An Algorithmic Proof of the Lopsided Lovász Local Lemma
(simplified and condensed into lecture notes)

Nicholas J. A. Harvey
University of British Columbia
Vancouver, Canada
nickhar@cs.ubc.ca

Jan Vondrák
IBM Almaden
San Jose, CA
jvondrak@us.ibm.com

Abstract

For any probability space and a collection of events we give an algorithmic proof of the Lovász Local Lemma if there is a *resampling oracle* for each event. The resampling oracles act as an abstraction layer that isolates the details of the proof from the details of the probability space. The number of resamplings performed by the algorithm is bounded by expressions similar to those obtained by Moser and Tardos, and others. The overall computational efficiency depends on the runtime of the resampling oracles. We develop efficient resampling oracles that unify previous algorithmic applications of the local lemma, and present new results for rainbow matchings.

The long version of this paper is available at http://arxiv.org/abs/1504.02044

1 Introduction

The Lovász Local Lemma (LLL) is a powerful tool with numerous uses in combinatorics and theoretical computer science. It asserts the existence of a point in a probability space that simultaneously avoids specified undesired events if their probabilities and dependences are bounded by certain criteria. The classical formulation of the LLL [2, 9, 3] is as follows.

Let Ω be a discrete probability space with probability measure μ. Let $E_1, \ldots, E_n$ be "undesired" events in that space. Let G be a graph with vertex set $[n] = \{1, \ldots, n\}$. We denote the edge set by $E(G)$, and the neighborhood of vertex i by $\Gamma(i) = \{\, j \neq i : (i,j) \in E(G) \,\}$. Also, let $\Gamma^+(i) = \Gamma(i) \cup \{i\}$ and $\Gamma^+(I) = \bigcup_{i \in I} \Gamma^+(i)$.

Theorem 1.1 (General Lovász Local Lemma [2, 9])**.** *Suppose that the events satisfy the following condition that controls their dependences*

$$\Pr_\mu[E_i \mid \cap_{j \in J} \overline{E_j}] \;=\; \Pr_\mu[E_i] \qquad \forall i \in [n],\, J \subseteq [n] \setminus \Gamma^+(i) \tag{Dep}$$

and the following criterion that controls their probabilities

$$\exists x_1, \ldots, x_n \in (0,1) \qquad \textit{such that} \qquad \Pr_\mu[E_i] \;\leq\; x_i \prod_{j \in \Gamma(i)} (1 - x_j) \quad \forall i \in [n]. \tag{GLL}$$

Then $\Pr_\mu[\bigcap_{i=1}^n \overline{E_i}] \geq \prod_{i=1}^n (1 - x_i) > 0$.

Algorithms. Algorithms to efficiently find such a desired point in $\bigcap_{i=1}^n \overline{E_i}$ have been the subject of much research. We have already discussed Moser's algorithm [7] that handles the bounded-degree k-SAT problem. We also mentioned the result of Moser and Tardos [8], that generalizes the algorithm and its analysis to the general LLL with events on a space of independent random variables.

Because these results place restrictions on the probability space, they cannot be viewed as algorithmic proofs of the LLL in full generality. There are plenty of applications of the General LLL involve more general probability spaces that do *not* consist of many independent variables. Common examples include random permutations, matchings and spanning trees (see [5, 6]). The methods discussed in this lecture can handle such probability spaces.

Our contributions. We propose an algorithmic framework that provides a constructive proof of the Lovász Local Lemma in an abstract setting. To make this precise, let us introduce further notation. An atomic event ω in the probability space Ω will be called a *state*. We write $\omega \sim \mu$ to denote that a random state ω is distributed according to μ, and $\omega \sim \mu|_{E_i}$ to denote that the distribution is μ conditioned on E_i. The precise assumptions of our model are as follows.

Algorithmic assumptions. Our framework requires only two assumptions in order to give an algorithmic proof of the Local Lemma.

- *Sampling from μ:* We are able to generate a random state $\omega \sim \mu$.
- *Resampling oracles:* For each event E_i, we have a (randomized) *resampling oracle* $r_i : \Omega \to \Omega$ with the following properties.

  (R1) If E_i is an event and $\omega \sim \mu|_{E_i}$, then $r_i(\omega) \sim \mu$. (The oracle r_i removes conditioning on E_i.)

  (R2) For every ω in which E_i occurs but E_j does not, for $j \notin \Gamma^+(i)$, then E_j does not occur in $r_i(\omega)$ either. (Resampling an undesired event that occurs cannot cause new non-neighbor events.)

It is permitted by (R2)that E_j occurs in ω and E_j does not occur in $r_i(\omega)$. That is, resampling can remove the occurrence of non-neighbor events. Note also that the behavior of $r_i(\omega)$ is undefined if E_i does not occur in ω.

Main Result. Our main result is that, for any probability space, if algorithmic access is provided via resampling oracles, then the Lovász Local Lemma can be proven algorithmically.

Theorem (Informal)**.** *For any probability space and any events with resampling oracles, if the Local Lemma criteria are satisfied, then our algorithm finds a point avoiding all events in oracle-polynomial time.*

This recovers the main result of Moser-Tardos [8] (with a slightly weaker bound on the number of resamplings). But it is much more general because it holds in the framework of abstract resampling oracles.

1.1 Oracle for independent variables

To illustrate the definition of a resampling oracle, let us consider the independent-variable model, considered originally by Moser and Tardos [8]. Here, Ω is a product space, with independent random variables $\{X_a : a \in \mathcal{U}\}$. The probability measure μ here is a product measure. Each bad event E_i depends on a particular subset of variables A_i, and two events are independent iff $A_i \cap A_j = \emptyset$.

Here our algorithmic assumptions correspond exactly to the Moser-Tardos framework [8]. Sampling from μ means generating a fresh set of random variables independently. The resampling operation r_i takes a state ω and replaces the random variables $\{X_a : a \in A_i\}$ by fresh random samples. It is easy to see that the assumptions are satisfied: in particular, a random state sampled from μ conditioned on E_i has all variables outside of A_i independently random. Hence, resampling the variables of A_i produces the distribution μ. Clearly, resampling $\{X_a : a \in A_i\}$ does not affect any events that do not intersect A_i.

1.2 The MaximalSetResample algorithm

As mentioned above, Moser and Tardos [8] have given a very simple algorithm that proves the LLL when the probability spaces consists of independent variables. We propose a slightly different algorithm that will allow us to handle more general probability spaces. Our algorithm resamples, in turn, a maximal independent set of events that occur, chosen according to an arbitrary but fixed ordering. This additional structure on the sequence of resamplings seems to simplify the analysis our abstract setting. Pseudocode is shown in Algorithm 1.

Algorithm 1 MaximalSetResample uses resampling oracles to output a state $\omega \in \bigcap_{i=1}^{n} \overline{E_i}$.

```
Initialize ω with a random state sampled from μ;
t := 0;
repeat
    t := t + 1;
    J_t := ∅
    while there is i ∉ Γ⁺(J_t) such that E_i occurs in ω do
        Pick the smallest such i;
        J_t := J_t ∪ {i};
        ω := r_i(ω);    ▷ Resample E_i
    end while
until J_t = ∅;
return ω.
```

Trivially, if MaximalSetResample terminates then it has found a state $\omega \in \bigcap_{i=1}^{n} \overline{E_i}$. Therefore, we only have to prove that MaximalSetResample terminates in polynomial time. In Section 2, we prove the following fairly simple result.

Theorem 1.2. *Suppose that* (GLL) *is satisfied with a slack of* $(1-\epsilon)$*, i.e.*

$$\Pr_{\mu}[E_i] \;\leq\; (1-\epsilon)x_i \prod_{j \in \Gamma(i)} (1-x_j) \qquad \forall i.$$

Then MaximalSetResample terminates in $O(\frac{1}{\epsilon}(1 + \sum_{i=1}^{n} \ln \frac{1}{1-x_i}))$ *iterations, with high probability.*

The full version of the paper proves several stronger results — in particular, the assumption of $(1-\epsilon)$ slack can be removed.

2 Analysis of the algorithm

2.1 Notation and basic facts

Our analysis uses several ideas introduced by Kolipaka and Szegedy [4].

Definition 2.1. *One execution of the outer repeat loop in MaximalSetResample is called an* iteration*. For a sequence of non-empty sets* $\mathcal{I} = (I_1, \ldots, I_t)$*, we say that the algorithm* follows $\mathcal{I}$ *if* I_s *equals the set* J_s *resampled in iteration* s *for* $1 \leq s \leq t$*.*

Definition 2.2. $\mathcal{I} = (I_1, I_2, \ldots, I_t)$ *is called a* neighbor set sequence *if* $I_1, \ldots, I_t \subseteq V(G)$ *and* $I_{s+1} \subseteq \Gamma^+(I_s)$ *for each* $1 \leq s < t$*. We call the sequence* $\mathcal{I}$ proper *if each set* I_s *is nonempty.*

Note that if $I_s = \emptyset$ for some s, then $I_t = \emptyset$ for all $t > s$. Therefore, the nonempty sets always form a prefix of the neighbor set sequence.

Lemma 2.3. *If MaximalSetResample follows a sequence $\mathcal{J} = (J_1, \ldots, J_t)$, then $\mathcal{J}$ is a neighbor set sequence.*

Proof. For each $i \in J_s$, we execute the resampling oracle r_i. Recall that r_i executed on a satisfied event E_i can only cause new events in the neighborhood $\Gamma^+(i)$ (and this neighborhood is not explored again until the following iteration). Since J_s is a maximal independent set of satisfied events, all the events satisfied in the following iteration are neighbors of some event in J_s, i.e., $J_{s+1} \subseteq \Gamma^+(J_s)$. □

We use the following notation: For $i \in [n]$, $p_i = \Pr_\mu[E_i]$. For $S \subseteq [n]$, $p^S = \prod_{i \in S} p_i$. For a neighbor set sequence $\mathcal{I} = (I_1, \ldots, I_t)$, $p_\mathcal{I} = \prod_{s=1}^t p^{I_s}$.

One of the main steps in the analysis is to bound the probability that the algorithm follows a given neighbor set sequence. The proof is quite similar to the analogous step of the Moser-Tardos analysis [8], except we only need the abstract properties of resampling oracles.

Lemma 2.4. *For any proper neighbor set sequence $\mathcal{I} = (I_1, I_2, \ldots, I_t)$, the probability that the MaximalSetResample algorithm follows $\mathcal{I}$ is at most $p_\mathcal{I}$.*

Proof. Given $\mathcal{I} = (I_1, I_2, \ldots, I_t)$, let us consider the following "$\mathcal{I}$-checking" random process. We start with a random state $\omega \sim \mu$. In iteration s, we process the events of I_s in the ascending order of their indices. For each $i \in I_s$, we check whether ω satisfies E_i; if not, we terminate. Otherwise, we apply the resampling oracle r_i and replace ω by $r_i(\omega)$. We continue for $s = 1, 2, \ldots, t$. We say that the $\mathcal{I}$-checking process succeeds if every event is satisfied when checked and the process runs until the end.

By induction, the state ω after each resampling oracle is distributed according to μ: Assuming this was true in the previous step and conditioned on E_i satisfied, we have $\omega \sim \mu|_{E_i}$. By assumption, the resampling oracle r_i removes this conditioning and produces again a random state $r_i(\omega) \sim \mu$. Therefore, conditioned on the $\mathcal{I}$-check process arriving at the step that checks E_i, the check succeeds with probability $\Pr_\mu[E_i]$. Taking the product of these conditional probabilities, the probability that the entire $\mathcal{I}$-checking process succeeds is exactly $\prod_{s=1}^t \prod_{i \in I_s} \Pr_\mu[E_i] = \prod_{s=1}^t p^{I_s} = p_\mathcal{I}$.

To conclude, we argue that the probability that MaximalSetResample follows the sequence $\mathcal{I}$ is at most the probability that the $\mathcal{I}$-checking process succeeds. To see this, suppose that we simultaneously run MaximalSetResample and the $\mathcal{I}$-checking process with the same source of randomness. (This is an example of "coupling".) In each iteration, if MaximalSetResample includes i in J_t, it means that E_i occurs. Both processes apply the resampling oracle $r_I(\omega)$ so their distributions in the next iteration are the same. Therefore, the event that MaximalSetResample follows the sequence $\mathcal{I}$ is contained in the event that the $\mathcal{I}$-checking process succeeds, which happens with probability $p_\mathcal{I}$. □

We emphasize that we do *not* claim that the distribution of the MaximalSetResample algorithm's state $\omega \in \Omega$ is μ after each resampling. This would mean that the algorithm is not making any progress in its search for a state avoiding all events. It is only the $\mathcal{I}$-checking process that has this property.

Definition 2.5. *Let $\mathcal{N}$ denote the set of all neighbor set sequences and $\mathcal{P}$ the set of proper neighbor set sequences.*

Lemma 2.6. *The probability that MaximalSetResample runs for at least ℓ iterations is upper-bounded by $\sum_{\mathcal{I}=(I_1,\ldots,I_\ell) \in \mathcal{N}} p_\mathcal{I}$.*

Proof. If the algorithm runs for at least ℓ iterations, it means that it follows some proper sequence $\mathcal{I} = (I_1, I_2, \ldots, I_\ell)$. By Lemma 2.4, the probability that the algorithm follows a particular neighbor set sequence $\mathcal{I}$ is at most $p_{\mathcal{I}}$. By the union bound, the probability that the algorithm runs for at least ℓ iterations is at most $\sum_{\mathcal{I}=(I_1,\ldots,I_\ell)\in\mathcal{P}} p_{\mathcal{I}}$. □

2.2 Analysis in the General LLL setting

Let us prove the following crude (typically exponentially large) bound on the number of iterations.

Lemma 2.7. *Assuming* (GLL) *is satisfied, we have* $\sum_{\mathcal{I}\in\mathcal{P}} p_{\mathcal{I}} \le \prod_{i=1}^{n} \frac{1}{1-x_i}$.

The proof uses a standard fact.

Fact 2.8. *For any set S and any values α_i, we have* $\prod_{i\in S}(1+\alpha_i) = \sum_{S'\subseteq S} \prod_{i\in S'} \alpha_i$.

Proof (of Lemma 2.7). We show the following statement by induction on ℓ: For any fixed set J,

$$\sum_{\substack{\mathcal{I}=(I_1,\ldots,I_\ell)\in\mathcal{N}\\ I_1=J}} p_{\mathcal{I}} \;\le\; \prod_{j\in J} \frac{x_j}{1-x_j}. \tag{1}$$

Base case: $\ell = 1$. This holds since $p_{(J)} = \prod_{j\in J} p_j \le \prod_{j\in J} x_j$ by (GLL).
Inductive step. Let us consider the expression for $\ell + 1$. We have

$$\begin{aligned}
\sum_{\substack{\mathcal{I}'=(I_0,I_1,\ldots,I_\ell)\in\mathcal{N}\\ I_0=J}} p_{\mathcal{I}'} \;&=\; p^J \sum_{\substack{\mathcal{I}=(I_1,\ldots,I_\ell)\in\mathcal{N}\\ I_1\subseteq\Gamma^+(J)}} p_{\mathcal{I}} \\
&\le\; p^J \sum_{I_1\subseteq\Gamma^+(J)} \prod_{i\in I_1} \frac{x_i}{1-x_i} \qquad \text{(by induction)} \\
&=\; p^J \prod_{i\in\Gamma^+(J)} \left(1+\frac{x_i}{1-x_i}\right) \qquad \text{(by Fact 2.8 with } \alpha_i = \tfrac{x_i}{1-x_i}) \\
&=\; \Big(\prod_{i'\in J} p_{i'}\Big) \cdot \prod_{i\in\Gamma^+(J)} \frac{1}{1-x_i} \\
&\le\; \Big(\prod_{i'\in J} x_{i'} \prod_{j\in\Gamma(i')} (1-x_j)\Big) \cdot \prod_{i\in\Gamma^+(J)} \frac{1}{1-x_i} \qquad \text{(by (GLL))} \\
&\le\; \prod_{i\in J} \frac{x_i}{1-x_i}
\end{aligned}$$

because each factor $\frac{1}{1-x_i}$ for $i \in \Gamma^+(J) \setminus J$ is covered at least once by $(1-x_j)$ where $j \in \Gamma(i')$ for some $i' \in J$. This proves (1).

Finally, summing (1) over all sets $J \subseteq [n]$, we use Fact 2.8 again to obtain

$$\sum_{\mathcal{I}=(I_1,\ldots,I_\ell)\in\mathcal{N}} p_{\mathcal{I}} \;\le\; \sum_{J\subseteq[n]} \prod_{j\in J} \frac{x_j}{1-x_j} \;=\; \prod_{i=1}^{n} \left(1+\frac{x_i}{1-x_i}\right) \;=\; \prod_{i=1}^{n} \frac{1}{1-x_i}. \tag{2}$$

Note that a sequence in $\mathcal{I} \in \mathcal{P}$ of length $k \leq \ell$ can be padded to a sequence $\mathcal{I}' \in \mathcal{N}$ of length ℓ by appending copies of $\emptyset$. furthermore $p_{\mathcal{I}} = p_{\mathcal{I}'}$. (E.g., $p_{(I_1,I_2)} = p_{(I_1,I_2,\emptyset,\ldots,\emptyset)}$.) So, (2) can be written equivalently as

$$\sum_{1 \leq k \leq \ell} \sum_{\mathcal{I}=(I_1,\ldots,I_k) \in \mathcal{P}} p_{\mathcal{I}} \;\leq\; \prod_{i=1}^{n} \frac{1}{1-x_i}.$$

Since this is true for every ℓ, it remains true in the limit $\ell \to \infty$. □

Proof (of Theorem 1.2). Let us estimate the probability that the algorithm runs for at least ℓ iterations. By Lemma 2.4, this can be upper-bounded by

$$\sum_{k=\ell}^{\infty} \sum_{\mathcal{I}=(I_1,\ldots,I_k) \in \mathcal{P}} p_{\mathcal{I}}.$$

Lemma 2.7 shows that this is at most $\prod_{i=1}^{n} \frac{1}{1-x_i}$. But since we assume that (GLL) holds with $(1-\epsilon)$ slack, and since each set in a proper sequence is nonempty, may include a factor of $1-\epsilon$ for each set in the sequence. We obtain

$$\sum_{k=\ell}^{\infty} \sum_{\mathcal{I}=(I_1,\ldots,I_k) \in \mathcal{P}} p_{\mathcal{I}} \;\leq\; (1-\epsilon)^{\ell} \prod_{i=1}^{n} \frac{1}{1-x_i}. \;\leq\; e^{-t},$$

by setting $\ell = \frac{1}{\epsilon}(t + \sum_{i=1}^{n} \ln \frac{1}{1-x_i})$. Therefore, the probability of running beyond $\frac{1}{\epsilon}\sum_{i=1}^{n} \ln \frac{1}{1-x_i}$ iterations decays exponentially. □

3 Perfect Matchings in Complete Graphs

3.1 Resampling Perfect matchings

Let us now discuss how to implement resampling oracles for a scenario without independent variables — perfect matchings in complete graphs. The probability space Ω is the set of all perfect matchings in K_{2n}, with the uniform measure. A state here is a perfect matching in K_{2n}, which we denote by $M \in \Omega$. We consider bad events of the following form: E_A for a set of edges A occurs if $A \subseteq M$. Obviously, $\Pr_{\mu}[E_A] > 0$ only if A is a (partial) matching. Let us define $A \sim B$ iff $A \cup B$ is *not* a matching.

We show that it is possible to implement a resampling oracle in this setting. In Algorithm 3.1, we describe the implementation for the simple case that A contains a single edge uv. The case of general A is discussed in the full paper.

Algorithm 2 Resampling oracle for the event E_A with $A = \{uv\}$.

1: **Function** $r_A(M)$:
2: If $uv \notin M$, **fail**.
3: Pick $xy \in M \setminus \{uv\}$ uniformly at random (with x and y randomly ordered)
4: With probability $1 - \frac{1}{2n-1}$, add uy, vx to M and remove uv, xy from M
5: **return** M.

Lemma 3.1. *Let $A = \{uv\}$ and let M be distributed uniformly among perfect matchings in K_{2n} such that $uv \in M$. Then after the resampling operation, $r_A(M)$ is a uniformly random perfect matching.*

Proof. Pick $xy \in M \setminus \{uv\}$ uniformly at random. Observe that for a uniformly random perfect matching, the edge uv should appear with probability $1/(2n-1)$, since u has $2n-1$ choices to be matched with and v is 1 of them. Consequently, we keep the edge uv with probability $1/(2n-1)$; conditioned on this, $M \setminus \{uv\}$ is uniformly random by our hypothesis. Conditioned on uv being removed from the matching, we re-match u and v using another random edge $xy \in M \setminus \{uv\}$. In this case, u and v get matched to a uniformly random pair of vertices $x, y \in V(K_{2n}) \setminus \{u, v\}$, as they should be. The rest of the matching $M \setminus \{uv, xy\}$ is uniformly random on $V(K_{2n}) \setminus \{u, v, x, y\}$ by our hypothesis. □

Lemma 3.2. *The resampling oracle $r_A(M)$ applied to a perfect matching satisfying event E_A does not cause any new event E_B such that $B \notin \Gamma^+(A)$.*

Proof. Observe that all the new edges that the resampling oracle adds to M are incident to some vertex matched by A. So if an event E_B was not satisfied before the operation and it is satisfied afterwards, it must be the case that B contains some edge not present in A but sharing a vertex with A. Hence, $A \cup B$ is not a matching and $A \sim B$. □

3.2 Rainbow matchings

Given an edge-coloring of K_{2n}, a perfect matching is called rainbow if each of its edges has a distinct color. Combinatorialists have been interested in rainbow matchings for many years. Achlioptas and Iliopoulos [1] showed how to find a rainbow matching in K_{2n} efficiently when each color appears on at most γn edges, $\gamma < \frac{1}{2e} \simeq 0.184$. In this short version of the paper, we achieve $\gamma = 0.1$; in the long version, we achieve $\gamma = 0.21$.

Theorem 3.3. *Given an edge-coloring of K_{2n} where each color appears on at most $n/10$ edges, a rainbow perfect matching exists and can be found in $O(n)$ resampling oracle calls with high probability.*

We apply our algorithm in the setting of uniformly random perfect matchings $M \subset K_{2n}$, with the following bad events: For every pair of edges e, f of the same color, E_{ef} occurs if $\{e, f\} \subset M$. If no bad event E_{ef} occurs then M is a rainbow matching. We also define the following dependency graph: $E_{ef} \sim E_{e'f'}$ unless e, f, e', f' are four disjoint edges. Note that this is more conservative than the dependency graph we considered in Section 3.1, where two events are only connected if they do not form a matching together. The more conservative definition will simplify our analysis. In any case, our resampling oracle is consistent with the dependency graph in the sense that resampling E_{ef} can only cause new events $E_{e'f'}$ such that $E_{ef} \sim E_{e'f'}$.

Lemma 3.4. *For any edge-coloring of K_{2n} such that every color appears at most $q = 0.1n$ times, the assumptions of Theorem 1.2 are satisfied with slack $\epsilon > 0.01$.*

Proof. We can apply the symmetric local lemma criterion instead of (GLL). We have

$$\Pr[\{e, f\} \subset M] = \frac{1}{(2n-1)(2n-3)} \approx \frac{1}{4n^2} =: p.$$

Consider the neighborhood of a bad event $\Gamma(E_{ef})$. It contains all events $E_{e'f'}$ such that there is some intersection among the edges e, f, e', f'. How many ways are there to choose e' and f' such that this holds?

There are 4 ways to choose the shared vertex, then $2n-1$ ways to choose the other endpoint of that edge, then $q-1$ ways to choose the other edge. So

$$|\Gamma(E_{ef})| \;\le\; 4(q-1)(2n-1) \;\le\; 0.8n^2 - 1 \;=:\; d.$$

By the symmetric local lemma, we need $pe(d+1) \le 1$, which is satisfied since $0.8e/3 < 1$. □

By Theorem 1.2, MaximalSetResample with the resampling oracle for matchings the will find a rainbow perfect matching in polynomial time with high probability. This proves Theorem 3.3.

References

[1] Dimitris Achlioptas and Fotis Iliopoulos. Random walks that find perfect objects and the Lovász local lemma. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 494–503, 2014.

[2] Paul Erdös and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal et al., editor, *Infinite and finite sets*, volume 10 of *Colloquia Mathematica Societatis János Bolyai*, pages 609–628. North-Holland, Amsterdam, 1975.

[3] Paul Erdös and Joel Spencer. The Lopsided Lovász Local Lemma and Latin transversals. *Discrete Applied Mathematics*, 30:151–154, 1991.

[4] Kashyap Kolipaka and Mario Szegedy. Moser and Tardos meet Lovász. In *Proceedings of STOC*, 2011.

[5] Lincoln Lu, Austin Mohr, and László Székely. Quest for negative dependency graphs. *Recent Advances in Harmonic Analysis and Applications*, 25:243–258, 2013.

[6] Austin Mohr. *Applications of the lopsided Lovász local lemma regarding hypergraphs*. PhD thesis, 2013.

[7] Robin A. Moser. A constructive proof of the Lovász local lemma. In *Proceedings of STOC*, 2009.

[8] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász Local Lemma. *Journal of the ACM*, 57(2), 2010.

[9] Joel Spencer. Asymptotic lower bounds for Ramsey functions. *Discrete Mathematics*, 20:69–76, 1977.