formal inductive proof of this formula is a straightforward calculation. We imagine each entry entering at the right end of the vector as $d'/2$ and divided by 2 every time it moves one place to the left — this explains the exponent $k+1-j$ of the 2 in the formula. This is accurate when the left-shift is the result of an application of E. However when some C_{r_i} is applied to an already existing entry, the entry moves l places to the left and the actual division is by $2^l - 1$ instead of 2^l, so the entry 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 $c(t,j) = t - t'$. This is exactly the number c of left shifts of length $q_{t-1}, \ldots, q_{t-c}$ that were needed to push the entry from its position in $x^{(t')}$ to its position j in $x^{(t)}$.

We claim next that $c(t,j)$ and $x_j^{(t)}$ increase monotonously in t for each fixed $s < j \leq k$, while q_t decreases monotonously in 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 (5.2). 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 $q_0 \geq q_1 \geq \cdots \geq q_{t+1}$, then by the definition of $c(i,j)$ we have $c(t+2,j) \geq c(t+1,j)$.

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

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

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 $x^{(t)}$ is infinite and coordinate-wise increasing. Notice also that these vectors were obtained through the operations E and C_r from the all-zero vector, so by Lemma 5.1 we must have $|x^{(t)}| \leq d$ for all t. Therefore the sequence $x^{(t)}$ must stabilize eventually. In fact, as the sequence stabilizes as soon as two consecutive vectors agree it must stabilize in at most d steps. So for some fixed vector $x = (x_0, \ldots, x_k)$ we have $x^{(t)} = x$ for all $t \geq d$. This implies that q_t also stabilizes with $q_t = q$ for $t \geq d$. (5.2) as applied to $t > d + k$ simplifies to

$$x_j = \left\lfloor \frac{d'}{2^{k+1-j}} \alpha^{\lfloor (k-j)/q \rfloor} \right\rfloor. \tag{5.3}$$

Recall that $q = q_t = r_t - s$ (take $t \geq d$), and r_t 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 $q \geq l$.

Proposition 5.2. *We have $q = l$.*

Proof: Assume for contradiction that $q > l$. Then by the minimality of r_t 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 $\frac{j}{q} \leq \frac{s+q}{q} = 1 + \frac{s}{q} \leq 1 + \frac{2l}{l} = 3$. This inequality simplifies to

$$2^{-l}(1 + 2^{-l})\alpha^4 \frac{2^{k+1}}{d'} > (q-1)\alpha^{k/q}. \tag{5.4}$$

Simple calculus gives that the right hand side takes its minimum for $q \geq 2$ between $q^* - 1$ and $q^* - 2$, where $q^* = q^*(k) = k \ln \alpha \sim \sqrt{k}$, and this minimum is more than $(q^* - 3)e$. Using $\alpha = 1 + 1/(2^l - 1) > e^{2^{-l}}$ we have $q^* \geq k/2^l$. So (5.4) yields

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

Thus,

$$(1 + 2^{-l})\alpha^4 \frac{2^{k+1}}{d'} - ke + 3e2^l > 0.$$

Substituting our choice for d, d', α and l (as functions of k) and assuming that k is large enough we get that the left hand side is at most

$$\begin{aligned}
& \left(1 + \frac{2}{\sqrt{k}}\right)\left(1 + \frac{4}{\sqrt{k}}\right)^4 \frac{2^{k+1}}{d\left(1 - \frac{4}{\sqrt{k}}\right)} - ek + 3e\sqrt{k} \\
\leq & \left(1 + \frac{19}{\sqrt{k}}\right) \frac{ek}{\left(1 - \frac{4}{\sqrt{k}}\right)\left(1 + \frac{99e}{\sqrt{k}}\right)} - ek + 3e\sqrt{k} \\
\leq & \left(1 - \frac{200}{\sqrt{k}}\right) ek - ek + 3e\sqrt{k} = -200e\sqrt{k} + 3e\sqrt{k} < 0,
\end{aligned}$$

which is a contradiction. Hence $q = l$ as claimed. □

Next we establish by downward induction that the vectors $x^{(t)}$ are constructible. We start with $t = d$. Using (5.3) and the fact that $q = l$ (Proposition 5.2) we get that for large k

$$\begin{aligned}
w(x^{(d)}) &\geq \frac{x^{(d)}_{s+1}}{2^{s+1}} \geq \frac{d'}{2^{k+1}} \alpha^{\frac{k-s-1}{l} - 1} - 1 \\
&\geq \frac{d'}{2^{k+1}} \alpha^{\frac{k}{2l}} - 1 \\
&\geq \frac{d/2}{2^{k+1}} \left(1 + \frac{1}{\sqrt{k}}\right)^{\frac{k}{\log k}} - 1 \\
&\geq \frac{1}{2ek} e^{\frac{k}{2\sqrt{k}\log k}} - 1 > 1.
\end{aligned}$$

So by Lemma 3.3, $x^{(d)}$ is constructible.

Now assume that $x^{(t+1)}$ is constructible for some $t \leq d-1$. Recall that $x^{(t+1)} = E^{r_t-s-l}(C_{r_t}(x^{(t)}))$, so by (the repeated use of) part (a) of Lemma 5.1, $C_{r_t}(x^{(t)})$ 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 the first $s+1 > l$ coordinates of $x^{(t)}$ are 0. This shows that the weight of $C^*_{r_t}$ is at least 1 and therefore Lemma 3.3 proves (i).

For (ii) we use monotonicity to see $|C^*_{r_t}(x^{(t)})| = \sum_{j=s+1}^{r_t} x_j^{(t)} \leq \sum_{j=s+1}^{r_t} x_j$. Here $r_t \leq r_0 \leq k/2$ for large enough k. Using the fact that $q = l$ (Proposition 5.2) and (5.3) we further have for large k that

$$\begin{aligned}
|C^*_{r_t}(x^{(t)})| &\leq d' \sum_{j=s+1}^{r_t} \frac{1}{2^{k+1-j}} \cdot \alpha^{\frac{k-j}{l}} \\
&\leq d \sum_{j=s+1}^{r_t} \left(\frac{\alpha}{2}\right)^{k-j} \\
&\leq d \left(\frac{3}{4}\right)^{k-r_0} \cdot 4 \quad \left(\text{since } \frac{\alpha}{2} \leq \frac{3}{4}\right) \\
&\leq d \left(\frac{3}{4}\right)^{\frac{k}{2}} \cdot 4 \quad \left(\text{since } r_t \leq \frac{k}{2}\right) \\
&< \frac{d}{\sqrt{k}} \leq \frac{d}{2^l}.
\end{aligned}$$

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 5.1 (a) implies that the all-zero vector z is also constructible. Thus the larger vector $(0, \ldots, 0, d)$ is constructible too, and by Observation 3.1 there exists a (k, d)-tree. □

6 Proof of the Lower Bound of Theorem 1.1

For our proof we use the Lopsided Local Lemma of Erdős and Spencer:

Lemma 6.1. *(Lopsided Local Lemma [11]) 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. Moreover, 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 equivalent to $Pr(\wedge_{D \in J} \bar{A}_D | A_C) \leq Pr(\wedge_{D \in J} \bar{A}_D)$. The right hand side is simply the probability of a random assignment satisfying all clauses in J. The left hand side can be interpreted as the same probability after a random assignment is modified by setting all literals in C to `false`. The random assignment does not satisfy any clauses in J after the modification that it did not satisfy before, since no clause from J contains a variable of C in the opposite form. Hence the probability of satisfying all of them does not grow by the modification proving the inequality. (The probability might, in fact, decrease if some of the clauses in J contain a literal also present in C.)

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 not to be satisfied we must not set any of the independent literals in C to `true`, and therefore

we have

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

The inequality in the fourth line holds due to the well-known inequality that $1-ax \leq (1-x)^a$ for every $0 < x < 1$ and $a \geq 1$.

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 $f(k) \geq s = \left\lfloor \frac{2^{k+1}}{e(k+1)} \right\rfloor$.

7 More About the Class MU(1) and Outlook

7.1 Constructing binary trees and MU(1) formulas

The structure of MU(1) formulas is well understood and it is closely related to binary trees. (Recall that by a *binary tree* we always mean a rooted tree where every non-leaf node has exactly two children.) In particular, given any binary tree T, we associate with it certain CNF formulas. Similarly to Section 2.1 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 clause than the number of variables associated with T. Note that for proving the upper bound in Theorem 1.1 we used (k,d)-trees T and the associated unsatisfiable (cf. Observation 2.1) (k,d)-CNF $F_k(T)$ that was

constructed similarly, but selecting the k vertices farthest from the root on every root-leaf path.

As proved in [7], 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.

Recall that $f_{\mathrm{tree}}(k)$ denotes the smallest integer d such that a (k,d)-tree exists. Clearly, $f(k) \leq f_1(k) < f_{\mathsf{tree}}(k)$, and we showed that $f(k) = (1+o(1))f_{\mathsf{tree}}(k)$.

7.2 On the size of unsatisfiable formulas

By the *size* of a rooted 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 containment-minimal (k,d)-tree T and the size of the corresponding (k,d)-CNF $F_k(T)$ 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. In fact, the sizes 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 vector $(x_0,\ldots,x_k)$, where x_j is the number of leaf-descendants of v of distance j from v. If a (k,d)-tree has minimal size it has no two vertices along the same branch with identical vectors. In fact, this statement is also true even if one forgets about the last entry in the vector. So the minimal height of a (k,d)-tree is limited by the number of vectors in $\mathbb{N}^k$ with L_1 norm at most d, which is $\binom{d+k}{k} \leq d^k$. For $d = f_{\mathrm{tree}}(k)$ this is $2^{O(k^2)}$. For the minimal size of a (k,d)-tree this implies a $2^{2^{O(k^2)}}$ bound that applies whenever such a tree exists. The same bound for minimal size (k,d)-CNF formulas in MU(1) is implicit in [17]. There is numerical evidence that the sizes of the minimal $(k, f_{\mathrm{tree}}(k))$-tree and the minimal $(k, f_1(k)+1)$-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 the upper bound in Theorem 1.3 shows that the height of the (k,d)-tree constructed there is at most $d \log k$. 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. Here we just sketch the argument for this statement and do not give a formal proof. By following the approach of Section 4 (and neglecting the small error terms neglected

there) we obtain the recurrence (4.5) on $F(s,0)$. As discussed there this recursion for parameters considered here diverges, so we get $F(s,0)$ arbitrarily large for a suitable (constant) value of s. Note that $F(s,0)$ is obtained as the approximate value of $2^{k+1-j}x_j/d$ for most j small compared to k, where $(x_0,\ldots,x_k)$ is the leaf-vector associated to a certain vertex in the (k,d) to be constructed. The distance of this vertex from the root is approximately ks. Note that when this value reaches a high constant Lemma 3.3 becomes applicable to this vertex and the construction of the (k,d)-tree ends within k more levels. This makes the height of the (k,d)-tree obtained $O(k)$. Note that a height of $O(k)$ makes the size of this tree polynomial in d.

Let us define $f_1(k,d)$ for $d > f_1(k)$ to be the minimal size of a (k,d)-CNF in MU(1), and let $f_{\mathsf{tree}}(k,d)$ stand for the minimal size of a (k,d)-tree, assuming $d \geq f_{\mathrm{tree}}(k)$. While $f_1(k,f_1(k)+1)$ and similarly $f_{\mathsf{tree}}(k,f_{\mathsf{tree}}(k)))$ are probably doubly exponential in k, the above argument shows that for slightly larger values of $d=(1+\epsilon)f(k)$ the values $f_1(k,d)$ and $f_{\mathrm{tree}}(k,d)$ are polynomial in d (and thus simply exponential in k).

7.3 Extremal Values and Algorithms

Finally, we mention the question whether $f_{\mathrm{tree}}(k) = f_1(k)+1$ for all k. (The $+1$ term comes from us defining these functions inconsistently. While $f(k)$ and $f_1(k)$ is traditionally defined as the *largest* value d with all (k,d)-CNF satisfiable or no (k,d)-CNF in MU(1) exist, respectively, it was more convenient for us to define $f_{\mathrm{tree}}(k)$ as the *smallest* value d for which (k,d)-trees exist.) This question asks if one necessarily loses (in the maximal number of appearances of a variable) by selecting the k vertices farthest from the root when making an MU(1) k-CNF formula from a binary tree. As mentioned above, $f(k)=f_1(k)$ is also open, but $f_{\mathrm{tree}}(k)=f_1(k)+1$ 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 $f_{\mathrm{tree}}(k,d)=f_1(k,d)$ for $k\leq 7$ and all $d > f_1(k)$ with just a single exception. We have $f_1(7)=17$ and $f_1(7,18)=10,197,246,480,846 < f_{\mathrm{tree}}(7,18)=10,262,519,933,858$. We also found $f_1(8,d)=f_{\mathrm{tree}}(8,d)$ for $f_1(8)=29<d\leq 33$. Is $f_1(7,18)<f_{\mathrm{tree}}(7,18)$ the exception or will it turn into the rule for larger values? Does it indicate that $f_{\mathrm{tree}}(k)$ and $f_1(k)+1$ 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 $f_{\mathrm{tree}}(k)$ substantially faster than the algorithm of Hoory and Szeider [17] 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.

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] N. Alon, J. Spencer, *The Probabilistic Method*, John Wiley-Interscience, 2008.

[3] J. Beck, *Combinatorial Games: Tic Tac Toe Theory*, Encyclopedia of Mathematics and Its Applications 114, 2008

[4] P. Berman and M. Karpinski, On Some Tighter Inapproximability Results Proc. 26th ICALP (1999), LNCS 1644, Springer 1999, 200-209.

[5] 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.

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

[7] 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.

[8] B. Doerr, Vector Balancing Games with Aging, *Journal of Combinatorial Theory, Series A* **95** (2001), 219–233.

[9] B. Doerr, European Tenure Games, *Theoretical Computer Science* **313** (2004), 339–351.

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

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

[12] P. Erdős and J.L. Selfridge, On a combinatorial game, *J. Combinatorial Theory Ser. A* **14**, (1973), 298–301.

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

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

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

[16] H. Gebauer, Disproof of the Neighborhood Conjecture with Implications to SAT, *Combinatorica* **32**(5), (2012), 573–587.

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

[18] 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.

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

[20] Fiachra Knox, Two constructions relating to conjectures of Beck on positional games, http://arxiv.org/pdf/1212.3345v1.pdf.

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

[22] 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.

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

[24] R.A. Moser, G. Tardos, A Constructive Proof of the General Lovász Local Lemma, *J. ACM* **57**(2), (2010)

[25] A. Pelc, Searching games with errors—fifty years of coping with liars *Theoretical Computer Science* **270** (1–2) (2002), 71–109.

[26] `http://polymathprojects.org/2012/07/12/minipolymath4-project-imo-2012-q3/`

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

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

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

[30] 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.

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

[32] J.B. Shearer, On a problem of Spencer, *Combinatorica* **5**, (1985), 241–245

[33] J. Spencer, Randomization, Derandomization and Antiderandomization: Three Games, *Theoretical Computer Science* **131** (1994), 415–429.

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

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