

Simple Local Computation Algorithms for the General Lovász Local Lemma

Dimitris Achlioptas
University of Athens
optas@di.uoa.gr

Themis Gouleakis *
University of Southern California
tgoule@mit.edu

Fotis Iliopoulos †
Institute for Advanced Study
fotios@ias.edu

Abstract

We consider the task of designing Local Computation Algorithms (LCA) for applications of the Lovász Local Lemma (LLL). LCA is a class of sublinear algorithms proposed by Rubinfeld et al. [38] that have received a lot of attention in recent years. The LLL is an existential, sufficient condition for a collection of sets to have non-empty intersection (in applications, often, each set comprises all objects having a certain property). The ground-breaking algorithm of Moser and Tardos [34] made the LLL fully constructive, following earlier results by Beck [7] and Alon [5] giving algorithms under significantly stronger LLL-like conditions. LCAs under those stronger conditions were given in [38], where it was asked if the Moser-Tardos algorithm can be used to design LCAs under the standard LLL condition. The main contribution of this paper is to answer this question affirmatively. In fact, our techniques yield LCAs for settings beyond the standard LLL condition.

*Research supported by NSF Award Numbers CCF-1650733, CCF-1733808, CCF-1740751, and IIS-1741137.

†This material is based upon work directly supported by the IAS Fund for Math and indirectly supported by the National Science Foundation Grant No. CCF-1900460. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is also supported by the National Science Foundation Grant No. CCF-1815328.

1 Introduction

The Lovász Local Lemma (LLL) [14] is a powerful tool of probabilistic combinatorics for establishing the existence of objects satisfying certain properties (constraints). As a probability statement, it asserts that given a family of “bad” events, if each bad event is individually not very likely and, in addition, is independent of all but a small number of other bad events, then the probability of avoiding all bad events is strictly positive. Given a collection of constraints, one uses the LLL to prove the existence of an object satisfying all of them (a perfect object) by considering, for example, the uniform measure on all candidate objects and defining one bad event for each constraint (containing all candidate objects that violate the constraint). Making the LLL constructive was the subject of intensive research for over two decades, during which several constructive versions were developed [7, 5, 31, 13, 42], but always under conditions stronger than those of the LLL. In a breakthrough work [33, 34], Moser and Tardos made the LLL constructive for any *product* probability measure (over explicitly presented variables). Specifically, they proved that whenever the LLL condition holds, their *Resample* algorithm, which repeatedly selects *any* occurring bad event and resamples all its variables according to the measure, quickly converges to a perfect object.

In this paper we consider the task of designing *Local Computation Algorithms* (LCA) for applications of the LLL. This is a class of sublinear algorithms proposed by Rubinfeld et al. in [38] that has received a lot of attention in the recent years [6, 19, 22, 26, 27, 28, 29, 37]. For an instance F , a local computation algorithm should answer in an online fashion, for any index i , the i -th bit of one of the possibly many solutions of F , so that the answers given are consistent with some specific solution of F . As an example, given a constraint satisfaction problem and a sequence of queries corresponding to variables of the problem, the algorithm should output a value assignment for each queried variable that agrees with some full assignment satisfying all constraints (assuming one exists).

The motivation behind the study of LCAs becomes apparent in the context of computations on massive data sets. In such a setting, inputs to and outputs from algorithms may be too large to handle within an acceptable amount of time. On the other hand, oftentimes only small portions of the output are required at any point in time by any specific user, in which case the use of a local computation algorithm is appropriate. We also note that LCAs can be seen as a generalization of several models such as local algorithms [43], locally decodable codes [44] and local reconstruction algorithms e.g., [4, 8, 10, 24, 39].

The algorithm we propose is simple and essentially corresponds to running the Moser-Tardos algorithm with a specific strategy for choosing which occurring bad event to resample. As an example, assume we are given a constraint satisfaction problem and a set of queries (variables) x_1, x_2, \dots, x_q . In this case, the algorithm first finds a satisfying assignment for the instance induced by the constraints within distance r of x_1 in the dependency graph, and then outputs the current value of x_1 . Then it considers variable x_2 and the instance of constraints within distance r of it, then x_3 and so on and so forth. Our key observation is that if the constraints within a ball of radius r around variable x are all satisfied after some step of the execution of the Moser-Tardos algorithm, then the probability that the algorithm needs to resample x in some subsequent step is exponentially small in r . We use this fact to show that if the LLL condition is satisfied, then we can choose r appropriately to get a sublinear time algorithm that makes no errors with high probability.

1.1 Related work in local computation algorithms

The original paper of Rubinfeld et al. [38] as well as the follow-up work of Alon et al. [6] provide LCAs for several problems, including applications of the LLL to k -SAT and hypergraph 2-coloring. The LCAs for LLL applications given in these works, though, are based on the earlier constructive versions of the LLL by Beck [7] and by Alon [5], thus requiring significantly stronger conditions than the (standard) LLL condition. Indeed, it was left as a major open question in [38] whether the Moser-Tardos algorithm can be used to design LCAs under the LLL condition. (Note also that, besides requiring stronger conditions, the

algorithms of [7, 5] are relatively involved compared to the Moser-Tardos algorithm.) We further discuss how our algorithm compares to the ones of [38, 6] in Section 1.2.1.

Moreover, there is a recent line of research on LLL in the distributed LOCAL model [11, 12, 16, 18] that often imply the existence of LCAs for various problems. However, these works also require stronger conditions than the standard LLL condition and the resulting LCAs are significantly more sophisticated than the algorithm we propose in this paper.

1.2 Our contributions

Our main contribution is to make the LLL *locally constructive*, i.e., to give a LCA under the LLL condition. Our techniques actually yield a LCA under more general recent conditions for the success of stochastic local search algorithms [1, 2, 21] that go beyond the variable setting of Moser and Tardos. For simplicity of exposition, though, we focus our presentation on the variable setting of Moser and Tardos, as it captures the great majority of LLL applications, and discuss the more general settings later. That is, we focus on constraint satisfaction problems $(\mathcal{X}, \mathcal{C})$, where \mathcal{X} is a set of variables and \mathcal{C} is a set of constraints over these variables. Given a product measure μ over \mathcal{X} , the LLL condition is said to be satisfied with ϵ -slack for the family of bad events induced by \mathcal{C} , if the “badness” of each bad event is bounded by $1 - \epsilon$ (see Section 2.1). Given an instance $(\mathcal{X}, \mathcal{C})$, we assume that each constraint entails at most $k = O(\text{polylog}|\mathcal{X}|)$ variables, and each variable is entailed by at most $d = O(\text{polylog}|\mathcal{X}|)$ constraints. Finally, a (t, s, δ) -LCA responds to each query in time t , using memory s , and makes no error with probability at least $1 - \delta$. An informal version of our main result can thus be stated as follows.

Theorem 1.1 (Informal Statement). *If $(\mathcal{X}, \mathcal{C}, \mu)$ satisfies the LLL conditions with ϵ -slack, then there exists an $(n^\beta, O(n), n^{-\gamma})$ -LCA for $(\mathcal{X}, \mathcal{C})$, for every $\beta, \gamma > 0$ such that $(1 + \gamma)/\beta < \log(1/(1 - \epsilon))/\log(kd)$.*

Theorem 1.1 gives a trade-off between the running time (per query) and the probability of error, while establishing that both decrease with the slack in the LLL conditions. Moreover, as we will see, if we know beforehand the total number of queries to our algorithm, then the condition of Theorem 1.1 can be significantly improved. (We stress that the latter is a feature of our results which only adds flexibility to the original definition of LCAs and does not impose any restrictions, as the user can always choose to not introduce a limitation on the number of queries. However, when dealing with large instances such limitations are natural and/or even unavoidable.)

Using our general results we design LCAs for the following problems, chosen to highlight different features of our results. As we will see formally in Section 2.2, our results apply to constraint satisfaction problems of large size, i.e., we assume that the number of variables is sufficiently large. This mild assumption is essentially inherent in the model of local computation algorithms.

1.2.1 k -SAT

Gebauer, Szabó and Tardos [17] used the LLL to prove that any k -CNF formula where every variable appears in at most d clauses is satisfiable if $d(k + 1) \leq 2^{k+1}/e$ and, moreover, that this is asymptotically tight in k . We show the following.

Theorem 1.2. *Let ϕ be a k -CNF formula on n variables with m clauses where every variable appears in at most d clauses.*

- (a) *Suppose that $[d(k + 1)]^{1+\eta} \leq 2^{k+1}/e$, for some constant $\eta > 0$. For every $\alpha, \beta, \gamma > 0$ such that $(\alpha + \gamma)/\beta < \eta$, there exists a $(n^\beta, O(n^{\min\{1, \alpha+\beta\}}), n^{-\gamma})$ -LCA for ϕ that answers up to n^α queries.*
- (b) *Suppose that $d(k + 1) \leq (1 - \epsilon)2^{k+1}/e$, for some constant $\epsilon > 0$. Then, for every $\beta, c > 0$, there exists a $(n^\beta, n^\beta \log^c(n), \log^{-c}(n))$ -LCA for ϕ that answers up to $\log^c(n)$ queries.*

For comparison, the work of Rubinfeld et al. [38] gave a LCA for k -CNF formulas only when there exist k_1, k_2, k_3 such that $k_1 + k_2 + k_3 = k$ and

$$\begin{aligned} 8d(d-1)^3(d+1) &< 2^{k_1} \\ 8d(d-1)^3(d+1) &< 2^{k_2} \\ e(d+1) &< 2^{k_3} . \end{aligned}$$

Notably, the LCA of [38] is logarithmic in time and space [6]. Unfortunately, the techniques of [6] that allow for space-efficient local algorithms are tailored to the LLL-algorithm of Alon [5] and do not appear to be compatible with our results.

More specifically, Alon et al. [6] are able to exploit a technique introduced in [35] that considers a random permutation of the input and feeds it to the algorithm in that order. In this way, they can use a pseudo-random generator in order to encode that permutation using logarithmic space. However, the successful application of this technique crucially relies on the fact that the algorithm in [5], which is being simulated, can afford to sample each variable exactly once during the execution. (An additional assumption, which we do not make in this paper, is that each variable should be contained in a constant number of clauses.) On the contrary, the Moser-Tardos algorithm works for the more general LLL conditions at the expense of the aforementioned property, which no longer holds. That is, it needs an explicit assignment of all variables at every point during the execution in order to know the set of currently violated clauses, while the algorithm in [5] can work only with partial value assignments until the very end of its execution, since each variable is assigned a value once. Therefore, a simple permutation of the input cannot capture the entire resampling sequence of the Moser-Tardos algorithm, which potentially involves multiple resamplings of each variable. Also, the constraint that only violated clauses are resampled makes certain resampling sequences invalid, and this even depends on the values sampled so far at any point of the execution, which is not the case in Alon's algorithm [5].

1.2.2 Coloring Graphs

In graph vertex coloring one is given a graph $G(V, E)$ and the goal is to find a mapping of V to a set of q colors so that no edge in E is monochromatic. The *chromatic number*, $\chi(G)$, of G is the smallest integer for which this is possible. Trivially, if the maximum degree of G is Δ , then $\chi(G) \leq \Delta + 1$. Molloy and Reed [30] proved that this can be significantly improved for graphs where the neighborhood of every vertex is bounded away from being a clique.

Theorem 1.3 ([30]). *There exists Δ_0 such that if G has maximum degree $\Delta > \Delta_0$ and the neighborhood of every vertex of G contains at most $\binom{\Delta}{2} - B$ edges, where $B \geq \Delta \log^4 \Delta$, then $\chi(G) \leq \Delta + 1 - B/(e^6 \Delta)$.*

Theorem 1.3 is a sophisticated application of the LLL. Our results imply local algorithms for finding the colorings promised by Theorem 1.3 that exhibit no trade-off between speed and accuracy, in the sense that for large enough n both constants β, γ , below, can be made arbitrarily small.

Theorem 1.4. *Let G be any graph on n vertices, m edges, and maximum degree Δ satisfying the conditions of Theorem 1.3. For every $\beta, \gamma > 0$ there exists a $(n^\beta, O(n), n^{-\gamma})$ -local algorithm for coloring G using $\Delta + 1 - B/(e^6 \Delta)$ colors.*

1.2.3 Non-Uniform Hypergraph Coloring

Our results can also handle applications of the LLL in non-uniform settings, i.e., where the probabilities of bad events may vary significantly. For example, it is known that a hypergraph \mathcal{H} with minimum edge size

at least 3 where every vertex lies in at most Δ_i edges of size i is 2-colorable, if $\sum_i \Delta_i 2^{-i/2} \leq \frac{1}{6\sqrt{2}}$ (see Theorem 19.2 in [32]).

Using our main theorem we can design a local algorithm for this problem when the number of queries is polylogarithmic. (Our main result, as well as extensions of the techniques in [38], can be applied to give local algorithms with no restriction on the number of queries, but under significantly stronger assumptions for the Δ_i . In particular, in these cases the fact that constraints corresponding to large hyperedges are “easier” to fix cannot be captured.)

Theorem 1.5. *Fix $\epsilon > 0$ arbitrarily small and $D > 0$ arbitrarily large. Let $\mathcal{H}_{\epsilon,D}$ be the set of hypergraphs with minimum edge size at least 3, where each vertex lies in at most $\Delta_i \leq D$ edges of size i such that*

$$\sum_{i \geq 3} \Delta_i 2^{-i/2} \leq \frac{1 - \epsilon}{6\sqrt{2}} . \quad (1)$$

For every $\beta, c > 0$ there exists a $(n^\beta, O(n^\beta \log^c n), \log^{-c} n)$ -LCA for 2-coloring hypergraphs in $\mathcal{H}_{\epsilon,D}$ that answers up to $\log^c(n)$ queries.

2 Background

2.1 The Lovász Local Lemma

To prove that a set of objects Ω contains at least one element satisfying a collection of constraints, we introduce a probability measure μ on Ω , thus turning the objects violating each constraint into a bad event.

General LLL. *Let (Ω, μ) be a probability space and $\mathcal{A} = \{A_1, A_2, \dots, A_m\}$ be a set of m (bad) events. For each $i \in [m]$, let $D(i) \subseteq [m] \setminus \{i\}$ be such that $\mu(A_i \mid \cap_{j \in S} \overline{A_j}) = \mu(A_i)$ for every $S \subseteq [m] \setminus (D(i) \cup \{i\})$. If there exist positive real numbers $\{\psi_i\}_{i=1}^m$ such that for all $i \in [m]$,*

$$\frac{\mu(A_i)}{\psi_i} \sum_{S \subseteq D(i) \cup \{i\}} \prod_{j \in S} \psi_j \leq 1 , \quad (2)$$

then the probability that none of the events in \mathcal{A} occurs is at least $\prod_{i=1}^m 1/(1 + \psi_i) > 0$.

Remark 2.1. *Condition (2) above is equivalent to the more well-known form $\mu(A_i) \leq x_i \prod_{j \in D(i)} (1 - x_j)$, where $x_i = \psi_i/(1 + \psi_i)$. As we will see, formulation (2) facilitates refinements. To see the equivalence, notice that since $x_i = 0$ is uninteresting, we may assume $x_i \in (0, 1)$. Taking $\psi_i > 0$, setting $x_i = \psi_i/(1 + \psi_i) \in (0, 1)$, and simplifying, the condition becomes $\mu(A_i) \prod_{j \in \{i\} \cup D(i)} (1 + \psi_j) \leq \psi_i$. Opening up the product yields (2).*

Definition 2.1. *We say that the general LLL condition holds with ϵ -slack if the righthand side of (2) is bounded by $1 - \epsilon$ for every $i \in [m]$.*

Let G be the digraph over the vertex set $[m]$ having an arc from each $i \in [m]$ to each element of $D(i) \cup \{i\}$. We call such a graph a *dependency graph*. Therefore, at a high level, the LLL states that if there exists a sparse dependency graph and each bad event is not too likely, then we can avoid all bad events with positive probability.

2.2 Local Computation Algorithms

Definition 2.2. For any input x , define the set $F(x) = \{y : y \text{ is a valid solution for input } x\}$. The search problem, given x , is to find any $y \in F(x)$. We use $\ell = |x|$ to denote the length of the input.

Our definition of LCA algorithms is almost identical to the one of [38], the only difference being that it is more flexible in the sense that it also takes as a parameter the number of queries to the algorithm.

Local Algorithms. Let $F(x)$ be as in Definition 2.2. A (q, t, s, δ) -local computation algorithm \mathcal{A} is a (randomized) algorithm which satisfies the following: \mathcal{A} receives a sequence i_1, i_2, \dots of up to $q(\ell)$ queries one by one; upon receiving each query i_j it produces an output o_j ; with probability at least $1 - \delta(\ell)$, there exists $y \in F(x)$ such that $o_j = y_{i_j}$ for every j . \mathcal{A} has access to a random tape and local computation memory on which it can perform current computations, as well as store and retrieve information from previous computations. We assume that the input x , the local computation tape and any random bits used are all presented in the RAM world model, i.e., \mathcal{A} is given the ability to access a word of any of these in one step. The running time of \mathcal{A} on any query is at most $t(\ell)$, which is sublinear in ℓ , and the local computation memory of \mathcal{A} is at most $s(\ell)$. Unless stated otherwise, we always assume that the error parameter $\delta(\ell)$ is at most some constant, say, $\frac{1}{3}$. We say that \mathcal{A} is a strongly local computation algorithm if both $t(\ell), s(\ell)$ are upper bounded by $\log^c \ell$ for some constant c .

As we have already mentioned, in this paper we will be interested in local computation algorithms for constraint satisfaction problems $(\mathcal{X}, \mathcal{C})$, where \mathcal{X} is a set of variables and \mathcal{C} is a set of constraints over these variables. To simplify the statement of our results, whenever we say there exists a (q, t, s, δ) -local computation algorithm for $(\mathcal{X}, \mathcal{C})$ we mean that there exists n_0 and an algorithm \mathcal{A} such that \mathcal{A} is a (q, t, s, δ) -local computation algorithm when the input is restricted to instances of $(\mathcal{X}, \mathcal{C})$ such that $|\mathcal{X}| \geq n_0$. In other words, our results apply to constraint satisfaction problems of large size.

3 Statement of Results

For simplicity, we will present our results and techniques for the general LLL in the *variable setting*, i.e., the setting considered by Moser and Tardos [34]. In Section B of the Appendix we discuss how our techniques can be adapted to capture improved LLL criteria and generalized to settings beyond the one of [34].

The Setting. Let $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ be a set of variables with domains D_1, \dots, D_n . We define $\Omega = \prod_{i=1}^n D_i$ to be the set of possible value assignments for the variables of \mathcal{X} , and we sometimes refer to its elements as *states*. We also consider a set of *constraints* $\mathcal{C} = \{c_1, c_2, \dots, c_m\}$. Each constraint c_i is associated with a set of variables $\text{var}(i) \subseteq \mathcal{X}$ and corresponds to a set of forbidden value assignments for these variables, i.e., that *violate* the constraint.

We consider an arbitrary product probability measure μ over the variables of \mathcal{X} along with the family of bad events $\mathcal{A} = \{A_1, \dots, A_m\}$, where A_i corresponds to the states in Ω that violate c_i . The *dependency graph* $G = G(V, E)$ related to $(\Omega, \mu, \mathcal{A})$ is the graph with vertex set $V = [m]$ and edge set $E = \{(i, j) : \text{var}(i) \cap \text{var}(j) \neq \emptyset\}$. (Notice that since this dependence relationship is always symmetric, we have a graph instead of a digraph.) The *neighborhood* of an event A_i is defined as $D(i) = \{j : (i, j) \in E\}$ and notice that A_i is mutually independent of $\mathcal{A} \setminus (D(i) \cup \{i\})$. Finally, for $i, j \in [m]$ we denote by $\text{dist}(i, j)$ the length of a shortest path between i and j in G .

Assumptions. We will make computational assumptions similar to [38] (but less restrictive). For a variable x , we let $N(x)$ denote the set of constraints that contain x and define $d = \max_{x \in \mathcal{X}} |N(x)|$. We further define an $n \times m$ incidence matrix \mathcal{M} such that, for any variable x and constraint c , $\mathcal{M}_{x,c} = 1$ if $c \in N(x)$.

$N(x)$ and $\mathcal{M}_{x,c} = 0$, otherwise. The input constraint satisfaction problem $(\mathcal{X}, \mathcal{C})$ will be represented by its variable-constraint incidence matrix \mathcal{M} . Let $k = \max_{i \in [m]} |\text{var}(i)|$ denote the maximum number of variables associated with a constraint. We will also assume that $d, k \in O(\log^c(n))$ for some constant $c \geq 0$, which means that matrix \mathcal{M} is necessarily very sparse. Therefore, we also assume that the matrix \mathcal{M} is implemented via linked lists for each row (i.e., variable x) and each column (i.e., constraint c) and that

$$\max_{i \in [m]} \psi_i = O(n^\lambda)$$

for some constant $\lambda > 0$. (Here the set of parameters $\{\psi_i\}_{i=1}^m$ is the one used in the LLL condition (2). We note that in most applications $\max_{i \in [m]} \psi_i = O(1)$.) We can now state our main result precisely.

Theorem 3.1. *Assume that $(\mathcal{X}, \mathcal{C}, \mu)$ satisfies the Lovász Local Lemma conditions with ϵ -slack and define $\zeta = \zeta(\epsilon, k, d) = \log(1/(1 - \epsilon)) / \log(kd)$. Let $\alpha, \beta, \gamma > 0$ be constants such that $\beta\zeta > \alpha + \gamma + \lambda$. Then there exists a $(n^\alpha, n^\beta, O(n^{\min\{1, \alpha + \beta\}}), n^{-\gamma})$ -local computation algorithm for $(\mathcal{X}, \mathcal{C})$.*

Remark 3.1. *If the number of queries is $O(\text{polylog}(n))$, the probability of error is $\Omega\left(\frac{1}{\text{polylog}(n)}\right)$, and $k, d = O(1)$, then if the LLL conditions hold with ϵ -slack for some fixed constant $\epsilon > 0$, then for any arbitrarily small constant $\beta > 0$ there exists a LCA that takes n^β time per query and uses $O(n^\beta \text{polylog}(n))$ space (for all sufficiently large n).*

4 Our Algorithm

In this section we describe our algorithm formally as well as the main idea behind its analysis.

To describe our algorithm, we first recall the algorithm of Moser and Tardos as well as a couple useful facts about its performance.

-
- 1: **procedure** RESAMPLE($\mu, \mathcal{C}, \mathcal{X}$)
 - 2: Sample all variables in \mathcal{X} according to μ
 - 3: **while** violated constraints exist **do**
 - 4: Pick an arbitrary violated constraint c_i
 - 5: (Re)sample every variable in $\text{var}(i)$ according to μ
-

Notice that the most expensive operation of the Moser-Tardos algorithm is searching for constraints which are currently violated. In [41], a simple optimization is suggested to reduce this cost, which will be helpful to us as well. The idea is to keep a stack which, at every step, contains all the currently violated constraints. To do that, initially, we go over all the constraints and add the violated ones into the stack. Then, each time we resample a constraint c , in order to update the stack, we are only required to check the constraints that share variables with c to determine whether they became violated, in which case, we add them to the stack. The main benefit of maintaining this data structure is that we avoid going over the whole set of constraints at each step. In particular, using this method, we only have to put a $O(kd)$ amount of work after each resampling. This method is usually referred to as Depth-First MT.

In the following, when we say “apply the Depth-First MT algorithm for at most t steps”, we mean that we apply the Resample algorithm above for at most t steps, *without* performing the initial sampling of the variables of \mathcal{X} (all relevant variables will have been assigned values by other means).

For $i \in [m]$ and $r \geq 0$, let $\text{Ball}(i, r) = \{j \in [m] : \text{dist}(i, j) \leq r\}$ be the elements of $[m]$ whose distance to i in G is at most r . Furthermore, for a variable x we denote by $\mathcal{I}(x, r)$ the sub-problem of $(\mathcal{X}, \mathcal{C})$ induced by the constraints in $\bigcup_{c_i \ni x} \text{Ball}(i, r)$ and the variables they contain. Notice that if $(\mathcal{X}, \mathcal{C})$ satisfies the LLL conditions, then $\mathcal{I}(x, r)$ does as well for any x and r . We are now ready to describe our meta-algorithm,

that takes as input q, t, δ and ϵ , i.e., the number of queries, the desired upper bounds on the running time per query, the probability of error, and the slack, respectively. For the sake of brevity, we slightly abuse notation and for $i \in [q]$ denote by x_i the variable of the i -th query.

```

1: procedure RESPOND TO QUERIES( $q, t, \delta, \epsilon$ )
2:    $\eta \leftarrow \max_{x \in \mathcal{X}} \sum_{c_j \ni x} \psi_j$ 
3:    $r \leftarrow \log(q\eta/(\delta - q/n^2))/\log(1/(1 - \epsilon))$ 
4:    $S \leftarrow \emptyset$ 
5:   for  $i = 1$  to  $q$  do
6:     Resample each variable in  $\mathcal{I}(x_i, r) \setminus S$   $\triangleright x_i, i \in [q]$ , is the  $i$ -th query.
7:      $S \leftarrow S \cup \mathcal{I}(x_i, r)$ 
8:     Apply the Depth-First MT algorithm to  $\mathcal{I}(x_i, r)$  for at most  $t$  steps
9:     if a satisfying assignment for  $\mathcal{I}(x_i, r)$  is found then
10:       Output the value of  $x_i$ 
11:     else
12:       Abort

```

The main idea behind our algorithm comes from the following property of the Moser-Tardos algorithm. Assume that in an execution of the Moser-Tardos algorithm, in the current step, every constraint in a ball of radius r around variable x is satisfied. We prove that the probability that the algorithm will have to resample x in a later step drops exponentially fast with r . In other words, for large enough r , the current value of x is a good guess for the value of x in the final output. To exploit this fact, we use that in the Moser-Tardos algorithm the strategy for choosing which violated constraint to resample can be arbitrary, so that we get an LCA as follows: upon receiving query (variable) x_i , our algorithm tries to create a large ball of satisfied constraints around x_i , by executing the Moser-Tardos algorithm with a strategy prioritizing the constraints in the ball. Naturally, then the radius of the ball governs the trade-off between speed and accuracy.

5 Proof of Theorem 3.1

In this section we present the proof of Theorem 3.1. Clearly, the running time of our algorithm on any query is at most t . Further, the local computation memory it requires is dictated by the number of variables it resamples (since it has to store the “current” value of every such variable), and the space required for the stack in the application of the Depth-First MT. The former is at most linear while the latter is sublinear. Therefore, we get a $O(n)$ bound overall. (As it will become clear later, when the number of queries is limited, i.e., when $\alpha < 1 - \beta$, then the memory required is $O(n^{\alpha+\beta})$, i.e., sublinear.)

In the rest of the proof we will focus on bounding the probability that our algorithm makes an error.

Observe that Line 6 allows us to see the execution of our algorithm as a prefix of a *complete* execution of the Moser-Tardos algorithm from a random initial state. The probability that our algorithm makes an error is bounded by the sum of (i) the probability that our algorithm ever aborts in Line 12; (ii) the probability that the complete execution of the Moser-Tardos algorithm resamples a (queried) variable after our algorithm has returned its response for it. We start by bounding the former, since it’s a more straightforward task.

5.1 Bounding the Running Time as a Function of the Radius

To bound the probability that our algorithm aborts in Line 12 we will use Theorem 5.1 below, a direct corollary of the main result in [2], bounding the running time of the Depth-First MT algorithm from an

arbitrary initial state. Let

$$\xi = \max_{i \in [m]} \log(1 + \psi_i) .$$

Theorem 5.1. *If the LLL conditions hold with ϵ slack, then the probability that the MT algorithm starting at an arbitrary initial state has not terminated after $(n + m\xi)/\log(1/(1 - \epsilon)) + s$ steps is at most $(1 - \epsilon)^s$.*

There are two reasons why we need to use Theorem 5.1 instead of the original running time bound of Moser and Tardos [34]. The first and most important one, is that the original bound assumes that the initial state of the algorithm is selected according to the product measure μ . However, when we run the MT algorithm in response to a query for variable x_i , some of the variables of $\mathcal{I}(x_i, r)$ may have been resampled multiple times in earlier executions of the for loop and, thus, be correlated with each other. The second reason is that Theorem 5.1 exploits the slack in the LLL conditions to ensure that the algorithm terminates fast with high probability and not just in expectation.

We are now ready to give a tail-bound for the running time of our algorithm on a single query, as a function of the radius r . Recall that each constraint contains at most k variables and that each variable is contained in at most d constraints and k, d are at most polylogarithmic. We use $\tilde{O}(\cdot)$ notation to hide polylogarithmic factors in n, m .

Lemma 5.2. *Let $T_0 = (kd)^r(\xi/\log(1/(1 - \epsilon)))$. Step 8 takes more than $\tilde{O}(T_0 + s)$ time with probability at most $(1 - \epsilon)^s$.*

Proof. Let us first derive an upper bound B on the number of constraints (and of variables) in $\text{Ball}(i, r)$. Since the maximum degree of the dependency graph is at most kd and the subgraph that maximizes the number of constraints inside $\text{Ball}(i, r)$ is the full kd -ary tree of depth r , we see that $|\text{Ball}(i, r)| = O((kd)^{r+1}) = \tilde{O}((kd)^r)$, since k, d are at most poly-logarithmic. Thus, we can assume that $B = \tilde{O}((kd)^r)$.

The running of our algorithm on query x_i consists of computing the sub-problem $\mathcal{I}(x_i, r)$ and then applying Depth-First MT to it. By “computing the sub-problem $\mathcal{I}(x_i, r)$ ” we mean creating an incidence matrix $\mathcal{M}_{i,r}$ that corresponds to the subgraph of the dependency graph associated with $\mathcal{I}(x_i, r)$, represented similarly to \mathcal{M} via linked lists. To perform this task we can do a Breadth First Search starting from a node j such that $c_j \ni v_i$ for depth r . This takes $\tilde{O}((kd)^r)$ time, since we can find the neighbors of a constraint in the dependency graph in poly-logarithmic time and the subgraph of the dependency graph that corresponds to $\mathcal{I}(x_i, r)$ has at most $Bkd = \tilde{O}((kd)^r)$ edges.

For the application of Depth-First MT to $\mathcal{I}(x_i, r)$, Theorem 5.1 asserts that if $T_0 = (B+B\xi)/\log(1/(1 - \epsilon))$, then the probability that a satisfying assignment is not found after $T_0 + s$ resamplings is at most $(1 - \epsilon)^s$. Recalling that $B = \tilde{O}((kd)^r)$, that the amount of work per resampling is $O(kd)$, and that both k and d are polylogarithmic and adding the bound above for formulating each subproblem, concludes the proof. \square

5.2 Bounding the Probability of Revising a Variable as a Function of the Radius

To bound the probability of error of our algorithm we first need to recall a key element of the analysis of [34].

5.2.1 Witness Trees

We denote by $\Sigma = \sigma_1 \xrightarrow{w_1} \sigma_2 \xrightarrow{w_2} \sigma_3 \xrightarrow{w_3} \dots$ the random variable that equals the *trajectory* of an execution of the Moser-Tardos algorithm, where, for each $i \geq 1$, $\sigma_i \in \Omega$ denotes the i -th state of the trajectory and $w_i \in [m]$ the index of the bad event resampled. We also call the random variable $W(\Sigma) = (w_1, w_2, \dots)$ the *witness sequence* of Σ .

We first recall the definition of witness trees from [34], while slightly reformulating to fit our setting. A witness tree $\tau = (T, \ell_T)$ is a finite rooted, unordered, tree T along with a labelling $\ell_T : V(T) \rightarrow [m]$

of its vertices with indices of bad events such that the children of a vertex $v \in V(T)$ receive labels from $D(\ell(v)) \cup \{\ell(v)\}$. To lighten notation, we will sometimes write (v) to denote $\ell(v)$ and $V(\tau)$ instead of $V(T)$. Given a witness sequence $W = (w_1, w_2, \dots, w_t)$ we associate with each $i \in [t]$ a witness tree $\tau_W(i)$ constructed in i steps as follows: let $\tau_W^i(i)$ be an isolated vertex labelled by w_i ; then, going backwards for each $j = i-1, i-2, \dots, 1$, if there is a vertex $v \in \tau_W^{j+1}(i)$ such that $w_j \in D((v)) \cup \{(v)\}$, then among those vertices we choose the one having maximum distance from the root (breaking ties arbitrarily) and attach a new child vertex u to v that we label w_j to get $\tau_W^j(i)$. If there is no such vertex v then $\tau_W^{j+1}(i) = \tau_W^j(i)$. Finally, $\tau_W(i) = \tau_W^1(i)$.

We will say that a witness tree τ occurs in a trajectory with witness sequence $W = (w_1, w_2, w_3, \dots)$, if there is $k \geq 1$ such that $\tau_W(k) = \tau$. Finally, we use the notation $\Pr[\cdot]$ to refer to the probability of events in the probability space induced by the execution of the Moser-Tardos algorithm.

Lemma 5.3 (The witness tree lemma [34]). *For every witness tree τ , $\Pr[\tau] \leq \prod_{v \in V(\tau)} \mu(A(v))$.*

5.2.2 The Analysis

Let E_i be the event that the complete execution of the Moser-Tardos algorithm ever resamples query variable x_i after the time, t_i , that it returned a response for it. Let c_j be a constraint that contains x_i and let $E_{i,j} \subseteq E_i$ denote the event that constraint c_j is resampled after t_i . Clearly, $E_i \subseteq \bigcup_{c_j \ni x_i} E_{i,j}$. The key insight is that in order for $E_{i,j}$ to occur, it should be that at least r constraints that form a path in the dependency graph which ends in j must have been resampled after t_i . This is because, by the nature of our algorithm, right after step t_i , every constraint in $\text{Ball}(i, r)$ is satisfied. This implies that the (first) resampling of the bad event A_j that corresponds to event $E_{i,j}$ occurring will be associated with a witness tree of size at least r . Thus, if r is large, $E_{i,j}$ is unlikely. Lemma 5.4 makes this idea rigorous.

Lemma 5.4. *Let $\mathcal{W}_{j,s}$ denote the set of all witness trees of size at least s whose root is labelled by j . Then,*

$$\sum_{\tau \in \mathcal{W}_{j,s}} \Pr[\tau] \leq \psi_j (1 - \epsilon)^s .$$

We prove Lemma 5.4 in Subsection 5.2.3. Using it, we can show the following.

Lemma 5.5. *Let $\eta = \max_{x \in \mathcal{X}} \sum_{c_j \ni x} \psi_j$. If*

$$r \geq \frac{\log\left(\left(\delta - \frac{q}{n^2}\right)^{-1} q \eta\right)}{\log(1/(1-\epsilon))} ,$$

then the probability that our algorithm answers at least one query incorrectly is at most $\delta - \frac{q}{n^2}$.

Proof. Combining Lemma 5.4 with our observation regarding the minimum size of witness trees related to event $E_{i,j}$, we obtain

$$\Pr[E_{i,j}] \leq \sum_{\tau \in \mathcal{W}_{j,r}} \Pr[\tau] \leq \psi_j (1 - \epsilon)^r .$$

Thus, taking $r \geq \frac{\log\left(\left(\delta - \frac{q}{n^2}\right)^{-1} q \eta\right)}{\log(1/(1-\epsilon))}$ and applying the union bound we obtain

$$\begin{aligned} \Pr \left[\bigcup_{i \in [q]} \bigcup_{c_j \ni x_i} E_{i,j} \right] &\leq (1 - \epsilon)^r \sum_{i=1}^q \sum_{c_j \ni x_i} \psi_j \\ &\leq q \eta (1 - \epsilon)^r \leq \delta - \frac{q}{n^2} . \end{aligned} \tag{3}$$

□

5.2.3 Proof of Lemma 5.4

A typical argument used in the algorithmic LLL literature to estimate sums over sets of witness trees, such as the sum in the statement of Lemma 5.4, is to consider a Galton-Watson branching process that produces each witness tree in the set of interest (and perhaps other trees) with positive probability. The idea is to then relate the probability of the branching process generating each tree with the probability that the tree occurs in the algorithm and exploit that the sum of the probabilities in the process is, by definition, bounded by 1.

Lemma 5.6 ([34]). *Let \mathcal{W}_j denote the set of witness trees whose root is labeled by j . There exists a branching process that outputs each witness tree $\tau \in \mathcal{W}_j$ with probability*

$$p_\tau = \psi_j^{-1} \prod_{v \in V(\tau)} \frac{\psi(v)}{\sum_{S \subseteq D((v)) \cup \{v\}} \prod_{f \in S} \psi_f}.$$

Observe that since $\mathcal{W}_{j,s} \subseteq \mathcal{W}_j$, Lemma 5.6 implies that

$$\psi_j \geq \psi_j \sum_{\tau \in \mathcal{W}_{j,s}} p_\tau = \sum_{\tau \in \mathcal{W}_{j,s}} \prod_{v \in V(\tau)} \frac{\psi(v)}{\sum_{S \subseteq D((v)) \cup \{v\}} \prod_{f \in S} \psi_f}. \quad (4)$$

Lemma 5.3 implies (6) below, the fact that the LLL conditions hold with ϵ slack implies (7), the fact that every witness tree in $\mathcal{W}_{j,s}$ has size at least s implies (8), while inequality (4), finally, implies (9).

$$\sum_{\tau \in \mathcal{W}_{j,s}} \Pr[\tau] \quad (5)$$

$$\leq \sum_{\tau \in \mathcal{W}_{j,s}} \prod_{v \in V(\tau)} \mu((v)) \quad (6)$$

$$\leq \sum_{\tau \in \mathcal{W}_{j,s}} \prod_{v \in V(\tau)} \frac{(1-\epsilon)\psi(v)}{\sum_{S \subseteq D((v)) \cup \{v\}} \prod_{f \in S} \psi_f} \quad (7)$$

$$\leq (1-\epsilon)^s \sum_{\tau \in \mathcal{W}_{j,s}} \prod_{v \in V(\tau)} \frac{\psi(v)}{\sum_{S \subseteq D((v)) \cup \{v\}} \prod_{f \in S} \psi_f} \quad (8)$$

$$\leq (1-\epsilon)^s \psi_j. \quad (9)$$

5.3 Concluding the Proof

Recall that $\eta = \max_{x \in \mathcal{X}} \sum_{c_i \ni x} \psi_i$, that $\xi = \max_{i \in [m]} \log(1 + \psi_i)$, and that t denotes the required upper bound on the running time of our algorithm on a single query. Lemma 5.2 and Lemma 5.5 imply that there exists $C = \tilde{O}(1)$ such that if

$$r \in \left[\frac{\log \left(\left(\delta - \frac{q}{n^2} \right)^{-1} q \eta \right)}{\log(1/(1-\epsilon))}, \frac{\log \left(\frac{t-s}{\xi^C} \log \frac{1}{1-\epsilon} \right)}{\log(kd)} \right], \quad (10)$$

where $s = \frac{2 \log n}{\log(1/(1-\epsilon))}$, then the probability that the algorithm aborts in Line 12 or responds inaccurately on any query is at most $\frac{1}{n^2} + \left(\delta - \frac{q}{n^2} \right) \leq \delta$.

Recall that $\max_{i \in [m]} \psi_i = O(n^\lambda)$ and that $\zeta = \log(1/(1-\epsilon))/\log kd$. It is not hard to see that if $q = n^\alpha$, $t = n^\beta$, $\delta = n^{-\gamma}$ and $\beta\zeta > \alpha + \gamma + \lambda$, then the interval in (10) is non-empty for large enough n , concluding the proof of Theorem 3.1. The proof of Remark 3.1 is very similar.

References

- [1] Dimitris Achlioptas and Fotis Iliopoulos. Random walks that find perfect objects and the Lovász local lemma. *J. ACM*, 63(3):22:1–22:29, July 2016.
- [2] Dimitris Achlioptas, Fotis Iliopoulos, and Vladimir Kolmogorov. A local lemma for focused stochastic algorithms. To appear in *SIAM Journal on Computing*. Preprint at *arXiv:1805.02026*.
- [3] Dimitris Achlioptas, Fotis Iliopoulos, and Alistair Sinclair. Beyond the Lovász local lemma: Point to set correlations and their algorithmic applications. To appear in *Proceedings of IEEE FOCS, 2019*. Preprint at *arXiv:1805.02026*.
- [4] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, April 2008.
- [5] Noga Alon. A parallel algorithmic version of the local lemma. *Random Struct. Algorithms*, 2(4):367–378, 1991.
- [6] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1132–1139. Society for Industrial and Applied Mathematics, 2012.
- [7] József Beck. An algorithmic approach to the Lovász local lemma. I. *Random Structures Algorithms*, 2(4):343–365, 1991.
- [8] Arnab Bhattacharyya, Elena Grigorescu, Madhav Jha, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. *SIAM Journal on Discrete Mathematics*, 26(2):618–646, 2012.
- [9] Rodrigo Bissacot, Roberto Fernández, Aldo Procacci, and Benedetto Scoppola. An improvement of the Lovász local lemma via cluster expansion. *Combinatorics, Probability & Computing*, 20(5):709–719, 2011.
- [10] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing, STOC '90*, pages 73–83, New York, NY, USA, 1990. ACM.
- [11] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2633–2652. SIAM, 2018.
- [12] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the Lovász local lemma and graph coloring. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 134–143. ACM, 2014.
- [13] Artur Czumaj and Christian Scheideler. Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász local lemma. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 2000)*, pages 30–39, 2000.
- [14] Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. II*, pages 609–627. Colloq. Math. Soc. János Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.

- [15] Paul Erdős and Joel Spencer. Lopsided Lovász local lemma and latin transversals. *Discrete Applied Mathematics*, 30(2-3):151–154, 1991.
- [16] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma, and the complexity hierarchy. *arXiv preprint arXiv:1705.04840*, 2017.
- [17] Heidi Gebauer, Tibor Szabó, and Gábor Tardos. The local lemma is tight for SAT. In Dana Randall, editor, *SODA*, pages 664–674. SIAM, 2011.
- [18] Mohsen Ghaffari, David G Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 662–673. IEEE, 2018.
- [19] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 784–797. ACM, 2017.
- [20] David G. Harris and Aravind Srinivasan. A constructive algorithm for the Lovász local lemma on permutations. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 907–925. SIAM, 2014.
- [21] Nicholas J. A. Harvey and Jan Vondrák. An algorithmic proof of the Lovász local lemma via resampling oracles. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1327–1346. IEEE Computer Society, 2015.
- [22] Avinatan Hassidim, Yishay Mansour, and Shai Vardi. Local computation mechanism design. *ACM Transactions on Economics and Computation (TEAC)*, 4(4):21, 2016.
- [23] Fotis Iliopoulos. Commutative algorithms approximate the lll -distribution. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPICs*, pages 44:1–44:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- [24] M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 433–442, Oct 2011.
- [25] Vladimir Kolmogorov. Commutativity in the algorithmic Lovász local lemma. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 780–787. IEEE Computer Society, 2016.
- [26] Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local algorithms for sparse spanning graphs. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Christopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPICs*, pages 826–842. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.
- [27] Reut Levi, Dana Ron, and Ronitt Rubinfeld. A local algorithm for constructing spanners in minor-free graphs. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation,*

- Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPICs*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- [28] Yishay Mansour, Aviad Rubinfeld, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2012.
- [29] Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 260–273. Springer, 2013.
- [30] Michael Molloy and Bruce Reed. A bound on the strong chromatic index of a graph. *journal of combinatorial theory, Series B*, 69(2):103–109, 1997.
- [31] Michael Molloy and Bruce Reed. Further algorithmic aspects of the local lemma. In *STOC '98 (Dallas, TX)*, pages 524–529. ACM, New York, 1999.
- [32] Michael Molloy and Bruce Reed. *Graph colouring and the probabilistic method*, volume 23 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2002.
- [33] Robin A. Moser. A constructive proof of the Lovász local lemma. In *STOC'09—Proceedings of the 2009 ACM International Symposium on Theory of Computing*, pages 343–350. ACM, New York, 2009.
- [34] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57(2):Art. 11, 15, 2010.
- [35] Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336. IEEE, 2008.
- [36] Christos H. Papadimitriou. On selecting a satisfying truth assignment. In *FOCS*, pages 163–169. IEEE Computer Society, 1991.
- [37] Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Sciences*, 82(7):1180–1200, 2016.
- [38] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238. Tsinghua University Press, 2011.
- [39] Michael Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing*, 39(7):2897–2926, 2010.
- [40] J.B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985.
- [41] Joel Spencer. Needles in exponential haystacks ii.
- [42] Aravind Srinivasan. Improved algorithmic versions of the Lovász local lemma. In Shang-Hua Teng, editor, *SODA*, pages 611–620. SIAM, 2008.
- [43] Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 45(2):24, 2013.

[44] Sergey Yekhanin et al. Locally decodable codes. *Foundations and Trends® in Theoretical Computer Science*, 6(3):139–255, 2012.

A Proofs for our Applications

In this section we prove Theorems 1.2, 1.4 and 1.5.

A.1 Proof of Theorem 1.2

We first briefly recall the application of the LLL in [17]. For each variable x_i , let d_i denote the number of clauses in which x_i occurs and assume that $\theta_i d_i$ of these occurrences are positive, for some $\theta_i \in [0, 1]$. Let $d = \max_{i \in [n]} d_i$ and let μ be the product measure over the variables of ϕ that sets each variable x_i to *true* with probability $\frac{1}{2} + \frac{2(1-\theta_i)d_i-d}{2dk}$. In [17] it is shown that if $d(k+1) \leq 2^{k+1}/e$ and we set $\psi_j = \frac{e}{2^k - e} = O(1)$ for each $j \in [m]$, then the LLL conditions are satisfied.

To establish part (a) of Theorem 1.2, recall the definition of ζ and ϵ in Theorem 3.1 and notice that it implies that

$$1 - \epsilon := (kd)^{-\zeta} \geq ((k+1)d)^{-\zeta} . \quad (11)$$

Thus, in order to meet the requirement of Theorem 3.1 that the LLL conditions hold with an ϵ -slack, i.e., that $d(k+1)e \leq (1 - \epsilon)2^{k+1}$, it is enough that

$$d(k+1)e \leq ((k+1)d)^{-\zeta} 2^{k+1} . \quad (12)$$

Setting $\eta = \zeta$ in (12), we get the condition of part (a) of Theorem 1.2, concluding the proof. Part (b) of Theorem 1.2 is a straightforward application of Theorem 3.1 and Remark 3.1.

A.2 Proof of Theorem 1.4

We'll need to briefly recall the key ideas in the analysis of the algorithm of [30].

In the first phase, the algorithm operates on the set Ω of complete but not necessarily proper colorings of G with at most $\Delta + 1 - Z$ colors, where $Z = B/(e^6 \Delta)$. For a vertex v and a state $\sigma \in \Omega$, say that a color c is *stable* if it is assigned to at least two non-adjacent neighbors of v and, moreover, all neighbors of v with color c do not belong in a monochromatic edge in σ . Let $X_v(\sigma)$ be the number of stable colors for v at σ . For each vertex v , define the bad event $A_v = \{\sigma \in \Omega : X_v(\sigma) \leq Z\}$ with respect to the probability space (μ, Ω) , where μ is the uniform measure over Ω . A coloring $\sigma^* \in \Omega$ that avoids all bad events, can be efficiently transformed to a proper coloring of G . To see this, consider the partial *proper* coloring σ' that results by uncoloring every vertex in σ^* that belongs in a monochromatic edge. Since σ^* avoided all bad events, this means that in the neighborhood of every [uncolored] vertex in σ' , at least Z colors appear at least twice. Therefore, in σ' , for every vertex v both of the following hold: (i) v has at most $\Delta - (2Z + s(v))$ uncolored neighbors, where $s(v)$ is the number of colors appearing exactly once in the neighborhood of v , and (ii) at least $\Delta + 1 - Z - (Z + s(v))$ colors are available, i.e., do not appear in v 's neighborhood. Thus, the graph induced by the uncolored vertices can be colored with available colors using the greedy heuristic.

To prove that we can find efficiently a coloring $\sigma^* \in \Omega$ that avoids all bad events we use the following two lemmas from the analysis of [30] (slightly modified to fit our needs). Below, both the expectation and the probability are with respect to μ .

Lemma A.1 ([30]). $\mathbb{E}[X_v] \geq 2Z$.

Lemma A.2 ([30]).

$$\Pr \left[|X_v - \mathbb{E}[X_v]| > (\log \Delta)^2 \sqrt{\mathbb{E}[X_v]} \right] \leq \Delta^{-\frac{\log \Delta}{1000}}.$$

Lemmata A.1 and A.2 imply that if $B \geq \Delta \log^4 \Delta$ and Δ is large enough, then $\Pr[A_v] \leq \Delta^{-\frac{\log \Delta}{1000}}$. On the other hand, each bad event A_v is mutually independent from all but at most Δ^4 other bad events, since it only depends on the color of vertices which are joined to v by a path of length at most 2. Thus, for large enough Δ , if $\psi_i = \frac{1}{\Delta^4 - 1}$ for every $i \in [m]$, then the LLL condition is satisfied, implying that the Moser-Tardos algorithm finds a coloring that avoids all bad events quickly.

Proof of Theorem 1.4. We consider the constraint satisfaction problem with one variable and one constraint per vertex v of the graph, the variable expressing the color of v , and the constraint including all vertices (variables) within distance 2 from v and forbidding all joint value assignments for which A_v occurs. Observe that knowing the color of the vertices included by the constraint of v is (more than) enough information to determine if v belongs in any monochromatic edge and, thus, whether it retains its color when we uncolor all vertices belonging in monochromatic edges. With this in mind, our local algorithm is the following.

Let q be the number of queries. For each $i \in [q]$, to answer the i -th query we use the procedure described in the proof of Theorem 3.1 to satisfy all constraints within a ball of some radius of the queried vertex v_i . Naturally, this colors all vertices in the neighborhood of v_i (and probably many others). If, in the resulting coloring, we find that v_i participates in a monochromatic edge we “guess” that v_i will be uncolored at the end of the first phase, otherwise we “guess” that it will have the colored assigned by our procedure. In the latter case, we return this color as our answer. To answer queries for vertices that we guess will be uncolored at the end of the first phase, we simulate the greedy coloring of the second phase, using the ordering of the vertices by the queries. That is, whenever we guess that v is uncolored, we chose one of its available colors, c , return it as our answer to the query, and record c as the color of v . If we later need to answer a query for a neighbor v' of v that we also guess to be uncolored, we do not consider c an available color for v' . Thus, if our algorithm does not make any wrong guesses, it doesn't make any error at all.

To bound the probability of error we use Theorem 3.1. Since the constraints correspond to the events A_v , we see that $k = \Delta$ and $d = \Delta^4$. We are interested in responding to at most n queries, i.e., $\alpha = 1$. Letting $p = \max_{v \in V} \Pr[A_v]$ and setting $\psi_i = \frac{1}{\Delta^4 - 1}$ we see that the LLL condition is satisfied if $p\Delta^4 e \leq 1$. Since β, γ are constants and $p \leq \Delta^{-\frac{\log \Delta}{1000}}$, we see that $p\Delta^4 e \leq \Delta^{-(\frac{5(1+\gamma)}{\beta} + 1)}$ for large enough Δ . Letting $\Delta^{-(\frac{5(1+\gamma)}{\beta} + 1)} =: 1 - \epsilon$ and noting that $\max_{i \in [m]} \psi_i = O(1)$, i.e., $\lambda = 0$, we obtain

$$\zeta = \frac{\log(\frac{1}{1-\epsilon})}{\log(kd)} = \frac{\left(\frac{5(1+\gamma)}{\beta} + 1\right) \log \Delta}{5 \log \Delta} > \frac{1 + \gamma}{\beta},$$

and in turn that $\beta\zeta > 1 + \gamma + 0$. Thus, Theorem 3.1 applies, concluding the proof. \square

A.3 Proof of Theorem 1.5

We consider the uniform measure over all possible 2-colorings of \mathcal{H} and define one bad event, A_e , for each edge e , corresponding to e being monochromatic. Clearly, $\Pr[A_e] = \frac{1}{2^{|e|-1}}$. If we set $\psi_e = \frac{2x_e}{1-x_e}$, where $x_e = \left(\frac{1}{2}\right)^{\frac{1}{2}(|e|-1)}$, the LLL conditions are satisfied assuming that

$$\sum_{i \geq 3} \Delta_i 2^{-i/2} \leq \frac{1}{6\sqrt{2}}. \quad (13)$$

(For more details, see the proof of Theorem 19.2 in [32]). Now, since $\max_e |e|$ and $\max_i \Delta_i$ are constants we see that the condition of Theorem 1.5 implies that the LLL condition holds with ϵ slack and, thus, the proof follows directly from Theorem 3.1 and Remark 3.1.

B Improved LLL Criteria and Commutative Algorithms

Our techniques can be generalized in two distinct directions. First, so that they apply under more permissive LLL conditions such as the *cluster expansion condition* [9] and *Shearer's condition* [40]. Second, they can be used to design local computation algorithms that simulate algorithms in the abstract settings of the algorithmic Lovász Local Lemma [1, 21, 2], where the probability space does not necessarily correspond to a product measure, and which capture the *lopsided* version of the LLL [15]. We briefly discuss these extensions below.

Given a dependency graph G over $[m]$ and a set $S \subseteq [m]$ we denote by $\text{Ind}([m]) = \text{Ind}_G([m])$ the family of subsets of S that correspond to independent sets in G .

Cluster Expansion condition. The cluster expansion criterion strictly improves upon the General LLL criterion (2) by taking advantage of the local density of the dependency graph.

Definition B.1. Given a sequence of positive real numbers $\{\psi_i\}_{i=1}^m$, we say that the cluster expansion condition is satisfied if for each $i \in [m]$

$$\frac{\mu(A_i)}{\psi_i} \sum_{S \in \text{Ind}(D(i) \cup \{i\})} \prod_{j \in S} \psi_j \leq 1 .$$

Shearer's condition. Shearer's condition improves upon the general and cluster expansion LLL conditions by exploiting the global structure of the dependency graph. It is best possible in the sense that if it is not satisfied, then one can always construct a probability space and bad events that are compatible with the given dependency graph, for which the probability of avoiding all bad events is zero.

Definition B.2. Let $\mu = (\mu_1, \mu_2, \dots, \mu_m) \in \mathbb{R}^m$ be the real vector such that $\mu_i = \mu(A_i)$. For $S \subseteq [m]$ define $\mu_S = \prod_{i \in S} \mu_i$ and the polynomial q_S

$$q_S = q_S(\mu) = \sum_{\substack{I \in \text{Ind}([m]) \\ S \subseteq I}} (-1)^{|I|-|S|} \mu_I .$$

We say that the Shearer's condition is satisfied if $q_S(\mu) \geq 0$ for all $S \subseteq [m]$, and $q_\emptyset(\mu) > 0$.

For the variable setting, the statement of our results remain identical under the cluster expansion and, essentially identical, under Shearer's conditions (ψ_i is replaced by $q_{\{i\}}(\mu)/q_\emptyset(\mu)$ and we say that the condition holds with ϵ -slack for a given vector μ , if it simply holds for vector $(1 + \epsilon)\mu$.) The only thing that changes in the analysis is the bound for the sum of probabilities of witness trees of large size in Lemma 5.6. (We refer the reader to Section 4 in [23] for further details.)

The first result that made the LLL constructive in a non-product probability space was due to Harris and Srinivasan in [20], who considered the space of permutations endowed with the uniform measure. Subsequent works by Achlioptas and Iliopoulos [1, 2, 3] introducing the flaws/actions framework, and of Harvey and Vondrák [21] introducing the resampling oracles framework, made the LLL constructive in more general settings. These frameworks [1, 2, 21, 3] provide tools for analyzing *focused* stochastic search algorithms [36], i.e., algorithms which, like the Moser-Tardos algorithm, search by repeatedly selecting a flaw of the current state and moving to a random nearby state that avoids it, in the hope that, more often than not, more flaws are removed than introduced, so that a flawless object is eventually reached.

Our techniques can be extended to these more general settings assuming they are *commutative*, a notion introduced by Kolmogorov [25, 3]. While we will not define the class of commutative algorithms here for

the sake of brevity, we note that it contains the vast majority of LLL algorithms, including the Moser-Tardos algorithm. The reason why our results apply in this case is because the *witness tree lemma*, i.e., Lemma 5.3 for the case of the Moser-Tardos algorithm (which was key to our analysis) holds for commutative algorithms [23, 3].