Parallel algorithms and concentration bounds for the Lovász Local Lemma via witness-DAGs

Bernhard Haeupler* David G. Harris†

Abstract

The Lovász Local Lemma (LLL) is a cornerstone principle in the probabilistic method of combinatorics, and a seminal algorithm of Moser & Tardos (2010) provides an efficient randomized algorithm to implement it. This algorithm can be parallelized to give an algorithm that uses polynomially many processors and runs in $O(\log^3 n)$ time, stemming from $O(\log n)$ adaptive computations of a maximal independent set (MIS). Chung et al. (2014) developed faster local and parallel algorithms, potentially running in time $O(\log^2 n)$, but these algorithms work under significantly more stringent conditions than the LLL.

We give a new parallel algorithm that works under essentially the same conditions as the original algorithm of Moser & Tardos but uses only a single MIS computation, thus running in $O(\log^2 n)$ time. This conceptually new algorithm also gives a clean combinatorial description of a satisfying assignment which might be of independent interest. Our techniques extend to the deterministic LLL algorithm given by Chandrasekaran et al. (2013) leading to an NC-algorithm running in time $O(\log^2 n)$ as well.

We also provide improved bounds on the runtimes of the sequential and parallel resampling-based algorithms originally developed by Moser & Tardos. Our bounds extend to any problem instance in which the tighter Shearer LLL criterion is satisfied. We also improve on the analysis of Kolipaka & Szegedy (2011) to give tighter concentration results.

1 Introduction

The Lovász Local Lemma (LLL), first introduced in [5], is a cornerstone principle in probability theory. In its simplest symmetric form, it states that if one has a probability space Ω and a set of m "bad" events $\mathcal{B}$ in that space, and each such event has probability $P_\Omega(B) \leq p$; and each event depends on at most d other events, then under the criterion

$$ep(d+1) \leq 1 \tag{1.1}$$

there is a positive probability that no bad events occurs. If equation (1.1) holds, we say *the symmetric LLL criterion is satisfied*.

Although the LLL applies to general probability spaces, and the notion of dependency for a general space can be complicated, in most applications in combinatorics a simpler setting is used, in which the probability space Ω is determined by a series of discrete variables $X_1, \dots, X_n$, each of which is drawn independently with $P_\Omega(X_i = j) = p_{ij}$. Each bad event $B \in \mathcal{B}$ is a Boolean function (possibly a complex one) determined by variables $S_B \subseteq [n]$. Then events B, B' are dependent if they share a common variable, i.e., $S_B \cap S_{B'} \neq \emptyset$.

We use the notation $B \sim B'$ to mean that B, B' are dependent. We also use the notation $N(B)$ to mean the *inclusive* neighborhood of B, that is, the set $N(B) = \{B' \in \mathcal{B} | B' \sim B\}$. Note that $B \in N(B)$. Finally, we use the notation $i \sim B$ to mean that $i \in S_B$.

There is a more general form of the LLL, known as the *asymmetric LLL*, which can be stated as follows. Suppose that one has a weighting function $x : \mathcal{B} \to (0, 1)$, with the following property:

$$\forall B \in \mathcal{B} \qquad P_\Omega(B) \leq x(B) \prod_{B' \sim B, B' \neq B} (1 - x(B')) \tag{1.2}$$

In this case, too, there is a positive probability of avoiding all bad events. The symmetric LLL is a special case of this, derived by setting $x(B) = ep$.

Both of these criteria are special cases of a yet more powerful criterion, known as the *Shearer criterion*. This criterion requires a number of definitions to state; we discuss this further in Section 1.3.

The probability of avoiding all bad events, while non-zero, is usually exponentially small; so the LLL does not lead to efficient algorithms directly. Moser & Tardos [14] introduced a remarkable randomized procedure, which we refer to as the *Resampling Algorithm*, which gives polynomial-time algorithms for nearly all LLL applications:

*School of Computer Science, Carnegie Mellon University. Research supported in part by NSF Awards CCF-1527110 and CCF-1618280. Email: haeupler@cs.cmu.edu

†Department of Computer Science, University of Maryland, College Park, MD 20742. Research supported in part by NSF Awards CNS-1010789 and CCF-1422569. Email: davidgharris29@gmail.com

1. Draw all variables $X \sim \Omega$.
2. While some bad events are true:
    3. Choose some true $B \in \mathcal{B}$ arbitrarily.
    4. Resample the variables in S_B, independently from the distribution Ω.

This resampling algorithm terminates with probability one under the same condition as the probabilistic LLL, viz. satisfying the Shearer criterion. The expected number of resamplings is typically polynomial in the input parameters.

We note that this procedure can be useful even when the total number of bad events is exponentially large. At any stage of this algorithm, the expected number of bad events which are currently true (and thus need to be processed), is still polynomial. If we have an oracle which lists the currently-true bad events in time $\text{poly}(n)$, then the overall run-time of this algorithm can still be polynomial in n. Such oracles are typically very problem-specific; see [8] for more details; for the remainder of this paper, we will simply assume that we have such an oracle.

1.1 Parallel algorithms for the LLL Moser & Tardos also gave a simple RNC algorithm for the LLL. Unlike their sequential algorithm, this requires a small slack in the LLL criterion. In the symmetric setting, this criterion is

$$ep(1+\epsilon)(d+1) \leq 1$$

and in the asymmetric setting, it is given by

$$\forall B \in \mathcal{B} \qquad (1+\epsilon)P_\Omega(B) \leq x(B) \prod_{B' \sim B, B' \neq B} (1 - x(B'))$$

for some parameter $\epsilon \in (0, 1/2)$. We refer to these stronger criteria as *ϵ-multiplicative slack.*

In this case, the parallel algorithm works as follows:

1. Draw all variables $X \sim \Omega$.
2. While some bad events are true:
    3. Choose a maximal independent set I of bad events which are currently true.
    4. Resample, in parallel, all the variables $\bigcup_{B \in I} S_B$ from the distribution Ω.

In order to implement this algorithm, we need a parallel subroutine which can find all the bad events that are true on a given configuration. We refer to this subroutine as a *Bad-Event Checker*; it is typically very problem-specific but routine to implement.

They showed that this algorithm terminates with high probability after $O\big(\epsilon^{-1} \log(n \sum_{B \in \mathcal{B}} \frac{x(B)}{1-x(B)})\big)$ rounds.[1] In each round, there are two main computational tasks: one must run the Bad-Event Checker and one must find a maximal independent set (MIS) among the bad events which are currently true. The former can typically be implemented in time $O(\log n)$. Using the algorithm of Luby [12], finding the MIS requires $O(\log^2(n \sum_{B \in \mathcal{B}} \frac{x(B)}{1-x(B)}))$ time. Thus the overall run-time is $O(\epsilon^{-1} \log^3(n \sum_{B \in \mathcal{B}} \frac{x(B)}{1-x(B)}))$ and the overall processor complexity is $\text{poly}(n, \sum_{B \in \mathcal{B}} \frac{x(B)}{1-x(B)})$.[2]

The computation of an MIS is relatively costly. In [4], Chung et al. gave several alternative algorithms for the symmetric LLL which either avoid this step or reduce its cost. Although the main focus of [4] was obtaining distributed algorithms for the LLL, these algorithm have reduced time complexity as well.

They give one algorithm, based on bad events choosing random priorities and resampling a bad event if it has earlier priority than its neighbors, which runs in $O(\epsilon^{-1} \log n)$ distributed rounds and $O(\epsilon^{-1} \log^2 n)$ time. Unfortunately, this algorithm of [4] requires a stronger criterion than the LLL: namely, in the symmetric setting, it requires that $epd^2 \leq (1-\epsilon)$. In many applications of the LLL, particularly those based on Chernoff bounds for the sum of independent random variables, satisfying the stricter criterion $epd^2 \leq (1-\epsilon)$ leads to qualitatively similar results as the symmetric LLL. In other cases, the criterion of [4] loses much critical precision leading to weaker results. In particular, their bound essentially corresponds to the state of the art [13] before the break-through result of Moser and Moser-Tardos [14].

Another algorithm given by Chung et al. requires only the standard symmetric LLL criterion and runs in $O(\epsilon^{-1} \log^2 d \log n)$ rounds. Recently, a key subroutine used by this algorithm was improved by [7], leading to a reduction to $O(\epsilon^{-1} \log d \log n)$ rounds. When d is polynomial in n, however, this does not improve on the Moser-Tardos algorithm.

In [3], a deterministic parallel (NC) algorithm was given for the LLL. In the symmetric LLL setting, this requires satisfying a criterion $epd^{1+\epsilon} \leq 1$, and the

[1] We say that an event occurs *with high probability* (abbreviated whp), if it occurs with probability $\geq 1 - n^{-\Omega(1)}$.

[2] We note that the weighting function $x(B)$ plays a somewhat mysterious role in the LLL, and it can be confusing to have it appear in the complexity bounds for Resampling Algorithm. There are even pathological cases in which the LLL criterion are satisfied, and the Parallel Resampling Algorithm terminates quickly, but $\sum_B \frac{x(B)}{1-x(B)} \to \infty$. But usually we have $\sum_B \frac{x(B)}{1-x(B)} = \text{poly}(n)$.

overall runtime is $O(\epsilon^{-1}\log^3(mn))$. This too can be extended to an asymmetric setting, but there are many more technical conditions on the precise form of $\mathcal{B}$.

1.2 Overview of our results In Section 2, we introduce a new theoretical structure to analyze the behavior of the Resampling Algorithm, which we refer to as the *witness DAG*. This provides an explanation or history for some or all of the resamplings that occur. This generalizes the notion of a witness tree, introduced by Moser & Tardos in [14], which only provides the history of a single resampling. We use this tool to show stronger bounds on the Parallel Resampling Algorithm given by Moser & Tardos:

THEOREM 1.1. *Suppose that the Shearer criterion is satisfied with ϵ-multiplicative slack. Then whp the Parallel Resampling Algorithm terminates after $O(\epsilon^{-1}\log n)$ rounds*

Suppose furthermore we have a Bad-Event Checker which uses polynomial processors and T time. Then total complexity of the Parallel Resampling Algorithm is $O(\frac{(\log n)(T+\log^2 n)}{\epsilon})$ expected time and $\epsilon^{-1}n^{O(1)}$ processors.

These bounds are independent of the LLL weighting function $x(B)$ and the number of bad events m. These significantly improve on qualitatively similar bounds shown in Kolipaka & Szegedy [11], which show that Parallel Resampling Algorithm terminates, with constant probability, after $\epsilon^{-1}\log(n/\epsilon)$ rounds.[3]

In Section 3, we show a new and stronger concentration result for the runtime of the Sequential Resampling Algorithm. Our bound improves on similar concentration bounds shown in [11] and [1]. We state this result here in terms of the asymmetric LLL criterion; we later also give a more general bound in terms of the Shearer criterion.

THEOREM 1.2. *Suppose that the asymmetric LLL criterion is satisfied with ϵ-multiplicative slack. Then, with high probability, the total number of resamplings made by the Resampling Algorithm is at most $O(\sum_B \frac{x(B)}{1-x(B)} + \epsilon^{-1}\log^2 n)$. Alternatively, suppose that the symmetric LLL criterion $ep(d+1)\leq 1$ is satisfied. Then, whp, the number of resamplings is at most $O(n + d\log^2 n)$.*

In Section 4 and 5, we develop a new parallel algorithm for the LLL. The basic idea of this algorithm is to select a random resampling table and then precompute all possible resampling-paths compatible with it. Surprisingly, this larger collection, which in a sense represents all possible choices for the trajectory of the Resampling Algorithm, can still be computed relatively quickly (in approximately $O(\epsilon^{-1}\log^2 n)$ time). Next, we find a *single* MIS of this larger collection, which will allow us to determine the complete set of resamplings necessary. It is this reduction from $\epsilon^{-1}\log n$ separate MIS algorithms to just one that is the key to our improved runtime.

We will later analyze the run-time of this parallel algorithm for the Shearer criterion, but this requires many preliminary definitions. We give a simpler statement of our new algorithm for the symmetric LLL criterion:

THEOREM 1.3. *Suppose that we have a Bad-Event Checker running in time $O(\log n)$ using polynomially many processors. Suppose that each bad event B has $P_\Omega(B)\leq p$ and is dependent with at most d other bad events and that $ep(1+\epsilon)(d+1)\leq 1$ for some $\epsilon>0$. Then, there is an algorithm finding a configuration avoiding $\mathcal{B}$ using $\tilde{O}(\epsilon^{-1}\log(mn)\log n)$ time and $(mn)^{O(1)}$ processors. (Here, the $\tilde{O}$ hides factors of the form poly$(\log\log(mn))$.)*

In Section 6, we show how this algorithm can be derandomized, to give an NC algorithm which also requires $O(\epsilon^{-1}\log^2(mn))$ time.

1.3 Stronger LLL criteria The LLL criterion, in either its symmetric or asymmetric form, depends on only two parameters: the probabilities of the bad events, and their dependency structure. The symmetric LLL criterion $ep(d+1)\leq 1$ is a very simple criterion involving these parameters, but it is not the most powerful. In [15], Shearer gave the strongest possible criterion that can be stated in terms of these parameters alone. This criterion is somewhat cumbersome to state and difficult to work with technically, but it is useful theoretically because it subsumes many of the other simpler criteria.

We note that the "lopsided" form of the LLL can be applied to this setting, in which bad events are atomic configurations of the variables (as in a k-SAT instance), and this can be stronger than the ordinary LLL. Moser & Tardos showed that, to a limited extent this can be made algorithmic. As shown in [9], there are forms of lopsidependency in the Moser-Tardos setting which can even go beyond the Shearer criterion itself. However, giving parallel algorithms which can take advantage of this lopsidependency phenomenon is very challenging. In [9], a complicated and much slower parallel algorithm was given for this setting. In this paper we are only concerned with the standard (not lopsided) LLL, and we

[3] Note that Kolipaka & Szegedy use m for the number of variables and n for the number of bad events, while we do the opposite. In this paper, we have translated all of their results into our notation. The reader should be careful to keep this in mind when reading their original paper.

do not attempt to match the algorithmic improvements of [9].

To state the Shearer criterion, it will be useful to suppose that the dependency structure of our bad events $\mathcal{B}$ is fixed, but the probabilities for the bad events have not been specified. We define the *independent-set polynomial* $Q(I,p)$ as

$$Q(I,p) = \sum_{\substack{I \subseteq J \subseteq \mathcal{B} \\ J \text{ independent}}} (-1)^{|J|-|I|} \prod_{B \in J} p(B)$$

for any $I \subseteq \mathcal{B}$. Note that $Q(I,p) = 0$ if I is not an independent set. This quantity plays a key role in Shearer's criterion for the LLL [15], and in turn plays a key role in the behavior of the Resampling Algorithm. We say that the probabilities p satisfy the Shearer criterion iff $Q(\emptyset,p) > 0$ and $Q(I,p) \geq 0$ for all independent sets $I \subseteq \mathcal{B}$.

PROPOSITION 1.1. ([15]) *Suppose that p satisfies the Shearer criterion. Then any probability space with the given dependency structure and probabilities $P_\Omega = p$ has a positive probability that none of the bad events B are true.*

Suppose that p do not satisfy the Shearer criterion. Then there is a probability space Ω with the given dependency structure and probabilities $P_\Omega = p$ for which, with probability one, at least one $B \in \mathcal{B}$ is true.

PROPOSITION 1.2. ([15]) *Suppose that $p(B) \leq p'(B)$ for all $B \in \mathcal{B}$. Then, if p' satisfies the Shearer criterion, so does p.*

One useful parameter for us will be the following:

DEFINITION 1.1. *For any bad event B, define the* measure *of B to be $\mu(B) = \frac{Q(\{B\},P_\Omega)}{Q(\emptyset,P_\Omega)}$.*

In [11], Kolipaka & Szegedy showed that if the Shearer criterion is satisfied, then the Resampling Algorithm terminates with probability one. Furthermore, the run-time of the Parallel and Sequential Resampling Algorithms can be bounded in terms of the measures μ.

PROPOSITION 1.3. ([11]) *The expected number of resamplings of any $B \in \mathcal{B}$ is at most $\mu(B)$.*

This leads us to define the *work parameter* for the LLL by $W = \sum_{B \in \mathcal{B}} \mu(B)$.

Although the sequential Resampling Algorithm can often work well when the Shearer criterion is satisfied (almost) exactly, for the parallel Resampling Algorithm one must often satisfy it with a small slack.

DEFINITION 1.2. *We say that the Shearer criterion is satisfied with ϵ-multiplicative slack, if the vector of probabilities $(1+\epsilon)P_\Omega$ satisfies the Shearer criterion.*

It it extremely difficult to directly show that the Shearer criterion is satisfied in a particular instance. There are alternative criteria, which are weaker than the full Shearer criterion but much easier to work with computationally. Perhaps the simplest is the asymmetric LLL criterion. The connection between the Shearer criterion and the asymmetric LLL criterion was shown by Kolipaka & Szegedy in [11].

THEOREM 1.4. *Suppose that there is a weighting function $x : \mathcal{B} \to (0,1)$ satisfying*

$$\forall B \in \mathcal{B} \qquad P_\Omega(B)(1+\epsilon) \leq x(B) \prod_{B' \sim B} (1 - x(B'))$$

Then the Shearer criterion is satisfied with multiplicative ϵ-slack, and $\mu(B) \leq x(B)/(1-x(B))$ for all $B \in \mathcal{B}$.

This was extended to the cluster-expansion LLL criterion of [2] by Harvey & Vondrak in [10]:

THEOREM 1.5. ([10]) *Suppose that there is a weighting function $\tilde{\mu} : \mathcal{B} \to [0,\infty)$ satisfying*

$$\forall B \in \mathcal{B} \qquad \tilde{\mu}(B) \geq P_\Omega(B)(1+\epsilon) \sum_{\substack{I \subseteq N(B) \\ I \text{ independent}}} \prod_{B' \in I} \tilde{\mu}(B')$$

Then the Shearer criterion is satisfied with multiplicative ϵ-slack, and $\mu(B) \leq \tilde{\mu}(B)$.

For the remainder of this paper, we will always assume, unless stated otherwise, that our probability space satisfies the Shearer criterion with ϵ-multiplicative slack. We will occasionally derive certain results for the symmetric LLL criterion as a corollary of results on the full Shearer criterion.

2 The witness DAG and related structures

There are two key analytical tools introduced by Moser & Tardos to analyze their algorithm: the resampling table and witness trees.

The *resampling table* R is a table of values $R(i,t)$, where i ranges over the variables $1,\ldots,n$ and t ranges over the natural numbers $1,2,\ldots$. Each cell $R(i,t)$ is drawn independently from the distribution on the variable i, that $R(i,t) = j$ with probability p_{ij}, independently of all other cells. The intent of this table is that, instead of choosing new values for the variables in "online" fashion, we precompute the future values of all the variables. The first entry in the table $R(i,1)$, is the initial value for the variable X_i; on the t^{th} resampling, we set $X_i = R(i,t+1)$.[4]

[4] Although nominally the resampling table provides a countably infinite stream of values for each variable, in practice we will only need to use approximately $\epsilon^{-1}\log n$ distinct values for each variable.

The *witness tree* is a structure which records the history of all variables involved in a given resampling. Moser & Tardos give a very clear and detailed description of the process for forming witness trees; we provide a simplified description here. Suppose that we run the resampling algorithm and resample bad events $B_1, \ldots, B_t$ (the algorithm has not necessarily terminated by this point). We build a witness-tree $\hat{\tau}_t$ for the t^{th} resampling, as follows. We place a node labeled by B_t at the root of the tree. We then go backwards in time for $j = t-1, \ldots, 1$. For each B_j, if there is a node v' in the tree labeled by $B' \sim B_j$, then we add a new node v labeled by B_j as a child of v'; if there are multiple choices of v', we always select the one of greatest depth (breaking ties arbitrarily.) If there is no such node v', then we do not add any nodes to the tree for that value of j.

2.1 The witness DAG. The witness tree $\hat{\tau}$ only provides an explanation for the single resampling at time t. It may discard some information about other resamplings that were not relevant to time t. We now consider a related object, the *witness DAG*, that can record information about multiple resamplings, or all of the resamplings.

A witness DAG is a directed acyclic graph, whose nodes are labeled by bad events. For nodes $v, v' \in G$, we write $v \prec v'$ if there is an edge from v to v'. We further impose two requirements. First, if nodes v, v' are labeled by B, B' and $B \sim B'$, then either $v \prec v'$ or $v' \prec v$; if $B \not\sim B'$ then there is no edge between v, v'. We refer to this as the *comparability condition*.

We let $|G|$ denote the number of vertices in a witness DAG G.

It is possible that a witness DAG can contain multiple nodes with the same label. However, because of the comparability condition, all such nodes are linearly ordered by $\prec$. Thus, for any witness DAG G and any $B \in \mathcal{B}$, one can unambiguously sort the nodes of G labeled by B. Thus, we use the notation (B, k) to mean that node v is the k^{th} node of G labeled by B. For any node v, we refer to this ordered pair (B, k) as the *extended label* of v. Every node in a witness DAG receives a distinct extended label. We emphasize that this is a notational convenience, as an extended label of a node can be recovered from the witness DAG G along with its un-extended labels.

Given a full execution of the Resampling Algorithm, one can form a a particularly important witness DAG $\hat{G}$ which we refer to as the *full witness DAG*, as follows. Suppose that we resample bad events $B_1, \ldots, B_t$. Then $\hat{G}$ has vertices $v_1, \ldots, v_t$ which are labeled $B_1, \ldots, B_t$. We place an edge from v_i to v_j iff $i < j$ and $B_i \sim B_j$. It is not hard to see that this graph is indeed a witness DAG as we have defined it. We emphasize that $\hat{G}$ is a random variable, and we distinguish between this notion and that of a witness DAG (which is a non-random variable). The full witness DAG (under different terminology) was analyzed by Kolipaka & Szegedy in [11], and we will use their results in numerous places. However, we will also consider partial witness DAGs, which record information about only a subset of the resamplings. As we will see, these partial witness DAGs can be useful even when we wish to analyze the full set of resamplings.

As witness trees and single-sink witness DAGs are closely related, we will often use the notation τ for a single-sink witness DAG.

2.2 Compatibility conditions for witness DAGs and resampling tables In the Moser-Tardos proof, a method was shown for converting an execution log into a witness tree, and necessary conditions were given for a witness tree being produced in this fashion in terms of its consistency with the resampling table. We will instead use these conditions as a *definition* of compatibility.

DEFINITION 2.1. (PATH OF A VARIABLE) *Let G be a witness DAG. For any $i \in [n]$, let $G[i]$ denote the subgraph of G induced on all vertices v labeled by B with $i \in S_B$. Because of the comparability condition, $G[i]$ is linearly ordered by $\prec$; thus we refer to $G[i]$ as the* path *of variable i.*

DEFINITION 2.2. (CONFIGURATION OF v) *Let G be a witness DAG and R a resampling table. Let $v \in G$ be labeled by B. For each $i \in S_B$, let $y_{v,i}$ denote the number of vertices $w \in G[i]$ such that $w \prec v$.*

We now define the configuration of v *by*

$$X_G^v(i) = R(i, 1 + y_{v,i})$$

DEFINITION 2.3. (COMPATIBILITY) *For a witness DAG G and a resampling table R, we say that G is* compatible *with R if, for all nodes $v \in G$ labeled by $B \in \mathcal{B}$, it is the case that B is true on the configuration X_G^v.*

Note that this is well-defined because because X_G^v assigns values to all the variables in S_B.

The following are key results used by Moser & Tardos to bound the running time of their resampling algorithm:

DEFINITION 2.4. (WEIGHT OF A DAG) *Let G be any witness DAG, whose nodes are labeled by bad events $B_1, \ldots, B_s$. We define the* weight *of G to be $w(G) = \prod_{k=1}^{s} P_\Omega(B_k)$.*

PROPOSITION 2.1. *Let G be any witness DAG. For a random resampling table R, G is compatible with R with probability $w(G)$.*

Proof. For any node $v \in G$, note that X_G^v follows the law of Ω, and so the probability that B is true of the configuration X_G^v is $P_\Omega(B)$. Next, note that each node $v \in G$ imposes conditions on disjoint sets of entries of R, and so these events are independent.

The following result shows how witness DAGs and resampling tables are related to the Resampling Algorithm:

PROPOSITION 2.2. *Suppose we run the Resampling Algorithm, taking values for the variables from the resampling table R. Then $\hat{G}$ is compatible with R.*

Proof. Suppose there is a node $v \in \hat{G}$ with an extended label (B, k). Thus, B must be resampled at least k times. Suppose that the k^{th} resampling occurs at time t. Let Y be the configuration at time t, just before this resampling. We claim that, for all $i \in S_B$, we have $Y(i) = X_{\hat{G}}^v(i)$. For, the graph $\hat{G}$ must contain all the resamplings involving variable i. All such nodes would be connected to vertex v (as they overlap in variable i), and those that occur before time t are precisely those that have an edge to v. So $y_{v,i}$ is exactly the number of bad events up to time t that involve variable i. Thus, just before the resampling at time t, variable i was on its $1+y_{v,i}$ resampling. So $Y(i) = R(i, 1+y_{v,i}) = X_{\hat{G}}^v(i)$, as claimed.

Now, in order for B to be resampled at time t, it must have been the case that B was true, i.e., that B held on configuration Y. However, since Y agrees with $X_{\hat{G}}^v$ on S_B, it must be also be the case that B holds on configuration $X_{\hat{G}}^v$. Since this is true for all v, it follows that G is compatible with R.

2.3 Prefixes of a witness DAG A witness DAG G records information about many resamplings. If we are only interested in the history of a subset of its nodes, then we can form a *prefix subgraph* which discards irrelevant information.

DEFINITION 2.5. (PREFIX GRAPH) *For any vertices $v_1, \ldots, v_l \in V$, let $G(v_1, \ldots, v_l)$ denote the subgraph of G induced on all vertices which have a path to at least one of $v_1, \ldots, v_l$.*

If H is a subgraph of G with $H = G(v_1, \ldots, v_l)$ for some $v_1, \ldots, v_l \in G$, then we say that H is a prefix *of G.*

Using Definition 2.5, we can give a more compact definition of the configuration of a node:

PROPOSITION 2.3. *For any witness DAG G and $v \in G$, we have $X_G^v(i) = R(i, |G(v)[i]|)$.*

Proof. Suppose that v is labeled by B. The graph $G(v)[i]$ contains precisely v itself and the other nodes $w \in G[i]$ with $w \prec v$. So $|G(v)[i]| = y_{v,i} + 1$.

PROPOSITION 2.4. *Suppose G is compatible with R and H is a prefix of G. Then H is compatible with R.*

Proof. Suppose $H = G(v_1, \ldots, v_l)$.

Consider $w \in H$ labeled by B. We claim that $H(w) = G(w)$. For, consider any $u \in H(w)$. So u has a path to w in H; it also must have a path to w in G. On the other hand, suppose $u \in G(w)$, so u has a path p to w in G. As w has a path to one of $v_1, \ldots, v_l$, this implies that every vertex in the path p also has such a path. Thus, the path p is in H, and hence u has a path in H to w, so $u \in H(w)$.

Next, observe that for any $i \in S_B$ we have

$$X_G^w(i) = R(i, |G(w)[i]|) = R(i, |H(w)[i]|) = X_H^w(i)$$

and by hypothesis, B is true on X_G^w.

2.4 Counting witness trees and witness DAGs If we are given a collection of witness DAGs $\mathcal{G}$, we define the *total weight* of $\mathcal{G}$ as $\sum_{G \in \mathcal{G}} w(G)$. In this section, we count the total weight of certain classes of witness DAGs. In light of Proposition 2.1, these will upper-bound the expected number of resamplings.

PROPOSITION 2.5. ([11]) *Let $B \in \mathcal{B}$. Then the total weight of all witness DAGs with a single sink node labeled B, satisfies*

$$\sum_{\tau \text{ has single sink node } B} w(\tau) \leq \mu(B).$$

Proof. For any witness DAG G with a single sink node v labeled B, define I'_j for $j = 0, \ldots, \infty$ inductively as follows. $I'_0 = \{v\}$, and I'_{j+1} is the set of vertices in G whose out-neighbors all lie in $I'_0 \cup \cdots \cup I'_j$. Let I_j denote the labels of the vertices in I'_j; so $I_0 = \{B\}$.

Now observe that by the comparability condition each set I_j is an independent set, and for each $B' \in I_{j+1}$ there is some $B'' \sim B', B'' \in I_j$. Also, the mapping from G to $I_0, \ldots, I_j$ is injective. We thus may sum over all such $I_1, \ldots, I_\infty$ to obtain an upper bound on the weight of such witness DAGs. In [11] Theorem 14, this sum is shown to be $Q(\{B\}, P_\Omega)/Q(\emptyset, P_\Omega)$ (although the notation they use is slightly different.)

We will now take advantage of the ϵ-multiplicative slack in our probabilities.

DEFINITION 2.6. (ADJUSTED WEIGHT) *For any witness DAG G, we define the* adjusted weight *with respect to rate factor ρ by*

$$a_\rho(G) = w(G)(1+\rho)^{|G|}.$$

Observe that $w(G) = a_0(G)$.

PROPOSITION 2.6. *For any variable $i \in [n]$ and any $\rho \in [0, \epsilon)$, we have*

$$\sum_{B \sim i} \sum_{\substack{\tau \text{ has single sink node} \\ \text{labelled } B}} a_\rho(\tau) \leq \frac{1+\rho}{\epsilon - \rho}.$$

Proof. Let $V = \{B \mid i \in S_B\}$.

The probabilities $(1+\epsilon)P_\Omega$ satisfy the LLL criterion, and by Proposition 1.2 so must $(1+\rho)P_\Omega$. Now, applying Proposition 2.5 to the probabilities $(1+\rho)P_\Omega$, we have that

(2.3)
$$\sum_{B \in V} \sum_{\substack{\tau \text{ has} \\ \text{single sink node } B}} a_\rho(\tau) \leq \sum_{B \in V} \frac{Q(\{B\}, (1+\rho)P_\Omega)}{Q(\emptyset, (1+\rho)P_\Omega)}$$

Our next task is to bound the RHS of (2.3). Consider the probability vector p defined by

$$p(B) = \begin{cases} (1+\epsilon)P_\Omega(B) & \text{if } B \in V \\ (1+\rho)P_\Omega(B') & \text{if } B \notin V \end{cases}$$

Note that $p \leq (1+\epsilon)P_\Omega$ and so by Propositions 1.1, 1.2 we have $Q(\emptyset, p) > 0$. But now consider that we have

$$\begin{aligned}
Q(\emptyset, p) &= \sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} (-1)^{|I|} \prod_{B' \in I} p(B') \\
&= \sum_{\substack{V \subseteq I \subseteq \mathcal{B} \\ I \text{ independent}}} (-1)^{|I|} \prod_{B \in I} p(B) \\
&\quad + \sum_{\substack{I \subseteq \mathcal{B} - V \\ I \text{ independent}}} (-1)^{|I|} \prod_{B \in I} p(B) \\
&= \sum_{B \in V} (1+\epsilon)P_\Omega(B) \sum_{\substack{B \in I \subseteq \mathcal{B} \\ I \text{ independent}}} (-1)^{|I|} \prod_{\substack{B' \in I \\ B' \neq B}} (1+\rho)P_\Omega(B') \\
&\quad + \sum_{\substack{I \subseteq \mathcal{B} - V \\ I \text{ independent}}} (-1)^{|I|} \prod_{B \in I} (1+\rho)P_\Omega(B) \\
&= \sum_{B \in V} (\epsilon - \rho)P_\Omega(B) \sum_{\substack{B \in I \subseteq \mathcal{B} \\ I \text{ independent}}} (-1)^{|I|} \prod_{\substack{B' \in I \\ B' \neq B}} (1+\rho)P_\Omega(B') \\
&\quad + \sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} (-1)^{|I|} \prod_{B \in I} (1+\rho)P_\Omega(B) \\
&= \frac{-(\epsilon - \rho)}{1+\rho} \sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega) + Q(\emptyset, (1+\rho)P_\Omega)
\end{aligned}$$

We may now compute the sum over $B \in V$ as:

$$\begin{aligned}
&\sum_{B \in V} \frac{Q(\{B\}, (1+\rho)P_\Omega)}{Q(\emptyset, (1+\rho)P_\Omega)} \\
&\quad = \frac{\sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega)}{Q(\emptyset, p) + \frac{(\epsilon-\rho)}{1+\rho} \sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega)} \\
&\quad \leq \frac{\sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega)}{\frac{(\epsilon-\rho)}{1+\rho} \sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega)} \\
&\qquad \text{as } Q(\emptyset, p) > 0 \\
&\quad = \frac{1+\rho}{\epsilon - \rho}
\end{aligned}$$

COROLLARY 2.1. ([11]) *The total weight of all single-sink witness DAGs satisfies*

$$\sum_{\text{single-sink witness DAGs } \tau} w(\tau) \leq n/\epsilon$$

Proof. We have

$$\begin{aligned}
&\sum_{\text{single sink witness DAGs } \tau} w(\tau) \\
&\quad \leq \sum_i \sum_{\substack{\tau \text{ has a single sink node} \\ \text{labeled by some } B \sim i}} w(\tau) \\
&\quad \leq \sum_i \epsilon^{-1} \qquad \text{by Proposition 2.6} \\
&\quad = n/\epsilon
\end{aligned}$$

PROPOSITION 2.7. *For $r \geq 1 + 1/\epsilon$, the expected number of single-sink witness DAG compatible with R containing more than r nodes is at most $enr(1+\epsilon)^{-r}$*

Proof. Summing over such DAGs:

$$\begin{aligned}
&\sum_{\substack{\tau \text{ has single sink node} \\ |\tau| \geq r}} P(\tau \text{ compatible with } R) \\
&\quad = \sum_{\substack{\tau \text{ has single sink node} \\ |\tau| \geq r}} w(\tau) \\
&\quad \leq (1+\rho)^{-r} \sum_{\substack{\tau \text{ has single sink node} \\ |\tau| \geq r}} w(\tau)(1+\rho)^{|\tau|} \\
&\qquad \text{for any } \rho \in [0, \epsilon] \\
&\quad \leq (1+\rho)^{-r} \sum_{\substack{\tau \text{ has single sink node} \\ |\tau| \geq r}} a_\rho(\tau)
\end{aligned}$$

$$\leq (1+\rho)^{-r} \sum_{i\in[n]} \sum_{B|i\in S_B} \sum_{\tau \text{ has single sink node } B \atop |\tau|\geq r} a_\rho(\tau)$$

$$\leq (1+\rho)^{-r} n \frac{1+\rho}{\epsilon-\rho} \qquad \text{by Proposition 2.6}$$

Now take $\rho = \epsilon - (1+\epsilon)/r$. By our condition $r \geq 1 + 1/\epsilon$ we have $\rho \in [0, \epsilon]$ and so Proposition 2.6 applies. Hence the expected number of such witness DAGs is thus at most $n\frac{r^r}{(r-1)^{r-1}(1+\epsilon)^r} \leq enr(1+\epsilon)^{-r}$.

COROLLARY 2.2. *Whp, all single-sink witness DAGs compatible with R contain $O(\frac{\log(n\epsilon^{-1})}{\epsilon})$ nodes. Whp all but $\frac{10\log n}{\epsilon}$ single-sink witness DAGs compatible with R contain at most $\frac{10\log n}{\epsilon}$ nodes.*

Proof. This follows immediately from Markov's inequality and Proposition 2.7.

COROLLARY 2.3. *Whp, all witness DAGs compatible with R have height $O(\frac{\log n}{\epsilon})$.*

Proof. Suppose that there is a witness DAG G of height T compatible with R. Then for $i = 1, \ldots, T$ there is a single-sink witness DAG of height i compatible with R, and all such DAGs are distinct. (Select a node v of height i, and set $G_i = G(v_i)$.) This implies that there $\Omega(T)$ single-sink witness DAGs of height $\Omega(T)$ compatible with R. By Proposition 2.2, this implies $T = O(\frac{\log n}{\epsilon})$.

With this Corollary 2.3, we are able to give a better bound on the complexity of the Parallel Resampling algorithm. The following Proposition 2.8 is remarkable in that the complexity is phrased solely in terms of the number of variables n and the slack ϵ, and is otherwise independent of $\mathcal{B}$.

PROPOSITION 2.8. *Suppose that the Shearer criterion is satisfied with ϵ-multiplicative slack.*

Whp, the Parallel Resampling Algorithm terminates after $O(\frac{\log n}{\epsilon})$ rounds.

Suppose we have a Bad-Event Checker in time T and polynomial processors. Then the total complexity of the Parallel Resampling Algorithm is $O(\frac{(\log n)(T+\log^2 n)}{\epsilon})$ time and $\epsilon^{-1}n^{O(1)}$ processors.

Proof. An induction on i shows that if the Parallel Resampling Algorithm runs for i steps, then $\hat{G}$ has depth i, and it is compatible with R. But Corollary 2.3 shows that, whp, this implies that $i = O(\frac{\log n}{\epsilon})$.

This implies that the total time needed to identify true bad events is $O(iT) \leq O(\frac{T\log n}{\epsilon})$.

Now, suppose that at stage i the number of bad events which are currently true is v_i. Then the total work spent computing the maximal independent sets, over the full algorithm, is $\sum_{i=1}^{t} O(\log^2 v_i) \leq O(t\log^2(\sum v_i/t))$. On the other hand, for each bad event which is at true at each stage, one can construct a corresponding witness tree, and all such trees are *unique*. Hence, $\mathbf{E}[\sum v_i] \leq W \leq n/\epsilon$. At $t \leq \frac{\log n}{\epsilon}$, we have $\mathbf{E}[t\log^2(\sum v_i/t)] \leq \epsilon^{-1}\log^3 n$. This shows the bound on the time complexity of the algorithm.

Now suppose we can enumerate all the currently true bad events. The expected number of bad events which are ever true is at most the weight of all single-sink witness DAGs, which is $W \leq n/\epsilon$. By Markov's inequality, whp the total number of bad events which are ever true is bounded by $\epsilon^{-1}n^{O(1)}$.

We contrast this with a qualitatively similar result in Kolipaka & Szegedy, which shows that Parallel Resampling Algorithm terminates, with constant probability, after $n/\epsilon\log(n/\epsilon)$ rounds.

3 Concentration for the number of resamplings

Although the main focus of this paper is to create a parallel algorithm for the LLL, using our results on witness DAGs we are able to show a powerful result for the runtime of the sequential Resampling Algorithm.

The expected number of resamplings for the Resampling Algorithm is at most W. Suppose we wish to ensure that the number of resamplings is bounded with high probability, not merely in expectation. One simple way to achieve this would be to run $\log n$ instances of the Resampling Algorithm in parallel; this is a generic amplification technique which ensures that whp the total number of resamplings performed will be $O(W\log n)$.

Can we avoid this extraneous factor of $\log n$? In this section, we answer this question in the affirmative by giving a concentration result for the number of resamplings. We show that whp the number of resamplings will not exceed $O(W)$ (assuming that W is sufficiently large).

We note that the straightforward approach here would be the following: the probability that there are T resamplings is at most the probability that there is a T-node witness DAG compatible with R; this can be upper-bounded by summing the weights of all such T-node witness DAGs. This straightforward approach can only the weaker result that number of resamplings is bounded by $O(W/\epsilon)$.

We contrast our result with Kolipaka & Szegedy [11], which shows that the Resampling Algorithm terminates after $O(n^2/\epsilon + n/\epsilon\log(1/\epsilon))$ resamplings with constant probability. In [1], a similar type of concen-

tration result is shown: they show that in the symmetric LLL setting, their algorithm (which is a variant/generalization of the Moser-Tardos algorithm) performs $O(n/\epsilon)$ resamplings whp.

PROPOSITION 3.1. *Given any distinct bad events $B_1, \dots, B_s$, the total weight of all witness DAGs with s sink nodes labeled $B_1, \dots, B_s$, is at most $\prod_{i=1}^{s} \mu(B_i)$.*

Proof. We define a function F which which maps s-tuples $(\tau_1, \dots, \tau_s)$ of single-sink witness DAGs with sink nodes labeled respectively $B_1, \dots, B_s$, to witness DAGs $G = F(\tau_1, \dots, \tau_s)$ whose sink nodes are labeled $B_1, \dots, B_s$. The function is defined by first forming the disjoint union of the graphs $\tau_1, \dots, \tau_s$. We then add an edge from a node $B \in \tau_i$ to $B' \in \tau_j$ iff $i < j$ and $B \sim B'$.

Now, consider any witness DAG G whose sink nodes $v_1, \dots, v_s$ are labeled $B_1, \dots, B_s$. For $i = 1, \dots, j$, define τ_i recursively by

$$\tau_i = G(v_i) - \tau_1 - \cdots - \tau_{i-1}$$

Note that each τ_i contains the sink node v_i, so it is nonempty. Also, all the nodes in τ_i are connected to v_i, so τ_i indeed has a single sink node. Finally, every node of G has a path to one of $v_1, \dots, v_j$, so it must in exactly one τ_i.

Thus, for each witness DAG G with sink nodes labeled $B_1, \dots, B_s$, there exist s separate single-sink witness DAGs $\tau_1, \dots, \tau_s$ such that $G = F(\tau_1, \dots, \tau_s)$, and furthermore such that the nodes of G are the union of the nodes of $\tau_1, \dots, \tau_s$. In particular, $w(G) = w(\tau_1) \cdots w(\tau_s)$. So we have:

$$\begin{aligned}
&\sum_{G \text{ has } s \text{ sink nodes } B_1, \dots, B_s} w(G) \\
&\leq \sum_{\tau_1, \dots, \tau_s} w(\tau_1) \dots w(\tau_s) \\
&= \prod_{i=1}^{s} \sum_{\tau \text{ has single sink node } B_i} w(\tau) \\
&\leq \prod_{i=1}^{s} \mu(B_i) \qquad \text{by Proposition 2.5}
\end{aligned}$$

THEOREM 3.1. *Whp, the total number of resamplings made the Resampling Algorithm is at most $O(W + \frac{\log^2 n}{\epsilon})$.*

Proof. First, consider the expected number of witness DAGs which are compatible with R and which contain exactly s sink nodes; here s is a parameter to be specified later. Each of these s sink nodes must receive distinct labels. We can estimate this quantity as

$$\begin{aligned}
&\sum_{G \text{ has } s \text{ sink nodes}} P(G \text{ compatible with } R) \\
&\leq \sum_{G \text{ has } s \text{ sink nodes}} w(G) \\
&\leq \sum_{B_1, \dots, B_s \text{ distinct}} \sum_{\substack{G \text{ has sink nodes} \\ \text{labeled } B_1, \dots, B_s}} w(G) \\
&\leq \sum_{B_1, \dots, B_s \text{ distinct}} \mu(B_1) \dots \mu(B_s) \\
&\qquad \text{by Proposition 3.1} \\
&\leq \frac{1}{s!} (\sum_{B \in \mathcal{B}} \mu(B))^s = \frac{W^s}{s!}
\end{aligned}$$

Now, suppose that the Resampling Algorithm runs for t time-steps, and consider the event $\mathcal{E}$ that $t \geq c(W + \frac{\log^2 n}{\epsilon})$ where c is some sufficiently large constant (to be determined).

Let $\hat{G}$ be the full witness DAG of the resulting execution. Each resampling at time $i \in \{1, \dots, t\}$ corresponds to some vertex v_i in $\hat{G}$.

By Proposition 2.2, all but $\frac{10 \log n}{\epsilon}$ single-sink witness DAGs contain at most $\frac{10 \log n}{\epsilon}$ nodes. Now, let X denote the set $\{i \mid |\hat{G}(v_i)| \leq h\}$ where $h = \frac{10 log n}{\epsilon}$. Under our assumption, we must have $|X| \geq t - \frac{10 \log n}{\epsilon}$; for c sufficiently large, we have $|X| \geq t/2$.

For each $i = 1, \dots, |X|$ let x_i denote the i^{th} largest element of X and let $u_i = v_{x_i}$, and let $H_i = G(u_i)$.

Suppose that we now select indices $|X| \geq i_1 > i_2 > i_3 > \cdots > i_{s-1} > i_s \geq 1$, satisfying $i_j \notin H_{i_1} \cup \cdots \cup H_{i_{j-1}}$ for all $j = 1, \dots, s$.

The subgraph $\hat{G}(u_{i_1}, \dots, u_{i_s})$ must contain exactly s sink nodes $u_{i_1}, \dots, u_{i_s}$. Furthermore, for any such choice of $i_1, \dots, i_j$, the resulting witness DAGs $\hat{G}(u_{i_1}, \dots, u_{i_s})$ are distinct. Finally, by Proposition 2.4 each such witness DAG is compatible with R.

Hence, the number of single-sink witness DAGs compatible with R must be at least the number of such s-tuples of indices, which is

$$\sum_{1 \leq i_1 \leq t/2} \sum_{\substack{i_2 < i_1 \\ i_2 \notin H_{i_1}}} \sum_{\substack{i_3 < i_2 \\ i_3 \notin H_{i_1} \cup H_{i_2}}} \cdots \sum_{\substack{i_s < i_{s-1} \\ i_s \notin H_{i_1} \cup H_{i_2} \cup \cdots \cup H_{i_{s-1}}}} 1$$

By Proposition 3.2 (which we defer to after this proof), under the assumption that $|H_j| \leq h$ for all j, this expression is at least $\binom{t/2-(s-1)h}{s} \geq \frac{(t/2-sh)^s}{s!}$.

Hence, we have shown that, barring low-probability events, $\mathcal{E}$ requires that the number of witness DAGs with s sink nodes compatible with R is at least $\frac{(t/2-sh)^s}{s!}$. As the expected number of such DAGs is at most $W^s/s!$, by Markov's inequality we have $P(\mathcal{E}) \leq W^s/(t/2-sh)^s$.

Now set $s = \frac{t}{4h}$. Then this can be bounded by

$$\begin{aligned}\frac{W^s}{(t/2-sh)^s} &= (4W/t)^s \\ &\leq 2^{-s} \qquad \text{for } t \geq 8W \\ &= n^{-\Omega(1)} \qquad \text{for } t \geq \Omega(\frac{\log^2 n}{\epsilon})\end{aligned}$$

And thus $P(\mathcal{E}) \leq n^{-\Omega(1)}$ as well.

COROLLARY 3.1. *The Resampling Algorithm performs $O(\frac{n}{\epsilon})$ resamplings whp.*

Proof. We have $W = \sum_B \mu(B) \leq \sum_i \sum_{B \sim i} \mu(B) \leq n/\epsilon$. Thus, by Theorem 3.1, with high probability the total number of resamplings made the Resampling Algorithm is at most $O(\frac{n}{\epsilon} + \epsilon^{-1} \log^2 n) = O(\frac{n}{\epsilon})$.

For the symmetric LLL, we can even obtain concentration without the need for the multiplicative ϵ-slack.

COROLLARY 3.2. *If the symmetric LLL criterion $ep(d+1) \leq 1$ is satisfied, then whp the number of resamplings is $O(n + d \log^2 n)$.*

Proof. Set $x(B) = \frac{1}{d+1}$ for all $B \in \mathcal{B}$. Now a simple calculation shows that we satisfy the asymmetric LLL condition for $\epsilon = e(d/(1+d))^d - 1 = \Omega(1/d)$. Thus $\mu(B) \leq x(B)/(1-x(B)) = 1/d$, and so $W \leq m/d$. We also may observe that $m \leq nd$. So, by Theorem 3.1, the total number of resamplings is, with high probability $O(n + d \log^2 n)$.

To finish this proof, we need to show the following simple combinatorial bound:

PROPOSITION 3.2. *Suppose that A is a set of positive integers of cardinality $|A| = t$. Suppose that for each $j \in \mathbf{Z}$ there is a set of positive integers I_j, with $|I_j| \leq h$ for all j. Define*

$$f(A,s,I) = \sum_{i_1 \in A} \sum_{\substack{i_2 < i_1 \\ i_2 \in A - I_{i_1}}} \sum_{\substack{i_3 < i_2 \\ i_3 \in A - I_{i_1} - I_{i_2}}} \cdots \sum_{\substack{i_s < i_{s-1} \\ i_s \in A - I_{i_1} - \cdots - I_{i_{s-1}}}} 1$$

Then we have

$$f(A,s,I) \geq \binom{t-(s-1)h}{s}$$

Proof. We prove this by induction on s. When $s = 1$ we have

$$f(A,1,I) = \sum_{i_1 \in A} 1 = t = \binom{t-(1-1)h}{1}$$

as claimed.

So we consider the induction step. Suppose that $A = \{a_1, \ldots, a_t\}$, and suppose that we select the value $i_1 = a_j$. Then observe that the remaining sum over $i_2, \ldots, i_s$ is equal to $f(A'_j, s-1, I)$, where $A'_j = \{a_1, \ldots, a_{j-1}\} - I_{i_1}$ which is a set of cardinality at least $j - 1 - h$.

Summing over all $j = 1, \ldots, t$ gives us:

$$\begin{aligned}f(A,s,I) &= \sum_{j=1}^{t} f(A'_j, s-1, I) \\ &\geq \sum_{j=(s-1)h+s}^{t} f(A'_j, s-1, I) \\ &\geq \sum_{j=(s-1)(h+1)}^{t} \binom{(j-1-h)-(s-2)h}{s-1} \\ &\qquad \text{by inductive hypothesis} \\ &= \sum_{j=s-1}^{t-1-h(s-1)} \binom{j}{s-1} = \binom{t-(s-1)h}{s}\end{aligned}$$

and the induction is proved.

4 Mutual consistency of witness DAGs

In Section 2, we have seen conditions for witness DAGs to be compatible with a *given* resampling table R. In this section, we examine when a set of witness DAGs can be mutually consistent, in the sense that they could all be prefixes of some (unspecified) full witness DAG.

DEFINITION 4.1. (CONSISTENCY OF G, G') *Let G, G' be witness DAGs. We say that G is* consistent *with G' is, for all variables i, either $G[i]$ is an initial segment of $G'[i]$ or $G'[i]$ is an initial segment of $G[i]$, both of these as labeled graphs. (Carefully note the presence of the quantifier here: If $n = 2$ and $G[1]$ is an initial segment of $G'[1]$ and $G'[2]$ is an initial segment of $G[2]$, then G, G' are compatible.)*

Let $\mathcal{G}$ be any set of witness DAGs. We say that $\mathcal{G}$ is pairwise consistent *if G, G' are consistent with each other for all $G, G' \in \mathcal{G}$.*

PROPOSITION 4.1. *Suppose H_1, H_2 are prefixes of G. Then H_1 is consistent with H_2.*

Proof. Observe that for any $w_1 \prec w_2 \in H_j$, we must have $w_1 \in H_j$ as well. It follows that $H_j[i]$ is an initial segment of $G[i]$ for any $i \in [n]$. As both $H_1[i]$ and $H_2[i]$ are initial segments of $G[i]$, one of them must be an initial segment of the other.

DEFINITION 4.2. (MERGE) *Let G, G' be consistent witness DAGs. Then we define the* merge *$G \vee G'$ as follows.*

If either G or G' has a node v with an extended label (B,k), then we create a corresponding node $w \in G \vee G'$ labeled by B. We refer to the corresponding label *of w as (B,k).*

Now, let $v_1, v_2 \in G \vee G'$ have corresponding label (B_1,k_1) and (B_2,k_2). We create an edge from v_1 to v_2 if either G or G' has an edge between vertices with extended label $(B_1,k_1),(B_2,k_2)$ respectively.

PROPOSITION 4.2. *Suppose that, when forming $G \vee G'$, that $v \in G \vee G'$ has corresponding label (B,k). Then v has extended label (B,k).*

Proof. Because of our rule for forming edges in $G \vee G'$, the only edges that can go to v from other nodes labeled B, would have corresponding labels (B,l) for $l < k$. Thus, there are at most $k-1$ nodes labeled B with an edge to v.

On the other hand, there must be nodes with extended label (B,k) in G or G'; say without loss of generality the first. Then G must also have nodes with extended labels $(B,1),\ldots,(B,k-1)$. These correspond to vertices $w_1,\ldots,w_{k-1}$ with corresponding labels $(B,1),\ldots,(B,k-1)$, all of which have an edge to v. So there are at least $k-1$ nodes labeled B with an edge to v.

Thus, there are exactly k nodes in G with an edge to v and hence v has extended label (B,k).

By Proposition 4.2, for every vertex $v \in G$ or $v \in G'$, there is a vertex in $G \vee G'$ with the same extended label. We will abuse notation slightly, so that we refer to this vertex in H also by the name v.

PROPOSITION 4.3. *Let G, G' be consistent witness DAGs and let $H = G \vee G'$. If there is a path $v_1,\ldots,v_l$ in H and $v_l \in G$, then also $v_1,\ldots,v_l \in G$.*

Proof. Suppose that this path has corresponding labels $(B_1,k_1),\ldots,(B_l,k_l)$. Suppose $i \le l$ is minimal such that $v_i,\ldots,v_l$ are all in G. (This is well-defined as $v_l \in G$). If $i = 1$ we are done.

Otherwise, we have $v_i \in G, v_{i-1} \in G' - G$. Note that $B_{i-1} \sim B_i$, so let $j \in S_{B_{i-1}} \cap S_{B_i}$. Note that $v_i \in G[j], v_{i-1} \in G'[j]$. But observe that in H there is an edge from v_{i-1} to v_i. As $v_{i-1} \notin G$, this edge must have been present in G'. So $G'[j]$ contains the vertices v_{i-1}, v_i, in that order, while $G[j]$ contains only the vertex v_i. Thus, neither $G[j]$ or $G'[j]$ can be an initial segment of the other. This contradicts the hypothesis.

PROPOSITION 4.4. *Let G, G' be consistent witness DAGs and let $H = G \vee G'$. Then H is a witness DAG and both G and G' are prefixes of it.*

Proof. Suppose that H contains a cycle $v_1,\ldots,v_l,v_1$, and suppose $v_1 \in G$. Then by Proposition 4.3 the cycle $v_1,\ldots,v_l,v_1$ is present also in G, which is a contradiction.

Next, we show that the comparability condition holds for H. Suppose that (B_1,k_1) and (B_2,k_2) are the corresponding labels of vertices in H, and $B_1 \sim B_2$. So let $i \in S_{B_1} \cap S_{B_2}$. Without loss of generality, suppose that $G[i]$ is an initial segment of $G'[i]$. So it must be that (B_1,k_1) and (B_2,k_2) appear in $G'[i]$. Because of the comparability condition for G', there is an edge in G' on these vertices, and hence there is an edge in H as well.

Finally, we claim that $G = H(v_1,\ldots,v_l)$ where $v_1,\ldots,v_l$ are the vertices of G. It is clear that $G \subseteq H(v_1,\ldots,v_l)$. Now, suppose $w \in H(v_1,\ldots,v_l)$. Then there is a path $w, x_1, x_2, \ldots, x_l, v$ where the vertices $x_1,\ldots,x_l$ lie in H and $v \in G$. By Proposition 4.3, this implies that $w, x_1, \ldots, x_l, v \in G$. So $w \in G$ and we are done.

PROPOSITION 4.5. *The operation $\vee$ is commutative and associative.*

Proof. Commutativity is obvious from the symmetric way in which $\vee$ was defined. To show associativity, note that we can give the following symmetric characterization of $H = (G_1 \vee G_2) \vee G_3$. If G_1, G_2 or G_3 has a node labeled (B_1,k_1) then so does H. We have an edge from (B_1,k_1) to (B_2,k_2) if there is such an edge in G_1, G_2 or G_3.

PROPOSITION 4.6. *Suppose G_1, G_2 are consistent with each other and with some witness DAG G_3. Then $G_1 \vee G_2$ is consistent with G_3.*

Proof. For any variable $i \in [n]$, note that either $G_1[i]$ is an initial segment of $G_2[i]$ or vice-versa. Also note that $(G_1 \vee G_2)[i]$ is the longer of $G_1[i]$ or $G_2[i]$.

Now we claim that for any variable i, either $G_3[i]$ is an initial segment of $(G_1 \vee G_2)[i]$ or vice-versa. Suppose without loss of generality that $G_1[i]$ is an initial segment of $G_2[i]$. Then $(G_1 \vee G_2)[i] = G_1[i]$. By definition of consistency, either $G_1[i]$ is an initial segment of $G_3[i]$ or vice-versa. So $(G_1 \vee G_2)[i]$ is an initial segment of $G_3[i]$ or vice-versa.

In light of these propositions, we can unambiguously define, for any pairwise consistent set of witness DAGs $\mathcal{G} = \{G_1,\ldots,G_l\}$, the merge

$$\bigvee \mathcal{G} = G_1 \vee G_2 \vee G_3 \cdots \vee G_l$$

The notation suggests that this may depend on the ordering $G_1,\ldots,G_l$, but because of associativity and

commutativity this can be well-defined in terms of the un ordered set $\{G_1, \ldots, G_l\}$.

We can give another characterization of pairwise consistency, which is more illuminating although less explicit:

PROPOSITION 4.7. *The witness DAGs $G_1, \ldots, G_l$ are pairwise consistent iff there is some witness DAG H such that $G_1, \ldots, G_l$ are all prefixes of H.*

Proof. For the forward direction: let $H = G_1 \vee \cdots \vee G_l$. By Proposition 4.4, each G_i is a prefix of H. For the backward direction: by Proposition 2.5, any G_{i_1}, G_{i_2} are both prefixes of H, hence consistent.

PROPOSITION 4.8. *Let G_1, G_2 be consistent witness DAGs and R a resampling table. Then $G_1 \vee G_2$ is compatible with R iff both G_1 and G_2 are compatible with R.*

Proof. For the forward direction: let $v \in G_1$ labeled by B. By Proposition 4.3, we have $G_1(v) = (G_1 \vee G_2)(v)$. Thus for $i \in S_B$ we have $|G_1(v)[i]| = |(G_1 \vee G_2)(v)[i]|$. This implies that $X^v_{G_1} = X^v_{G_1 \vee G_2}$. By hypothesis, B is true on $X^v_{G_1 \vee G_2}$ and hence $X^v_{G_1}$. As this is true for all $v \in G_1$, it follows that G_1 is compatible with R. Similarly, G_2 is compatible with R.

For the backward direction: Let $v \in G_1 \vee G_2$. Suppose without loss of generality that $v \in G_1$. As in the forward direction, we have $X^v_{G_1} = X^v_{G_1 \vee G_2}$; by hypothesis B is true on the former so it is true on the latter. Since this holds for all $v \in G_1 \vee G_2$, it follows that $G_1 \vee G_2$ is compatible with R.

5 A new parallel algorithm for the LLL

In this section, we will develop a parallel algorithm to enumerate *all* the single-sink witness DAGs which are compatible with R. This will allow us to enumerate (implicitly) all witness DAGs compatible with R. In particular, we are able to simulate all the possible values for $\hat{G}$, the full witness DAG. We are able to do this without actually running the Resampling Algorithm.

In a sense, both the Parallel Resampling Algorithm and our new parallel algorithm are building up $\hat{G}$. However, the Parallel Resampling Algorithm does this layer by layer, in an inherently sequential way: it does not determine layer $i+1$ until it has fixed a value for layer i, and resolving each layer requires a separate MIS calculation.

Our algorithm dispenses with this MIS calculation at each stage. As a result, it is not able to completely resolve layer i before moving on to layer $i+1$. There are multiple possible values for layer i, and our algorithm computes all the possible layer $i+1$-witness DAGs which they could lead to. Although the number of such witness DAGs is exponential, we can still do this efficiently because they can be built out of single-sink witness DAGs compatible with R. These are only polynomial in number, and can be processed in parallel.

5.1 Collectible witness DAGs The goal of our algorithm is to enumerate the single-sink witness DAGs (nearly equivalently, the witness trees.) We will build them up node-by-node. However, in order to do so, we must keep track of a slightly more general type of witness DAGs, namely, those derived by removing the root node from a single-sink witness DAG. Such witness DAGs have multiple sink nodes, which are all at distance two in the dependency graph. Although this is a much larger set than the set of single-sink witness DAGs, it is still small enough to enumerate. This is very close to the concept of partial witness trees introduced in [3].

DEFINITION 5.1. (COLLECTIBLE WITNESS DAG) *Suppose we are given a witness DAG G, whose sink nodes are labeled $B_1, \ldots, B_s$. We say that G is* collectible to B *if $B \sim B_1, \ldots, B \sim B_s$.*

We say that G is collectible *if it is collectible to some $B \in \mathcal{B}$. Note that if G has a single sink node labeled by B, it is collectible to B.*

PROPOSITION 5.1. *Define*

$$W' = \sum_{B \in \mathcal{B}} \frac{1}{P_\Omega(B)} \sum_{\substack{\text{single-sink witness DAGs } \tau \\ \text{with root node labelled } B}} w(\tau)$$

The expected total number of collectible witness DAGs compatible with R is at most W'.

Proof. Suppose that G is a witness DAG collectible to B. Then define G' by adding to G a new sink node labeled by B. As all the sink nodes in G are labeled by $B' \sim B$, now G' is a single-sink witness DAG containing $r+1$ nodes.

Now

$$P(G \text{ compatible with } R) = w(G) = \frac{w(G')}{P_\Omega(B)}$$

The total probability that there is some G compatible with R and collectible to B, is at most the sum over all such G. When we sum over all such witness DAGs G, then each witness DAG G' with a single sink node labeled by B appears at most once in the sum. Hence,

we have

$$\begin{aligned}\sum_{G \text{ collectible}} w(G) &\leq \sum_{\substack{B\in\mathcal{B}\\ G \text{ collectible to } B}} w(G)\\ &\leq \sum_{\substack{B\in\mathcal{B}\\ G \text{ collectible to } B}} \frac{w(G')}{P_\Omega(B)}\\ &\leq \sum_{B\in\mathcal{B}} \sum_{\substack{\text{single-sink witness DAGs } \tau\\ \text{rooted in } B}} \frac{w(\tau)}{P_\Omega(B)}\\ &= W'\end{aligned}$$

COROLLARY 5.1. *We have* $W' \leq \sum_{B\in\mathcal{B}} \frac{\mu(B)}{P_\Omega(B)}$.

Proof. This follows from Proposition 2.5.

The parameter W', which dictates the run-time of our parallel algorithm, has a somewhat complicated behavior. For most applications of the LLL where the bad events are "balanced," we have $W' \approx m$. For example, consider the symmetric LLL setting:

PROPOSITION 5.2. *If the symmetric LLL criterion* $ep(d+1) \leq 1$ *is satisfied then* $W' \leq me$.

Proof. Observe that the asymmetric LLL criterion is satisfied by setting $x(B) = \frac{eP_\Omega(B)}{1+eP_\Omega(B)}$ for all $B \in \mathcal{B}$. Now by Theorem 1.4, we have $\mu(B) \leq eP_\Omega(B)$ for all $B \in \mathcal{B}$. So $W' \leq me$.

More generally, W' is small as long as none of the bad events has a probability which is too small.

PROPOSITION 5.3. *Let* $p : \mathcal{B} \to [0,1]$ *be a vector satisfying the two conditions:*

1. $P_\Omega(B) \leq p(B)$ *for all* $B \in \mathcal{B}$
2. p *satisfies the Shearer criterion with* ϵ*-multiplicative slack.*

Then we have

$$W' \leq \frac{(n/\epsilon)}{\min_{B\in\mathcal{B}} p(B)}$$

Proof. For any witness DAG G whose nodes are labeled $B_1, \ldots, B_s$, define $w'(G)$ to be $p(B_1)\cdots p(B_s)$.

Now consider some single-sink witness DAG τ with root node labeled by B and s additional nodes labeled $B_1, \ldots, B_s$. We have that

$$\frac{w(G)}{P_\Omega(B)} = \prod_{i=1}^{s} P_\Omega(B_i) \leq \prod_{i=1}^{s} p(B_i) \leq \frac{w'(G)}{p'(B)}$$

Thus, we have that

$$\begin{aligned}W' &\leq \sum_{B\in\mathcal{B}} \frac{1}{p(B)} \sum_{\tau \text{ rooted in } B} w'(\tau)\\ &\leq \frac{\sum_{\text{single-sink witness DAGs } \tau} w(\tau)}{\min_{B\in\mathcal{B}} p(B)}\\ &\leq \frac{(n/\epsilon)}{\min_{B\in\mathcal{B}} p(B)} \qquad \text{by Corollary 2.1}\end{aligned}$$

On the other hand, for instances in which there are some bad events which have very low probability and very high dependency, then W' can become exponentially large.

5.2 Algorithmically enumerating witness DAGs In the Moser-Tardos setting, the witness trees were not actually part of the algorithm but were a theoretical device for analyzing it. In our algorithm, we will operate directly on witness DAGs. The following algorithm draws a random resampling table and then builds a list of witness DAGs compatible with it:

1. Randomly sample the resampling table R.
2. For each bad event B true in the initial configuration $R(\cdot, 0)$, create a graph with a single vertex labeled B. We denote this initial set by F_1.
3. For $k = 1, 2, \ldots K$:
   4. For each consistent pair of witness DAGs $G_1, G_2 \in F_k$, form $G' = G_1 \vee G_2$. If G' is collectible, then add it to F_{k+1}.
   5. For each witness DAG $G \in F_k$ which is collectible to B, create a new witness DAG G' by adding to G a new sink node labeled by B. If G' is compatible with R then add it to F_{k+1}.
   6. Finally, add every $G \in F_k$ to F_{k+1}. (So that $F_k \subseteq F_{k+1}$).

We will show that for $K = O(\epsilon^{-1}\log(\epsilon^{-1}n))$, with high probability this algorithm generates all the single-sink witness DAGs compatible with R.

PROPOSITION 5.4. *Let* $G \in F_k$ *for any integer* $k \geq 1$. *Then* G *is compatible with* R.

Proof. We show this by induction on k. When $k = 1$, then $G \in F_1$ is a singleton node v labeled by B. Note that $X^v_G(i) = R(i, 0)$ for all $i \in S_B$, and so B is true on X^v_G. So G is compatible with R.

Now for the induction step. Suppose first G was formed by $G = G_1 \vee G_2$, for $G_1, G_2 \in F_{k-1}$. By

induction hypothesis, G_1, G_2 are compatible with R. So by Proposition 4.6, G is compatible with R. Second suppose G was formed in step (5), so by definition it must be compatible with R.

PROPOSITION 5.5. *Suppose that G is a collectible witness DAG with k nodes compatible with R. Then $G \in F_k$.*

Proof. We show this by induction on k. When $k = 1$, then G is a singleton node v labeled by B, and $X^v_G(i) = R(i, 0)$. So B is true on the configuration $R(\cdot, 0)$, and so we put G into F_1.

For the induction step, first suppose G has a single sink node v labeled by B. If G has only one vertex, then $G \in F_1 \subseteq F_k$. So we may suppose G has multiple nodes. Now consider the witness DAG $G' = G - v$. This witness DAG has at most $r - 1$ nodes. Also, all the sink nodes in G' must be labeled by some $B' \sim B$ (as otherwise they would remain sink nodes in G). So G' is collectible to B. So, by induction hypothesis, $G' \in F_{k-1}$. Now iteration $k - 1$ transforms the graph $G' \in F_{k-1}$ into G (by adding a new sink node labeled by B), and so $G' \in F_k$ as desired.

Next, suppose that G is a witness DAG with multiple sink nodes $v_1, \ldots, v_s$ labeled by $B_1, \ldots, B_s$, with the property $s \geq 2$ and that $B \sim B_1, \ldots, B_s$ for some $B \in \mathcal{B}$. Let $G' = G(v_1)$ and let $G'' = G(v_2, \ldots, v_s)$. Note that G' is missing the vertex v_s and similarly G'' is missing the vertex v_1. So G', G'' have strictly less than k nodes. Also, note that G', G'' are collectible to B.

Finally, observe that $G = G' \vee G''$. Clearly every vertex in G appears in G' or G''. Also, by Proposition 4.1, G, G' are both prefixes of G and hence are compatible with R.

So by induction hypothesis, we have $G', G'' \in F_{k-1}$, and thus $G = G' \vee G''$ is added to F_k in step (4).

PROPOSITION 5.6. *Suppose that we can check whether any given bad event is true in time $O(\log n)$ and poly(m) processors. Then whp, this procedure enumerates all single-sink witness DAGs compatible with R, with the following complexity bounds:*

$$\tilde{O}(\epsilon^{-1}(\log \epsilon^{-1} n)(\log(W'\epsilon^{-1} n)) \text{ time}$$
$$(W'\epsilon^{-1} n)^{O(1)} \text{ processors.}$$

Proof. We have shown that F_k contains all the collectible witness DAGs compatible with R using at most k nodes. Furthermore, by Corollary 2.2, whp all the single-sink witness DAGs compatible with R contain at most $O(\epsilon^{-1} \log(\epsilon^{-1} n))$ nodes. Hence, for $K = O(\epsilon^{-1} \log(\epsilon^{-1} n))$, we have that with high probability F_K contains all such single-sink witness DAGs.

Furthermore, the expected total number of such DAGs is at most W'. Hence, with high probability, the total number of such DAGs is at most $W' n^{O(1)}$. Each DAG could involve up to $n\epsilon^{-1} \log n$ cells from the resampling table. So we can store the entire collection of such DAGs using $(W'\epsilon^{-1} n)^{O(1)}$ processors.

We describe in Section 5.5 further details about how these witness DAGs can be processed. Given that there are $M = (W'n)^{O(1)}$ total DAGs and that each bad event can be checked in $O(\log n)$ time, we can implement individual steps using $(Mm\epsilon^{-1} n)^{O(1)}$ processors and $\tilde{O}(\log(M\epsilon^{-1} mn))$ time. Observe that $W' \geq m$, so this can be simplified to $(W'\epsilon^{-1} n)^{O(1)}$ etc.

(We note that the condition that individual bad event can be checked in time $O(\log n)$ and polynomial processors is much weaker than the condition that there is a Bad-Event Checker running in time $O(\log n)$ and poly(n) processors. In the former case, the total number of processors for checking bad events can become as larger as poly(m).)

5.3 Producing the final configuration So far, our parallel algorithm has generated the complete set of single-sink witness DAGs compatible with R. We can define a graph $\mathcal{G}$, whose nodes correspond to such single-sink witness DAGs, with an edge between DAGs if they are pairwise inconsistent. Let $\mathcal{I}$ be a maximal independent set of $\mathcal{G}$, and let $G = \bigvee \mathcal{I}$. Now define the configuration X^*, which we refer to as the *final configuration*, by

$$X^*(i) = R(i, |G[i]| + 1)$$

for all $i \in [n]$.

The final stage of our algorithm is to output X^*.

PROPOSITION 5.7. *With high probability, no bad event $B \in \mathcal{B}$ is true on the configuration X^*.*

Proof. Suppose that B is true on X^*. Now define the witness DAG H by adding to G a new sink node v labeled by B. Observe that G is a prefix of H. By Proposition 4.1 H, G are consistent.

We claim that H is compatible with R. By Proposition 2.4, G is compatible with R so this is clear for all the vertices of H except for its sink node v. For this vertex, observe that for each $i \in S_B$ we have $X^v_H(i) = R(i, |H[i]|) = R(i, |G[i]| + 1) = X^*(i)$. By Proposition 2.4, this implies that $H(v)$ is compatible with R as well.

So $H(v)$ is a single-sink witness DAG compatible with R. By Proposition 5.6, with high probability $H(v) \in F_K$. So $H(v)$ is a node of $\mathcal{G}$. Observe that $H(v)$ and all the witness DAGs $G' \in \mathcal{I}$ are prefixes of H. By

Proposition 4.7, $H(v)$ is pairwise consistent with all of them. As $\mathcal{I}$ was chosen to be a maximal independent set, this implies that $H(v) \in \mathcal{I}$.

By Proposition 4.4, this implies that $H(v)$ is a prefix of G. This implies that $|G[i]| \geq |H(v)[i]|$ for any variable i. But for $i \in S_B$ we have $|H(v)[i]| = |H[i]| = |G[i]| + 1$, a contradiction.

Putting this all together gives a faster algorithm for the LLL.

THEOREM 5.1. *Suppose we satisfy the Shearer criterion with multiplicative ϵ-slack. Suppose we can check, for any bad event B and any configuration, if B is true in time $O(\log n)$. Then there is an algorithm to find a configuration avoiding $\mathcal{B}$, running in time $\tilde{O}(\epsilon^{-1}(\log \epsilon^{-1} n)\log(W'\epsilon^{-1}n))$ and using $(W'\epsilon^{-1}n)^{O(1)}$ processors.*

Proof. By Proposition 5.6, enumerate all the single-sink witness DAGs using $\tilde{O}(\epsilon^{-1}(\log \epsilon^{-1} n)(\log(W'\epsilon^{-1}n))$ time and $(W'\epsilon^{-1}n)^{O(1)}$ processors.

Whp, the total number of such single-sink witness DAGs is $Wn^{O(1)}$. Using Luby's MIS algorithm, one find a maximal independent set of such DAGs in time $O(\log^2(Wn))$ and using $(Wn)^{O(1)}$ processors. Note that $W \leq n/\epsilon$ and so this running time is dominated by the enumeration of the single-sink witness DAGs.

Finally, one can form the configuration X^* as indicated in Proposition 5.7 in time $\log(W\epsilon^{-1}n)$ and using $(W\epsilon^{-1}n)^{O(1)}$ processors. This configuration avoids all bad events, as desired.

COROLLARY 5.2. *Suppose that the symmetric LLL criterion is satisfied with ϵ-multiplicative slack, i.e., $ep(1+\epsilon)(d+1) \leq 1$, and we can determine if any bad event B is true on a given configuration in time $O(\log n)$. Then we can find a configuration avoiding $\mathcal{B}$ in expected time $\tilde{O}(\epsilon^{-1}\log(mn)\log n)$ and using $(mn)^{O(1)}$ processors.*

Proof. We have $W = \sum_{B \in \mathcal{B}} ep \leq O(m/d)$. By Proposition 5.2, we have $W' \leq me$. Now note that $m \leq nd$, so $W \leq O(n)$.

Next, note that even if $ep(d+1) = 1$, then we can still satisfy the Shearer criterion with multiplicative ϵ-slack, for $\epsilon = \Omega(1/d)$. Hence, we can assume that $\epsilon^{-1} \geq \Omega(1/m)$ and hence the terms $\log \epsilon^{-1}$ can be upper-bounded by $\log m$.

Now apply Theorem 5.1.

5.4 A heuristic lower bound In this section, we give some intuition as to why we believe that the run-time of this algorithm, namely $O(\epsilon^{-1}\log^2 n)$, is essentially optimal for LLL algorithms *which are based on the resampling paradigm*. We are not able to give a formal proof, because we do not have any fixed model of computation in mind. Also it is not clear whether our new algorithm is based on resampling.

Suppose we are given a problem instance on n variables whose distributions are all Bernoulli-q, where $q \in [0,1]$ is a parameter to be chosen. The space $\mathcal{B}$ consists of $\sqrt{n}$ bad events, each of which is a threshold function on $\sqrt{n}$ variables, and all these events are completely disjoint from each other. By adjusting the exact threshold used and the parameter q, we can ensure that the probability p of each bad event event is $p = 1 - \epsilon$.

The number of resamplings of each event is a geometric random variable, and it is not hard to see that with high probability there will be some bad event B which requires $\Omega(\epsilon^{-1}\log n)$ resamplings in order to cause B to become false.

Also, note that whenever we perform a resampling of B, we must compute whether B is currently true. This requires computing a sum of $\sqrt{n}$ binary variables, which itself requires time $\Omega(\log n)$.

Thus, the overall running time of this algorithm must be $\Omega(\epsilon^{-1}\log^2 n)$.

The reason we consider this a *heuristic* lower bound is that, technically, the parallel algorithm we have given is not based on resampling. That is, there is no current "state" of the variables which is updated as bad events are discovered. Rather, all possible resamplings are precomputed in advance from the table R.

5.5 Processing the witness DAGs In our algorithm, we have assumed that if we have M total witness DAGs under consideration, that we can perform the basic operations of the parallel algorithm in $\tilde{O}(\log(Mmn))$ time using $(Mmn)^{O(1)}$ processors on a PRAM.

The main task we must perform is, given two witness DAGs G_1, G_2, we must first determine if G_1, G_2 are consistent and if so form $G = G_1 \vee G_2$. Next, we must check if G is collectible to some B. A related task is: given a graph G collectible to some B, create a new graph G' which has an added sink node labeled B.

If we worked with the full graph structure of the witness DAGs then these steps might appear to require a traversal of the graphs. However, we only need to store a limited amount of information about these DAGs. Namely, we must store the list of all sink nodes, and we must store the association between entries of R and the corresponding bad events. That is, for each variable $i \in [n]$ and each $t \leq O(\epsilon^{-1}\log n)$, we must determine a list of all the nodes $v \in G$ and their labels (B,k) such that $G(v)[i] = t$. Using this information, we may easily determine if G_1, G_2 are consistent. It is also straightforward to compute this association table for

$G_1 \vee G_2$, given the association tables for the individual graphs. (We simply merge the lists; this can be done using standard parallel sorting algorithms).

We can likewise determine the sink nodes of $G_1 \vee G_2$. Using our association table, we can determine if any sink node of G_1 appears in G_2; if so, this node is a sink node of $G_1 \vee G_2$ if is also a sink node of G_2. If the sink node of G_1 does not appear in G_2, then it becomes a sink node in $G_1 \vee G_2$, and so forth.

Next, we enumerate in parallel over all $B \in \mathcal{B}$. Suppose we are given a fixed B and a fixed G; we want to determine if G is collectible to B. We can check, in parallel, whether the sink nodes of G overlap the variables in B; this takes time $O(\log n)$ and $n^{O(1)}$ processors. We then check if every sink node of G overlapped in some variable; this takes another $O(\log n)$ time and $n^{O(1)}$ processors.

Finally, we need to determine if we can form a new graph G' by adding a new sink node v labeled B to G. In addition to the graph-theoretic structure needed for this, we need to check if B is true on the configuration X^v. This will be possible under our assumption that we can check if a bad event is true in time T.

Other operations used in our algorithm can be handled in similar ways.

6 A deterministic variant

In [3], a deterministic parallel (NC) algorithm is given for the LLL. This algorithm requires an additional slack compared to the Parallel Resampling Algorithm (which in turn requires additional slack compared to the sequential algorithm). Although [3] gives a general asymmetric criterion, it is quite technical and has many parameters. We will discuss the simpler symmetric setting. In that case, the algorithm of [3] requires that

$$epd^{1+\epsilon} < 1$$

There are additional constraints on how the bad events are represented. Again, these can be fairly technical, so we will focus on the simplest scenario: the set $\mathcal{B}$ contains m bad events, which are each explicitly represented as *atomic* events (that is, they are a conjunction of terms of the form $X_i = j$). The paradigmatic example of this setting is the k-SAT problem. We will also suppose that $m \gg n$ (in fact, typically m is exponentially larger than n), to simplify the notation. In this simplified setting, their algorithm requires $O(\epsilon^{-1} \log^3 m)$ time and $m^{O(1/\epsilon)}$ processors.

In the parallel algorithm as given, we assume that R is drawn from a completely independent probability distribution. Such probability distributions have exponentially large support. A key result from [3] is that substantially less independence is required.

DEFINITION 6.1. *We say a probability space Ω' is k-wise, ϵ-approximately independent, if for all subsets of variables $X_{i_1}, \ldots, X_{i_k}$, and all possible valuations $j_1, \ldots, j_k$, we have*

$$\Big| P_{\Omega'}(X_{i_1} = j_1 \wedge \cdots \wedge X_{i_k} = j_k) - P_{\Omega}(X_{i_1} = j_1 \wedge \cdots \wedge X_{i_k} = j_k) \Big| \leq \epsilon$$

THEOREM 6.1. ([6]) *There are k-wise, ϵ-approximately independent probability spaces which have a support of size poly$(\log n, 2^k, \epsilon^{-1})$.*

PROPOSITION 6.1. ([3]) *Suppose that $\mathcal{B}$ consists of atomic events.*

There are sufficiently large constants c, c' such that the following holds: Suppose that R is drawn from a probability distribution which is $m^{-c/\epsilon}$-approximately, $\log m$-wise independent. Then, with probability $> 1/2$, there are no single-sink witness DAGs compatible with R containing at least $c' \frac{\log m}{\epsilon \log d}$ nodes.

Note that this event only depends on the entries $R(i, x)$ for $x \leq \log m / \log d$. Hence, this event only depends on polynomially many entries in R. By Theorem 6.1, probability spaces with the required level of independence on this number of elements exist which are supported on only $m^{O(1/\epsilon)}$ events

There is one major difference between the RNC algorithm of Section 5 and the NC algorithm in this section. In Section 5, we only enumerated witness DAGs compatible with R. Potentially, there could be many collectible witness DAGs not compatible with R, but we never need to deal with them. For our NC algorithm, we will enumerate all single-sink witness DAGs, and then we later check whether they are compatible with R. The stronger slack condition $epd^{1+\epsilon} < 1$ is needed to ensure that this process remains bounded.

We give a deterministic algorithm to enumerate single-sink DAGs, with no restriction on their compatibility with any resampling table R:

1. Initialize F_1 by creating, for each $B \in \mathcal{B}$, a single-node DAG with a vertex labeled by B.
2. For $k = 1, \ldots, K = \frac{c \log m}{\epsilon \log d}$:
    3. Suppose that G, G' are witness DAGs in F_k with sink nodes v, v' labeled B, B' where $B \sim B'$. Create a new witness DAG G'' as follows. The nodes of G'' are the union of the nodes in G, G'. We add an edge from v' to v. Also, for any pair of nodes $w \in G, w' \in G'$ (other than $w = v, w' = v'$), if w and w' are labeled by dependent bad events, we add an edge from w to w'. If G'' has $\leq k + 1$ nodes, add it F_{k+1}

PROPOSITION 6.2. *Suppose that G is a single-sink witness DAG containing k nodes. Then $G \in F_k$.*

Proof. For $k = 1$ this is clear.

Suppose G is a single-sink witness DAG containing $k > 1$ nodes. Let v be the sink nodes of G and let v' be the sink node of $G - v$. Let X be the subset of nodes of $G - v$ which are disconnected from v in the graph $G - v'$.

Now let G_1 be the subgraph of G induced on the vertices $\{v'\} \cup X$, and let G_2 the subgraph of G induced on the remaining vertices of G. Note that G_1 is a single-sink witness DAG, with sink node v'; the reason is that since every node in G has a path to v, it follows that every node in X has a path to v through v' and hence every node in G' has a path to v'. Also, G_2 is a single-sink witness DAG. For, every node outside X has a path in G to v avoiding v', and this path remains in G_2. Also, the vertices of G_1, G_2 clearly partition the vertices of G.

Both G_1, G_2 are missing at least one vertex from G: G_1 is missing v and G_2 is missing v'. Hence, by inductive hypothesis, we have $G_1, G_2 \in F_{k-1}$.

Now let G'' be the result of applying step (3) of the above parallel algorithm with the graphs G_1, G_2. We claim that $G = G''$. The labeled nodes of G'' are clearly as in G. Also, G'' contains an edge from v' to v and so does G. Now, consider any pair of vertices $w, w' \in G$, other than v, v'.

If w, w' have no edge in G, then they also have no edge in G''.

If w, w' both lie in X, or both lie outside X, then they have edges in the induced subgraphs G_1, G_2 respectively; hence they have edges in G''.

So suppose $w \in X, w' \notin X$ and there is an edge connecting w to w'. Then $w \in G_1, w' \in G_2$. We claim that the edge must go from w' to w. For, there is a v'-avoiding path from w' to v; if there was an edge connecting w to w', then this could be extended to a v'-avoiding path from w to v, which would imply that $w \notin X$. But, by definition of G'', there is an edge from w' to w in G'', as desired.

The number of processors requires is $\text{poly}(F_K, n)$; we will show that this is polynomial in m, n. Also, critically, we show this enumerates *every* single-sink witness DAG.

PROPOSITION 6.3. *The total number of single-sink witness DAGs containing k vertices is at most $m(ed)^k$. In particular, for $k \le K = \frac{c \log m}{\epsilon \log d}$ this is $m^{O(1/\epsilon)}$.*

Proof. Taking advantage of the correspondence between single-sink witness DAGs and witness trees, it suffices to bound the number of witness trees. There are m choices for the label and there are at most $(ed)^k$ choices for the tree structure (using the standard formula counting labeled d-ary tree structures with k nodes).

Thus, we can enumerate all sufficiently large single-sink witness DAGs. Given a fixed witness DAG G and a fixed resampling table R, we can easily check if G is compatible with R. Thus, after enumerating F_K, we can filter it down to obtain F'_K, which is considerably smaller.

PROPOSITION 6.4. *Suppose R is drawn from a probability distribution which is $m^{-c/\epsilon}$-approximately, $\log m$-wise independent. Then, for c sufficiently large, the expected total number of single-sink witness DAGs compatible with R is $m^{O(1)}$.*

Proof. The event that a k-node witness DAG is compatible with R is a conjunction of events corresponding to the vertices in G. Each such event depends on at most d variables, so in total this is an atomic event depends on at most $kd \le \log m$ terms. So, by Definition 6.1, this event also has probability $\le w(G) + m^{-c/\epsilon}$. Now sum over all single-sink witness DAGs. The term $w(G)$ sums to $O(m)$ and the term $m^{-c/\epsilon}$ sums to $O(1)$ for c sufficiently large.

THEOREM 6.2. *There is a deterministic algorithm running in time $\tilde{O}(\epsilon^{-1} \log^2 m)$ and using $m^{O(1/\epsilon)}$ processors to find a configuration avoiding $\mathcal{B}$.*

Proof. We can enumerate F_K in time $O(\log^2 m)$ and using $m^{O(1/\epsilon)}$ processors. Next, we form a probability space for drawing R which is $m^{-c/\epsilon}$-approximately, $\log m$-wise independent, and is supported on $m^{O(1/\epsilon)}$ elements. Each processor explores a single event in this space.

For each R, we filter down the set F_k to the smaller set F'_k consisting of witness DAGs compatible with R. We proceed as for the randomized algorithm: we define a graph $\mathcal{G}$, whose nodes correspond to such single-sink witness DAGs compatible with R, with an edge between DAGs if they are pairwise inconsistent. By Propositions 6.1, 6.4, there is a positive probability that the two events jointly occur:

1. F'_K contains $m^{O(1)}$ nodes
2. There are no witness DAGs outside F'_K that are compatible with R.

For a fixed resampling table R, we can find a maximal independent subset $\mathcal{I} \subseteq \mathcal{G}$, using time $O(\log^2 m)$ and using $m^{O(1)}$ processors. Let $G = \bigvee \mathcal{I}$. As in Proposition 5.7, defining $X^*(i) = R(i, |G[i]| + 1)$ gives a configuration avoiding all bad events.

It requires $O(\epsilon^{-1}\frac{\log^2 m}{\log d})$ time to enumerate F'_K and it require $O(\log^2 m)$ to generate an MIS of it. Thus the total time is $O(\epsilon^{-1}\log^2 m)$ and the total processor count is $m^{O(1/\epsilon)}$.

7 Acknowledgments

Thanks to Aravind Srinivasan and Navin Goyal, for helpful discussions and comments. Thanks to the anonymous reviewers, for many helpful comments and corrections.

References

[1] Achlioptas, D., Iliopoulos, F.: Random walks that find perfect objects and the Lovász Local Lemma. Foundations of Computer Science (2014).

[2] Bissacot, R., Fernandez, R., Procacci, A., Scoppola, B.: An improvement of the Lovász Local Lemma via cluster expansion. Combinatorics, Probability and Computing 20-5, pp. 709-719 (2011).

[3] Chandrasekaran, K., Goyal, N., Haeupler, B.: Deterministic algorithms for the Lovász local lemma. SIAM Journal on Computing 42-6 , pp. 2132-2155 (2013)

[4] Chung, K., Pettie, S., Hsin-Hao, S.: Distributed algorithms for the Lovász local lemma and graph coloring. PODC 2014, pp. 134-143 (2014)

[5] Erdős, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal, R. Rado, and V. T. Sos, eds. Infinite and Finite Sets II, pp. 607-726 (1975).

[6] Even, G., Goldreich, O., Luby, M., Nisan, N., Velickovic, B.: Efficient approximation of product distributions. Random Structures and Algorithms, 13(1), pp. 1-16 (1998)

[7] Ghaffari, M.: Towards an optimal distributed algorithm for maximal independent set. arxiv:1506.05093. (2015)

[8] Haeupler, B., Saha, B., Srinivasan, A.: New constructive aspects of the Lovász Local Lemma. Journal of the ACM, 58-6, 2011.

[9] Harris, D.: Lopsidependency in the Moser-Tardos framework: beyond the Lopsided Lovász Local Lemma. SODA 2015.

[10] Harvey, N., Vondrak, J.: An algorithmic proof of the Lopsided Lovász Local Lemma. Arxiv 1504.02044 (2015). To appear, Proc. IEEE Symposium on Foundations of Computer Science, 2015.

[11] Kolipaka, K., Szegedy, M.: Moser and Tardos meet Lovász. Symposium on Theory of Computing, pp. 235-244 (2011).

[12] Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM Journal on Computing 15-4, pp. 1036-1053 (1996).

[13] Moser, R. : Derandomizing the Lovász Local Lemma more effectively. ArXiv abs/0807.2120, pp. 1-8 (2008).

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

[15] Shearer, J. B.: On a problem of Spencer. Combinatorica 5, pp. 241-245 (1985).