Deterministic Algorithms for the Lovász Local Lemma

Karthekeyan Chandrasekaran[*]   Navin Goyal[†]   Bernhard Haeupler[‡]

Abstract

The Lovász Local Lemma [5] (LLL) is a powerful result in probability theory that states that the probability that none of a set of bad events happens is nonzero if the probability of each event is small compared to the number of events that depend on it. It is often used in combination with the probabilistic method for non-constructive existence proofs. A prominent application is to k-CNF formulas, where LLL implies that, if every clause in the formula shares variables with at most $d \leq 2^k/e$ other clauses then such a formula has a satisfying assignment. Recently, a randomized algorithm to efficiently construct a satisfying assignment was given by Moser [12]. Subsequently Moser and Tardos [13] gave a randomized algorithm to construct the structures guaranteed by the LLL in a very general algorithmic framework. We address the main problem left open by Moser and Tardos of derandomizing these algorithms efficiently. Specifically, for a k-CNF formula with m clauses and $d \leq 2^{k/(1+\epsilon)}/e$ for some $\epsilon \in (0,1)$, we give an algorithm that finds a satisfying assignment in time $\tilde{O}(m^{2(1+1/\epsilon)})$. This improves upon the deterministic algorithms of Moser and of Moser-Tardos with running time $m^{\Omega(k^2)}$ which is superpolynomial for $k = \omega(1)$ and upon other previous algorithms which work only for $d \leq 2^{k/16}/e$. Our algorithm works efficiently for a general version of LLL under the algorithmic framework of Moser and Tardos [13], and is also parallelizable, i.e., has polylogarithmic running time using polynomially many processors.

---

[*]karthe@gatech.edu

[†]navingo@microsoft.com

[‡]haeupler@mit.edu

1   Introduction

The Lovász Local Lemma [5] (LLL) is a powerful result in probability theory that states that the probability that none of a set of bad events happens is nonzero if the probability of each event is small compared to the number of events that depend on it. LLL is often used in combination with the probabilistic method to prove the existence of certain structures. For this, one designs a random process guaranteed to generate the desired structure if none of a set of bad events happen. If those events fall under the above assumption, the Lovász Local Lemma guarantees that the probability that the random process builds the desired structure is greater than zero, thereby implying its existence. Often, the probability of this good event is exponentially small. Consequently, the same random process cannot directly and efficiently find the desired structure. The original proof of the Lovász local lemma [5] is non-constructive in this sense. Starting with the work of [2], a series of papers [1, 4, 10, 15, 11] have sought to make the LLL constructive. We note that in most applications where LLL is useful (e.g., [7, 8, 9]), the proof of existence of the desired structure is known only through LLL.

Algorithms for the LLL are often formulated for one of the two model problems: k-CNF and k-uniform hypergraph 2-coloring. Interesting in their own right, these problems seem to capture the essence of the LLL in a simple way. Further, algorithms for these problems usually also lead to algorithms for more general settings to which LLL applies. For k-CNF, the LLL implies that k-CNF formulas in which each clause intersects at most $2^k/e$ other clauses, always have a satisfying assignment. The objective is to find such a satisfying assignment efficiently.

Recently Moser [12] made the breakthrough by discovering an efficient randomized algorithm for finding a satisfying assignment for k-CNF formulas in which each clause intersects at most $2^k/32 - 1$ other clauses. This is best possible up to a constant factor. Subsequently, Moser and Tardos [13], building upon the work of Moser [12], gave an efficient randomized algorithm that works for the general version of LLL under a very general algorithmic framework (discussed below). They also give a randomized parallel algorithm under slightly stronger assumptions. They derandomize their sequential algorithm under the strong condition that *the maximum degree in the dependency graph is a constant*. We address the open question posed by them about derandomizing their algorithm when the degrees of the dependency graph are unbounded.

For k-CNF, the running time of the deterministic algorithms of Moser [12] and of Moser–Tardos [13] behaves like $m^{\Omega(k^2)}$. In this paper, we extend the work of Moser and of Moser–Tardos in several ways. We give a deterministic algorithm with running time $\tilde{O}(m^{2(1+\frac{1}{\epsilon})})$, for k-CNF formulas with m clauses such that no clause shares variables with more than $2^{k/(1+\epsilon)}/e$ other clauses, where ϵ is any positive constant. We obtain this algorithm as a corollary to our deterministic algorithm which works for a general version of LLL in the algorithmic framework of Moser and Tardos [13]. We also give deterministic parallel algorithms in this framework as long as the events satisfy appropriate complexity assumptions (which seem essential to have a handle on the events).

1.1   Algorithmic Framework

To get an algorithmic handle on the LLL, some restrictions on the probability space under consideration are imposed. In this paper we follow the general algorithmic framework for

the local lemma due to Moser–Tardos [13]: Every event A in a finite collection of events $\mathcal{A} = \{A_1, \cdots, A_m\}$ is determined by a subset $S \subseteq \mathcal{P}$ of a finite collection of mutually independent discrete random variables $\mathcal{P} = \{P_1, \cdots, P_n\}$; let D_i be the domain of P_i. We denote by $\mathrm{vbl}(A)$ the minimal subset $S \subseteq \mathcal{P}$ that determines A. We define the *dependency graph* $G = G_\mathcal{A}$ for $\mathcal{A}$ to be the graph on vertex set $\mathcal{A}$ with an edge between events $A, B \in \mathcal{A}$ if $A \neq B$ but $\mathrm{vbl}(A) \cap \mathrm{vbl}(B) \neq \emptyset$. For $A \in \mathcal{A}$ we write $\Gamma(A) = \Gamma_\mathcal{A}(A)$ for the neighborhood of A in G and $\Gamma^+(A) = \Gamma(A) \cup \{A\}$. Note that events that do not share variables are independent.

We think of $\mathcal{A}$ as a family of "bad" events, and the objective is to find a point in the probability space, or equivalently an evaluation of the random variables, where none of the bad events happen. We call such an evaluation a **good evaluation**. Moser and Tardos [13] gave a constructive proof of the general version of the local lemma (Theorem 1) in this framework which seems sufficiently general to capture most applications of LLL, using Algorithm 1.

Algorithm 1: Sequential Moser–Tardos Algorithm

1. For every $P \in \mathcal{P}$, $v_P \leftarrow$ a random evaluation of P.

2. While $\exists A \in \mathcal{A} : A$ happens when $(P = v_P : \forall P \in \mathcal{P})$, do

   (a) Pick one such A that happens.
   (b) Resample(A): For all $P \in \mathrm{vbl}(A)$, do
      - $v_P \leftarrow$ a new random evaluation of P;

3. Return $(v_P)_{P \in \mathcal{P}}$

Theorem 1. *[13] If there exists an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that*

$$\forall A \in \mathcal{A} : \Pr(A) \leq x'(A) := x(A) \prod_{B \in \Gamma(A)} (1 - x(B)), \tag{1}$$

then the probability that none of the events in $\mathcal{A}$ happens is at least $\prod_{A \in \mathcal{A}} (1 - x(A)) > 0$. The expected number of resamplings done by Algorithm 1 before it finds a good evaluation, is $O(\sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)})$.

We work under the framework described in this subsection throughout the paper.

1.2   Our results

To state our results we define the following parameters for $x(A)$ and $x'(A)$ as in Theorem 1 whenever such an assignment exists.

- $D := \max_{P \in \mathcal{P}} \{\mathrm{domain}(P)\}$.

- $M := \max\left\{ n, 4m, 4 \sum_{A \in \overline{\mathcal{A}}} \frac{2|\mathrm{vbl}(A)|}{x'(A)} \cdot \frac{x(A)}{1 - x(A)} \right\}$ where $\overline{\mathcal{A}} = \{A \in \mathcal{A} \mid x'(A) \geq \frac{1}{4m}\}$.

- $w_{min} := \min_{A \in \mathcal{A}} \{-\log x'(A)\}$. Note that $w_{min} \geq 1/M$.

2

These definitions are used throughout the rest of this paper.

To obtain the sequential deterministic algorithm for the general version, we need certain complexity assumptions about the events. The complexity assumptions regarding the conditional probabilities in the following theorem are same as the ones used by Moser-Tardos [13].

Theorem 2. *Let the time needed to compute the conditional probability* $\Pr[A|\forall i \in I : P_i = v_i]$ *for any* $A \in \mathcal{A}$ *and any partial evaluation* $(v_i \in D_i)_{i \in I}$ *where* $I \subseteq [n]$ *be at most* t_C. *If there is an* $\epsilon \in (0,1)$ *and an assignment of reals* $x : \mathcal{A} \to (0,1)$ *such that:*

$$\forall A \in \mathcal{A} : \mathsf{Pr}\,(A) \leq x'(A)^{1+\epsilon} = \left(x(A) \prod_{B \in \Gamma(A)} (1 - x(B)) \right)^{1+\epsilon}.$$

Then, a deterministic algorithm can find a good evaluation in time

$$O\left(t_C \cdot \frac{DM^{2(1+1/\epsilon)} \log M}{\epsilon w_{min}} \right) = \tilde{O}\left(t_C \cdot D \cdot M^{3+2/\epsilon} \right).$$

It is illuminating to look at the special case of k-CNF both in the statement of our theorems and proofs as many of the technicalities disappear, while retaining the basic ideas. For this reason, we will state our results also for k-CNF.

Corollary 3. *Let* F *be a* k-*CNF formula with* m *clauses such that each clause in* F *shares variables with at most* $2^{\frac{k}{1+\epsilon}}/e$ *other clauses for some* $\epsilon \in (0,1)$. *Then there is a deterministic algorithm that finds a satisfying assignment for* F *in time* $\tilde{O}(m^{2(1+\frac{1}{\epsilon})})$.

This improves upon the deterministic algorithms of Moser [12] and of Moser–Tardos (specialized to k-CNF) [13] with running time $m^{\Omega(k^2)}$, which is superpolynomial for $k = \omega(1)$. Our algorithm works in polynomial time under a far wider range of parameters than previous algorithms. To the best of our knowledge, the previous best deterministic algorithm was for hypergraph 2-coloring: This is the problem of coloring the vertices of a hypergraph with two colors so that no hyperedge is monochromatic. A polynomial time deterministic algorithm for k-uniform hypergraphs in which no edge intersects more than $2^{k/16-2}$ other edges appears in [9]. Our theorem above, applies equally well to the 2-coloring problem to state that a deterministic algorithm running in time $\tilde{O}(m^{2(1+\frac{1}{\epsilon})})$ can give a 2-coloring of a k-uniform hypergraph with m edges in which no edge intersects more than $2^{\frac{k}{1+\epsilon}}/e$ other edges. Thus, while the previous algorithms for hypergraph 2-coloring run efficiently only in the case when $\epsilon > 15$, our algorithm is efficient for any $\epsilon > 0$.

For the general setting, the deterministic parallel algorithm works under a fairly general complexity assumption regarding the events, which is the case for many applications; for example, our algorithm applies to k-CNF and hypergraph 2-coloring.

Theorem 4. *Let the time needed to check if an event* $A \in \mathcal{A}$ *happens using* $M^{O(1)}$ *processors be at most* t_{eval}. *Suppose, there is an* $\epsilon \in (0,1)$ *and an assignment of reals* $x : \mathcal{A} \to (0,1)$ *such that:*

$$\forall A \in \mathcal{A} : \mathsf{Pr}\,(A) \leq x'(A)^{1+\epsilon} = \left(x(A) \prod_{B \in \Gamma(A)} (1 - x(B)) \right)^{1+\epsilon}.$$

If there is a constant c such that every event $A \in \mathcal{A}$ has decision tree complexity[1] at most $c \min\{-\log x'(A), \log M\}$. Then there is a parallel algorithm that finds a good evaluation in time

$$O\left(\frac{\log M}{\epsilon w_{\min}}(t_{MIS} + t_{eval})\right)$$

using $M^{O(\frac{c}{\epsilon}\log D)}$ processors, where t_{MIS} is the minimum time to compute the maximal independent set in a m-vertex graph using $M^{O(1)}$ parallel processors on a EREW PRAM.

Restating our theorem again for k-CNF, we get the following corollary.

Corollary 5. *Let F be a k-CNF formula with m clauses such that each clause in F shares variables with at most $2^{\frac{k}{1+\epsilon}}/e$ other clauses for some $\epsilon \in (0,1)$. Then there is a deterministic parallel algorithm that finds a satisfying assignment for F in time $O(\frac{1}{\epsilon}\log^3 m)$ using $m^{O(1/\epsilon)}$ processors on a EREW PRAM.*

Organization. In the next section we give an intuitive description of the new ideas in the paper. We will define the partial witness structure formally in Section 3 and give the sequential deterministic algorithm and the proof that it works efficiently in Section 4. Finally, we present the parallel algorithm and its running time analysis in Section 5. We end with a brief discussion on Lopsided Local Lemma and further work. All missing proofs are given in the appendix.

2 Techniques

In this section, we informally describe the main ideas in the paper at a high level for the special case of k-CNF and indicate how they generalize; reading this section is not essential but provides intuitive aid for understanding the proofs in this paper. For the sake of exposition in this section, we omit numerical constants in some mathematical expressions. Familiarity with the Moser–Tardos paper [13] is very useful but not necessary for this section.

Let F be a k-CNF formula with m clauses. We note immediately that if $k > \log m$, then the probability that a random assignment does not satisfy a clause is $1/2^k \leq 1/(2m)$. Thus the probability that on a random assignment, F has an unsatisfied clause is $\leq 1/2$, and so it can be easily found in polynomial time using the method of conditional probability (see, e.g., [9]). So, from now on we assume that $k \leq \log m$. We also assume that each clause in F intersects with at most $2^k/e$ other clauses; thus LLL guarantees the existence of a satisfying assignment.

To explain our techniques we first need to outline the deterministic algorithm of Moser and of Moser–Tardos which work in polynomial time, though only for $k = O(1)$. Consider a table T of values of the random variables: for each variable in $\mathcal{P}$ the table has a list of values for that variable. Now, we can run Algorithm 1 described above using T: instead of randomly sampling afresh each time the value of a random variable is needed, we pick its next unused value from T. The fact that the randomized algorithm terminates quickly in expectation (Theorem 1), implies that there are small tables on which the algorithm terminates with a satisfying assignment. The deterministic algorithm finds one such table.

---

[1]Informally, a function f having decision tree complexity at most k means that we can determine its evaluation $f(x)$ by adaptively querying at most k components of x.

The constraints such a table has to satisfy can be described in terms of *witness trees*: For a run of the randomized algorithm, whenever an event is resampled, a witness tree "records" the sequence of resamplings that lead to the current resampling. We say that a witness (we will often just say "witness" instead of "witness tree") is *consistent* with a table, if this witness arises when the table is used to run the randomized algorithm. If the algorithm runs for a long time, then it has a large consistent witness certifying this fact. Thus if we use a table which has no large consistent witness, the algorithm terminates quickly.

The deterministic algorithms of Moser and of Moser–Tardos compute a list L of witness trees satisfying several properties: (1) If no tree in L is consistent with a table, then there is no large witness tree consistent with the table. (2) The expected number of trees in L consistent with a random table is less than 1. We need this property to apply the method of conditional probabilities. (3) For the method of conditional probabilities to be efficient, we need the list L to be polynomial in size.

We now motivate how these properties arise naturally while using Algorithm 1. In the context of k-CNF formulas with m clauses satisfying the degree bound, Moser (and also Moser–Tardos when their general algorithm is interpreted for k-CNF) prove two lemmas which they use for derandomization. The *expectation lemma* states that the expected number of large consistent witness trees of size at least $\log m$ is less than $1/2$ (here randomness is over the choice of the table). We could try to use the method of conditional probability to find a table such that there are no large witness trees consistent with it. However there are infinitely many of them. This difficulty is resolved by the *range lemma* which states that if for some u, no witness tree with size in the range $[u, ku]$ is consistent with a table, then no witness tree of size at least u is consistent with the table. Now we can find the required table by the method of conditional probability to exclude all tables with a consistent witness of size in the range $[\log m, k \log m]$. The number of witnesses of size in this range is $m^{\Omega(k^2)}$. To run the method of conditional probability we need to maintain a list of all these witnesses, and find the values in the table so that none of the witnesses in the list remain consistent with it. So, the algorithm of Moser (and respectively Moser–Tardos) works in polynomial time only for constant k.

One possible approach to resolve the issue of large sized list would be to find a way to implicitly maintain the list and carry out the method of conditional probability efficiently even though the list size is large. We have not succeeded in this. However we are able to reduce the size of this list using two new ingredients:

Partial Witness Trees. For a run of the Moser–Tardos randomized algorithm, for each time instant of resampling of an event, we get one witness tree consistent with the input table. Given one consistent witness tree of size $ku + 1$ removing the root gives rise to up to k new consistent witnesses, whose union is the original witness minus the root. Clearly one of these new subtrees has size at least u. This proves their range lemma. The range lemma is optimal for the witness trees: for a given u it is not possible to reduce the multiplicative factor of k between the two endpoints of the range $[u, ku]$.

We overcome this limitation by introducing *partial witness trees*, which have similar properties as the witness trees, but have the additional advantage of allowing a tighter range lemma. The only difference between witness trees and partial witness trees is that the root instead of being labeled by a clause C (as is the case for witness trees), is labeled by a *subset* of variables from C. Now, instead of removing the root to construct new witness trees as in the proof of the Moser–Tardos range lemma, each subset of the set labeling the root gives a

new consistent partial witness tree. This flexibility allows us to prove the range lemma for the tighter range $[u, 2u]$. The number of partial witness trees is larger than the number of witness trees because there are $2^k m$ choices for the label of the root (as opposed to m choices in the case of witnesses) since the root may be labeled by any subset of variables in a clause. But $2^k \leq m$ since as explained at the beginning of this section, we may assume without loss of generality that $k \leq \log m$. So the number of partial witnesses is not much larger and the expectation lemma holds with similar parameters for partial witnesses as well. The method of conditional probability now needs to handle partial witness trees with size in the range $[\log m, 2 \log m]$. The number of partial witnesses in this range is $m^{\Omega(k)}$, which is still too large. The next ingredient brings this number down to a manageable size.

ϵ-**slack.** By introducing ϵ-slack, that is to say, by making the stronger assumption that each clause intersects at most $2^{(1-\epsilon)k}/e$ other clauses, we can prove a stronger expectation lemma: The expected number of partial witnesses of size more than $(\log m)/(\epsilon k)$ is less than $1/2$. We use the fact that the number of labeled trees of size u and degree at most d is less than $(ed)^u \leq 2^{(1-\epsilon)ku}$. Thus number of partial witnesses of size u is less than $2^k m 2^{(1-\epsilon)ku}$, where the factor $2^k m \leq m^2$ accounts for the number of possible labels for the root. Moreover, the probability that a given partial witness of size u is consistent with a random table is $2^{-k(u-1)}$ as opposed to 2^{-ku} in the case of a witness tree; this is proved in a similar manner as for witness trees. Thus the expected number of partial witnesses of size at least $\gamma = O\left(\dfrac{\log m}{\epsilon k}\right)$ consistent with a random table is

$$\leq \sum_{u \geq \gamma} m^2 2^{(1-\epsilon)ku} \cdot 2^{-k(u-1)} \leq \sum_{u \geq \gamma} m^3 2^{-\epsilon ku} \leq 1/2$$

Now, by the new expectation and range lemmas we can reduce the size range to $[(\log m)/\epsilon k, 2(\log m)/\epsilon k]$. The number of partial witnesses in this range is polynomial in m; thus the list of trees that the method of conditional probability needs to maintain has polynomial size.

General Version. For the general version, the root of a partial witness, instead of being labeled by a subset of the variables of an event, is labeled by a subset of the neighbors of that event, and all children of the root must have labels from this subset. This leads to a large number of possible labels for the root. However, for the range lemma to hold true it suffices if we restrict the root of a partial witness to have as label, only a small number of certain chosen subsets of the neighbors. This choice of subset labels is given by a binary tree $\mathbb{B}_A$ for event A.

Another ingredient of the algorithm in the general setting is the notion of *weight of a witness tree*. The weight of a tree is the sum of the weights of individual vertices; more weight is given to those vertices whose corresponding bad events have smaller probability of occurrence.

Our deterministic algorithm for the general version also finds a list L satisfying the three properties mentioned earlier; however as opposed to size, the weight becomes the parameter of concern: Our list L consists of partial witnesses with weight (as opposed to size) in the range $[\gamma, 2\gamma]$, where γ is a number depending on the problem. If we define L using a range for the sizes instead of weights, no list with the properties mentioned above may exist.

Parallel Algorithm. For the parallel algorithm, we use the technique of k-wise δ-dependent probability spaces due to Naor–Naor[14] and its extensions in the later work [6, 3]. This tool was also applied by Alon [1] to give deterministic parallel algorithm for k-uniform hypergraph 2-coloring and other applications of the LLL, but with much worse parameters than ours. Our application of this tool is quite different from the way in which Alon uses this: Alon starts with a random 2-coloring of the hypergraph chosen from a small size limited independence space; at least one of the sample points in this space has the property (roughly speaking) that the almost monochromatic hyperedges form small connected components. For such an assignment, one can alter it locally over vertices in each component to get a good 2-coloring.

However, we do the following: recall that for a random table the expected number of consistent partial witness with size in the range $[(\log m)/(\epsilon k), 2(\log m)/(\epsilon k)]$ is at most $1/2$. Each of these partial witness uses at most $\frac{2 \log m}{\epsilon k} \cdot k$ variables. Now, instead of using a completely random table, we use a table chosen according to a $\frac{2 \log m}{\epsilon}$-wise independent distribution (i.e., any subset of at most $\frac{2 \log m}{\epsilon}$ has the same joint distribution as in the original random table). So the partial witness tree is consistent with the new random table with the same probability as before. And hence the expected number of consistent partial witness trees with the new random table is still at most $1/2$. But now the key point to note is that the number of tables in the new limited independence distribution is much smaller and we can try each of them in parallel till we succeed with one of the tables. To make the probability space even smaller we use k-wise δ-dependent distributions, but the idea remains the same. Finally, to determine whether a table has no consistent heavy partial witness we run the parallel algorithm of Moser–Tardos on it.

The above strategy requires that the number of variables on which witnesses depend be small, and hence the number of variables on which events depend should also be small. In our general parallel algorithm we relax this to some extent: instead of requiring that each event depend on few variables, we require that the decision tree complexity of the event is small. The proof goes through in the same way as above.

3 The Partial Witness Structure

In this section we define partial witness structure and prove a range lemma using weights.

For every $A \in \mathcal{A}$ we fix a rooted **binary tree** $\mathbb{B}_A$. All vertices in $\mathbb{B}_A$ have labels which are nonempty subsets of vbl(A): The root of $\mathbb{B}_A$ is labeled by vbl(A) itself, the leaves are labeled by singleton subsets of vbl(A) and every non-leaf vertex in $\mathbb{B}_A$ is labeled by the disjoint union of the labels of its two children. This means that every non-root non-leaf vertex is labeled by a set $\{v_{i_1}, \cdots, v_{i_k}\}$, $k \geq 2$ while its children are labeled by $\{v_{i_1}, \cdots, v_{i_j}\}$ and $\{v_{i_{j+1}}, \cdots, v_{i_k}\}$ for some $1 \leq j \leq k - 1$. Note that $\mathbb{B}_A$ consists of at most $2|\text{vbl}(A)| - 1$ vertices. We abuse the notation $\mathbb{B}_A$ to denote the labels of the vertices of this binary tree which are actually subsets of vbl(A). This tree is not to be confused with the witness tree. The elements from $\mathbb{B}_A$ will be used to define the roots of the partial witness trees.

Remark 1: (Regarding definition of the binary tree $\mathbb{B}_A$) For the proof of the range lemma it is sufficient to assume that the leaf-subsets of $\mathbb{B}_A$ induce cliques in the dependency graph. This is true when the leafs are singleton subsets. Using a minimum clique cover of the neighborhood as a leaf-set can significantly reduce the size of $\mathbb{B}_A$. We did not state this in Section 1.2 for the sake of simplicity, but the smaller $|\mathbb{B}_A|$ can replace the $2|\text{vbl}(A)|$

factor in the definition of M, reducing the running time bounds of our algorithms. This is for example necessary to obtain the running time bounds stated in Corollary 3 for k-CNF. In the case of k-CNF the neighborhood $\text{vbl}(A)$ of a clause A is naturally covered by at most k cliques, which arise from all the clauses that share a common variable $p \in \text{vbl}(A)$. Thus using a minimum clique cover gives $|\mathbb{B}_A| < 2k$ for k-CNF, saving a factor of $O(m)$ in the running time.

A **partial witness tree** τ_S is a finite rooted tree whose vertices apart from the root are labeled by events while the root is labeled by some $S \in \mathbb{B}_A$ for some $A \in \mathcal{A}$. The children of the root receive labels from the set of events which depend on any of the variables in S; the children of every other vertex labeled by B, receive labels from $\Gamma^+(B)$. For notational convenience, we use $\overline{V}(\tau_S) := V(\tau_S) \setminus \{\text{Root}(\tau_S)\}$ and denote the label of a vertex $v \in \overline{V}(\tau_S)$ by $[v]$.

A **full witness tree** is a special case of a partial witness where the root is the complete set $\text{vbl}(A)$ for some $A \in \mathcal{A}$. In such a case, we relable the root with A instead of $\text{vbl}(A)$. Note that this definition of full witness tree matches the one of witness trees in [13].

Define the weight of an event $A \in \mathcal{A}$ to be $w(A) = -\log x'(A)$. Define the **weight of a partial witness tree** τ_S as the sum of the weights of the labels of the vertices in $\overline{V}(\tau_S)$, i.e.,

$$w(\tau_S) := \sum_{v \in \overline{V}(\tau_S)} w([v]) = -\log \prod_{v \in \overline{V}(\tau_S)} x'([v]).$$

The **depth** of a vertex in a witness tree is the distance of that vertex from the root in the witness tree. We say that a partial witness tree is **proper** if all children of a vertex have distinct labels.

Similar to [11], we will control the randomness using a **table** of evaluations. Let us denote this table by T. This table contains a row for each variable. The row for each variable contains evaluations for the variable. Note that the number of columns in the table could possibly be infinite. In order to use such a table in the algorithm, we maintain a pointer t_p for each variable $p \in \mathcal{P}$ indicating the column containing its current value used in the evaluation of the events. We denote the value of p at t_p by $T(p, t_p)$. If we want to resample for a variable, we just increment this pointer by one.

It is clear that running the randomized algorithm is equivalent to picking a table with random values and using such a table to run the algorithm. We call a table T a **random table** if, for all variables $p \in \mathcal{P}$ and all positions j, the entry $T(p, j)$ is picked independently randomly according to the distribution of p.

Algorithm 2: Moser-Tardos Algorithm with input table
Input: Table T with values for variables.
Output: An assignment of values for variables which makes none of the events in $\mathcal{A}$ happen.

1. For every variable $p \in \mathcal{P}$: Initiate the pointer $t_p = 1$.

2. While $\exists A \in \mathcal{A}$ that happens when $(\forall P \in \mathcal{P} : p = T(p, t_p)$, do:

    (a) Pick one such A.

    (b) Resample(A): For all $P \in \text{vbl}(A)$ increment t_p by one

3. Return $p = T(p, t_p) : \forall p \in \mathcal{P}$

In the above algorithm, Step 2(a) is performed by a fixed arbitrary deterministic procedure. This makes the algorithm well-defined.

Let $C : \mathbb{N} \to \mathcal{A}$ be an ordering of the events, which we call the **event-log**. Let the ordering of the events as they have been selected for resampling in the execution of Algorithm 2 using a table T be denoted by an event-log C_T. Note that C is partial if the algorithm terminates in finite time.

Given event-log C, associate with each step t and each $S \in \mathbb{B}_{C(t)}$, a partial witness tree $\tau_C(t, S)$ as follows. Define $\tau_C^{(t)}(t, S)$ to be an isolated root vertex labeled S. Going backwards through the event-log, for each $i = t - 1, t - 2, \cdots, 1$: (i) if there is a non-root vertex $v \in \tau_C^{(i+1)}(t, S)$ such that $C(i) \in \Gamma^+([v])$, then choose among all such vertices the one having the maximum distance from the root (break ties arbitrarily) and attach a new child vertex u to v with label $C(i)$, thereby obtaining the tree $\tau_C^{(i)}(t, S)$, (ii) else if $S \cap \text{vbl}(C(i))$ is non-empty, then attach a new child vertex to the root with label $C(i)$ to obtain $\tau_C^{(i)}(t, S)$, (iii) else, set $\tau_C^{(i)}(t, S) = \tau_C^{(i+1)}(t, S)$.

Note that the witness tree $\tau_C(t, S)$ is partial unless $S = \Gamma^+(C(t))$, in which case the witness tree is identified with a full witness tree. For such a full witness tree, our construction matches the construction of *witness trees associated with the log* in [13].

We say that the partial witness tree τ_S **occurs** in event-log C if there exists $t \in \mathbb{N}$ such that for some $A \in \mathcal{A}$, $C(t) = A$ and $\tau_S = \tau_C(t, S)$ for some $S \in \mathbb{B}_A$.

For a table T, a T-**check** on a partial witness tree τ_S uses table T as follows: In an order of decreasing depth, visit the non-root vertices of τ_S and for a vertex with label A, take the first unused value from T and check if the resulting evaluation makes A happen. The T-check passes if all events corresponding to vertices apart from the root, happen when checked. We say that a partial witness tree is **consistent with a table** T if T-check passes on the partial witness tree.

Lemma 6. *If a partial witness tree τ_S occurs in the event-log C_T, then*

1. *τ_S is proper.*

2. *τ_S passes the T-check.*

Proof. The proof of this lemma is similar to the one in [13].

Since τ_S occurs in C_T, there exists some time instant t such that for $S \in \mathbb{B}_{C_T(t)}$, $\tau_S = \tau_{C_T}(t, S)$. Let $d(v)$ denote the depth of vertex $v \in \overline{V}(\tau_S)$ and let $q(v)$ denote the time instant of the algorithm constructing $\tau_{C_T}(t, S)$ in which v was attached, that is, $q(v)$ is the largest value q with v contained in $\tau_{C_T}^{(q)}(t)$.

If $q(u) < q(v)$ for vertices $u, v \in \overline{V}(\tau_S)$ and $\text{vbl}([u])$ and $\text{vbl}([v])$ are not disjoint, then $d(u) > d(v)$. Indeed, when adding the vertex u to $\tau_{C_T}^{(q(u)+1)}(t)$ we attach it to v or to another vertex of equal or greater depth. Therefore, for any two vertices $u, v \in \overline{V}(\tau_S)$ at the same depth $d(v) = d(u)$, $[u]$ and $[v]$ do not depend on any common variables, that is the labels in every level of τ_S form an independent set in G. In particular τ_S must be proper.

Now consider a vertex v in the partial witness tree τ_S. Let $S(p)$ be the set of vertices $w \in \tau_S$ with depth greater than that of v such that $[w]$ depends on variable p.

When T-check considers the vertex v labeled by B and uses the next unused evaluation of the variable P, it uses the evaluation $T(p, |S(p)|)$. This is because the witness check visits the vertices in order of decreasing depth and among the vertices with depth equal to that

of v, only $[v]$ depends on p and so before the witness-check considers v it must have used values for p exactly when it was considering the vertices in $S(p)$.

At the time instant of resampling $[v]$, say t_v, the algorithm chooses $[v]$ to be resampled which implies that $[v]$ happens before this resampling. For $p \in \text{vbl}([v])$, the value of the variable p at t_v is $T(p, |S(p)|)$. This is because, the pointer for P was increased for events $[w]$ which were resampled before the current instance, where $w \in S(p)$. Note that every event which was resampled before t_v and which depend on $[v]$ would be present at depth greater than that of v in τ_S by construction. Hence, $S(p)$ is the complete set of events which led to resampling of p before the instant t_v.

As the T-check uses the same values for the variables in $\text{vbl}([v])$ when considering v, it also must find that $[v]$ happens. $\qquad\square$

Next, we state the important lemma used for derandomizing efficiently.

Lemma 7. *If a partial witness tree of weight at least γ occurs in the event-log C_T such that $x'([v]) \geq 1/4m$ for every non-root vertex v, then a partial witness of weight $\in [\gamma, 2\gamma]$ occurs in the event-log C_T.*

Proof. Suppose not. Then, consider the least weight partial witness tree whose weight is at least γ which occurs in the event-log C_T, namely $\tau_S = \tau_{C_T}(t, S)$ for some t, $S \in \mathbb{B}_A$. By assumption, $w(\tau_S) \geq 2\gamma$. We have two cases:

Case (i): $\text{Root}(\tau_S)$ has only one child. Let this child be labeled by v. Let t' be the largest instant before t at which $[v]$ was resampled. Now, consider the partial witness tree $\tau'_S = \tau_{C_T}(t', S') = \Gamma^+([v])$ (it is a partial witness tree since $S' \in \mathbb{B}_{[v]}$). Since τ'_S contains one less vertex than τ_S, $w(\tau'_S) < w(\tau_S)$. Also, since $x'(A) \geq 1/4m$, we have $\log(1/x'(A)) < \log(4m) < \frac{\log M}{\epsilon} = \gamma$ (by the parameters defined in section 1.2) we have that $w(\tau'_S) = w(\tau_S) - \log(1/x'(A)) \geq \gamma$. Finally, by definition of τ'_S, it is clear that τ'_S occurs in the event-log C_T. Thus, τ'_S is a counterexample of smaller weight contradicting our choice of τ_S.

Case (ii): $\text{Root}(\tau_S)$ has at least two children. This means that $S = \{A_{i_1}, \cdots, A_{i_k}\}$ for some $k \geq 2$. In $\mathbb{B}_A$, starting from S, we now explore the children of S in the following way, looking for the first vertex whose children S_L and S_R reduce the weight of the tree, i.e., $0 < w(\tau_{S_L}), w(\tau_{S_R}) < w(\tau_S)$, where $\tau_{S_L} = \tau_{C_T}(t, S_L)$ and $\tau_{S_R} = \tau_{C_T}(t, S_R)$: if a vertex S_L reduces the weight of the tree without making it zero (i.e., $0 < w(\tau_{S_L}) < w(\tau_S)$), then its variable disjoint sibling S_R must also reduce the weight of the tree; on the other hand if a vertex S_L reduces the weight of the tree to zero, then its sibling S_R can not reduce the weight of the tree. Suppose S_L reduces the weight to zero, then we explore S_R to check if its children reduce the weight. Lastly the weight has to reduce while considering a vertex which is a parent of leaves.

By definition, both τ_{S_L} and τ_{S_R} occur in the event-log C_T. Since we pick the first siblings S_L and S_R (in the breadth first search) which reduce the weight, their parent S' is such that $w(\tau_{S'}) \geq w(\tau_S)$, where $\tau_{S'} = \tau_{C_T}(t, S')$. We are considering only those S' such that $S' \subseteq S$ which implies that $w(\tau_{S'}) \leq w(\tau_S)$. Hence, $w(\tau_{S'}) = w(\tau_S)$ and for every vertex with label A in τ_S, one can find a unique vertex labeled by A in $\tau_{S'}$ and vice-versa. Further, S' is the disjoint union of S_L and S_R; therefore, for vertex with label A in $\tau_{S'}$, one can find a unique vertex labeled by A either in τ_{S_L} or τ_{S_R}.

As a consequence, we have that for every vertex with label A in τ_S, one can find a unique vertex labeled by A either in τ_{S_L} or τ_{S_R}. Hence, $w(\tau_{S_L}) + w(\tau_{S_R}) \geq w(\tau_S)$ and therefore, $\max\{w(\tau_{S_L}), w(\tau_{S_R})\} \geq w(\tau_S)/2 \geq \gamma$. So, if we consider the witness among τ_{S_L} and τ_{S_R} with

larger weight, it is a counterexample of weight at least γ but of weight less than that of τ_S, contradicting our choice of τ_S. $\qquad\square$

4   Deterministic Algorithm

In this section we describe a sequential deterministic algorithm and prove Theorem 2.

For the rest of the paper we define a set of **forbidden witnesses** F which contains all full witnesses consisting of single events of large weight and all partial witness trees of weight between γ and 2γ consisting only of small weighted events. More formally, $F := F_1 \cup F_2$ where

- $F_1 := \{\tau = v \mid [v] \in \mathcal{A} \setminus \overline{\mathcal{A}}\}$ and

- $F_2 := \{\tau \mid w(\tau) \in [\gamma, 2\gamma] \wedge \forall v \in \tau, [v] \in \overline{\mathcal{A}} \wedge (Root(\tau) \in \mathbb{B}_A \text{ for some } A \in \overline{\mathcal{A}} \}$.

Recall that $\overline{\mathcal{A}} = \left\{ A \in \mathcal{A} \mid x'(A) \geq \frac{1}{4m} \right\}$. With these definitions we can state our deterministic algorithm.

Algorithm 3: Sequential Deterministic Algorithm

1. Enumerate all forbidden witnesses in F.

2. Construct table T by the method of conditional probability: For each variable $p \in \mathcal{P}$, and for each j, $0 \leq j \leq \frac{2\gamma}{w_{min}}$, do

   - Select a value for $T(p, j)$ that minimizes the expected number of forbidden witnesses that can occur in the event-log given that all values chosen so far and $T(p, j)$ are fixed and the yet uncomputed values are random.

3. Run Algorithm 2 using table T as input.

For the Algorithm 3 to work efficiently, we need that the table T can be constructed by the method of conditional probability quickly. For this, we prove that the number of forbidden witnesses is polynomial using Lemma 9. Further, we prove that the table given by the method of conditional probability is a good table; that is, the execution of the sequential algorithm using this table terminates within polynomial number of steps. We show this as follows: Lemma 8 proves that the number of forbidden witnesses that occur in the event-log by using this table is zero (due to the invariant maintained by the method of conditional probability while obtaining the table). Therefore, the sequential algorithm does not encounter any of the forbidden witnesses. Putting this together with Lemma 7, we can show that the maximum weight of any partial witness tree occurring in the event-log produced by the execution of the sequential algorithm with any fixed order to resample is at most γ. Finally, the maximum number of vertices in a partial witness tree of weight at most γ is not too large. Hence, the maximum number of times any event is resampled while running the sequential algorithm using table T is not too large.

Lemma 8. *The expected number of forbidden witnesses occurring in the event-log C_T, where T is a random table, is less than $\frac{1}{2}$.*

Proof. We first prove that the expected number of witnesses $\tau \in F_2$ occurring in C_T is at most $\frac{1}{4}$: For each event $A \in \overline{\mathcal{A}}$, let Υ_A and Υ'_A be the set of partial and respectively full witness trees in F_2 with root from $\mathbb{B}_A$. With this notation the expectation in question is exactly:

$$E := \sum_{A \in \overline{\mathcal{A}}} \sum_{\tau \in \Upsilon_A} \mathsf{Pr}\left(\tau \text{ occurs in the event-log } C_T\right)$$

Note that according to Lemma 6, a partial witness tree occurs in the event-log C_T only if it passes the T-check. Clearly, the probability that a witness τ passes the T-check for the random table T is $\prod_{v \in \overline{V}(\tau)} \mathsf{Pr}\left([v]\right)$. Using this and the assumption in Theorem 2 that $\mathsf{Pr}\left([v]\right) \leq x'([v])^{1+\epsilon}$ we get

$$E \leq \sum_{A \in \overline{\mathcal{A}}} \sum_{\tau \in \Upsilon_A} \prod_{v \in \overline{V}(\tau)} x'([v])^{1+\epsilon}$$

Note that each full witness tree with root $A \in \overline{\mathcal{A}}$ can give rise to at most $|\mathbb{B}_A|$ partial witness trees since the possible subsets that we consider are only the sets which occur as labels of vertices in the binary tree $\mathbb{B}_A$. Hence, we can rewrite to get that the expectation is

$$E \leq \sum_{A \in \overline{\mathcal{A}}} |\mathbb{B}_A| \sum_{\tau \in \Upsilon'_A} \prod_{v \in \overline{V}(\tau)} x'([v])^{1+\epsilon}$$

$$= \sum_{A \in \overline{\mathcal{A}}} |\mathbb{B}_A| \sum_{\tau \in \Upsilon'_A} \left(\prod_{v \in \overline{V}(\tau)} x'([v]) \right) 2^{-\gamma\epsilon},$$

where the last expression follows because, for $\tau \in \Upsilon'_A$, we have:

$$w(\tau) = -\log \prod_{v \in \overline{V}(\tau)} x'([v]) \geq \gamma$$

$$\implies \prod_{v \in \overline{V}(\tau)} x'([v]) \leq 2^{-\gamma}.$$

Since every partial witness tree that occurs in the event-log is proper (Lemma 6), by using the Galton-Watson process, similar to the summation in [13], we get that the expectation is

$$E \leq \sum_{A \in \overline{\mathcal{A}}} \frac{|\mathbb{B}_A|}{x'(A)} \left(\sum_{\tau \in \Upsilon'_A} \prod_{v \in V(\tau)} x'([v]) \right) 2^{-\gamma\epsilon}$$

$$\leq \sum_{A \in \overline{\mathcal{A}}} \frac{|\mathbb{B}_A|}{x'(A)} \cdot \left(\frac{x(A)}{1 - x(A)} \right) 2^{-\gamma\epsilon}$$

$$< \frac{M}{4} 2^{-\gamma\epsilon}.$$

where the last inequality follows since $|\mathbb{B}_A| < 2|\mathrm{vbl}(A)|$ and using $\gamma = \frac{\log M}{\epsilon}$. Hence, the expected number of witnesses $\tau \in F_2$ occurring in C_T is at most $\frac{1}{4}$.

To calculate the expected number of forbidden witnesses in $F = F_1 \cup F_2$ that occur in C_T we use the linearity of expectation to obtain the desired result:

$$\sum_{\tau \in F} \mathsf{Pr}\left(\tau \text{ occurs in } C_T\right) = \sum_{\tau \in F_1} \mathsf{Pr}\left(\tau \text{ occurs in } C_T\right) + \frac{1}{4} < m\frac{1}{4m} + \frac{1}{4} = \frac{1}{2}$$

$\square$

Owing to the definition of forbidden witnesses *via weights*, there is an easy way to count the number of witnesses using the fact that their expected number is small.

Lemma 9. *The number of forbidden witnesses $|F|$ is less than $M^{2(1+1/\epsilon)}$.*

Proof. We first count the number forbidden witnesses in F_2. Each of them have weight $w(\tau) \leq 2\gamma$ and thus:

$$|F_2|(2^{-2\gamma})^{(1+\epsilon)} \leq \sum_{\tau \in F_2} (2^{-w(\tau)})^{(1+\epsilon)} = \sum_{\tau \in F_2} \left(\prod_{v \in \overline{V}(\tau)} x'(B) \right)^{(1+\epsilon)} \leq \frac{1}{4}$$

where the last inequality comes directly from the proof of Lemma 8. Therefore $|F_2| \leq \frac{1}{4}2^{2\gamma(1+\epsilon)} = \frac{1}{4}M^{2(1+1/\epsilon)}$ and the total number of forbidden witnesses is $|F| = |F_1| + |F_2| \leq m + \frac{1}{4}M^{2(1+1/\epsilon)} < M^{2(1+1/\epsilon)}$. $\square$

We are ready to prove Theorem 2.

Proof of Theorem 2. The first step of the deterministic algorithm enumerates all forbidden witnesses in F. By Lemma 9, there are at most $M^{2(1+1/\epsilon)}$ forbidden witnesses and it is easy to enumerate them using a dynamic programming approach.

The running time to find the suitable table T using the method of conditional probabilities can be computed as follows: each variable has at most D possible choices of values; for each value we compute the expected number of forbidden witnesses that can occur in the event-log by incrementally computing the probability of each forbidden witness in F to occur and keeping track of the sum of these probabilities. Further, each forbidden witness $\tau \in F_2$ has weight at most 2γ and thus each forbidden witness consists of at most $k = \left(\frac{2\gamma}{w_{min}} + 1\right) = \frac{2 \log M}{\epsilon \, w_{min}} + 1$ vertices. Hence, the time to compute T is at most

$$O(D \cdot |F| \cdot k \cdot t_C) = O\left(\frac{DM^{2(1+1/\epsilon)} \log M}{\epsilon w_{min}} t_C \right) = \tilde{O}\left(DM^{3 + \frac{2}{\epsilon}} t_C \right)$$

since $w_{min} \geq 1/M$. To complete the proof we show that the running time of the sequential algorithm on a table T obtained by Step 2 of the deterministic algorithm is at most $O\left(\frac{2m \log M}{\epsilon w_{min}} t_C \right)$:

First, note that by running the sequential algorithm using the table T, none of the forbidden witnesses can occur in the event-log C_T. This is because the table is obtained by the method of conditional probabilities: In the beginning of the construction of the table, when no value is fixed, the expected number of forbidden witnesses that occur in the event-log is less than $1/2$ as proved in Lemma 8; this invariant is maintained while picking values

13

for variables in the table; thus, once all values are fixed the number of witness trees in F that occur in the event-log C_T is still less than $1/2$ and hence zero. This implies that none of the events $A \in \mathcal{A} \setminus \overline{\mathcal{A}}$ are resampled while using the table T since these events are labels of forbidden witnesses in F_1. It also guarantees that the sequential algorithm with T as input resamples each event $A \in \mathcal{A}$ at most k times. This is because, if some event $A \in \mathcal{A}$ is resampled more than k times, then, A occurs in the event-log C_T at least k times. Now, the weight of the partial witness tree associated with the last instance at which A was resampled, would weigh at least $kw_{\min}$ which is more than 2γ, a contradiction to Lemma 7. Therefore, the running time for the sequential algorithm using the table T is $O\left(m \cdot k \cdot t_C\right)$ which is smaller than the upper bound for the time needed to find the good table T. $\square$

To obtain the corollary regarding k-CNF, we use the optimum values for the parameters used to obtain the deterministic algorithm for the general version.

Proof of Corollary 3. For a k-CNF formula with clauses $\mathcal{A} = \{A_1, \cdots, A_m\}$, for each clause $A \in \mathcal{A}$ we define an event A and say that the event happens if the clause is unsatisfied. Further, each variable appearing in the formula picks values uniformly at random from $\{0, 1\}$. Then, for every event A, $\Pr(A) = 2^{-k}$. We assume that $k = O(\log m)$ for otherwise the problem becomes simple. If d is the maximum number of clauses that a clause shares its variables with, setting $x(A) = 1/d \; \forall A \in \mathcal{A}$, we obtain that $x'(A) > \frac{1}{de}$. The condition that $d \leq 2^{k/(1+\epsilon)}/e$ can be translated to $\Pr(A) \leq x'(A)^{1+\epsilon} \; \forall A \in \mathcal{A}$. Therefore, we use parameters $t_C = O(k)$, $w_{min} = O(k)$, $D = 2$ and a slightly optimized parameter of M with $\Gamma^+(A)$ replaced by k (see the Remark at the beginning of Appendix Section 7.1), to obtain $M = O(mk) = O(m \log m)$. With these parameters the following corollary follows directly from Theorem 2. $\square$

5 Parallel Algorithm

We need the definition of (k, δ)-approximate distribution to describe our algorithm.

Definition 10. (k, δ)-*approximations[6]: Let $\mathcal{D}$ be the probability distribution on $D_1 \times D_2 \times \ldots \times D_n$ given by the random variables in $\mathcal{P}$. For positive integer k and constant $\delta \in (0, 1)$ a probability distribution $\mathcal{Y}$ on $D_1 \times D_2 \times \ldots \times D_n$ is said to be a (k, δ)-approximation of $\mathcal{D}$ if the following holds. For every $I \subseteq [n]$ such that $|I| \leq k$, and every $v \in D_1 \times D_2 \times \ldots \times D_n$ we have*

$$| \Pr_{\mathcal{D}}[v_I] - \Pr_{\mathcal{Y}}[v_I]| \leq \delta,$$

where $\Pr_{\mathcal{D}}[v_I]$ denotes the probability that for a random vector $(x_1, \ldots, x_n)$ with probability distribution $\mathcal{D}$ we get $x_i = v_i$ for $i \in I$ and $\Pr_{\mathcal{Y}}[v_I]$ is defined similarly.

Algorithm 4: Parallel Deterministic Algorithm

1. Take Y as the domain of a (k, δ)-approximation $\mathcal{Y}$ for the probability space of random tables.

2. For each table $T \in Y$ do in parallel:

   (a) For every variable $p \in \mathcal{P}$: Initiate the pointer $t_p = 1$.

(b) While $\exists A \in \mathcal{A}$ that happens when $(\forall P \in \mathcal{P} : p = T(p, t_p))$, do:

- Compute a maximal independent subset I of those clauses.
- Resample(A): For all $P \in \bigcup_{A \in I} \mathrm{vbl}(A)$ increment t_p by one.

3. Once a valid assignment is found using one of the tables, output it and terminate.

As in the sequential algorithm, the problem reduces to finding a table on which the above algorithm terminates quickly. Our algorithm relies on the following observation: instead of sampling the values in the table independently at random, if we choose it from a distribution that is a (k, δ)-approximation of the original distribution (for appropriate k and δ), the algorithm behaves as if the values in the table had been chosen independently at random (Proposition 11). The support of a (k, δ) distribution is polynomially small, and so this gives us a polynomial sized set of tables which is guaranteed to contain at least one table on which the algorithm terminates quickly (Corollary 12). Our algorithm runs the Moser–Tardos parallel algorithm on each of these tables in parallel, and stops as soon as one of the tables lead to the good evaluation.

Proposition 11. *Let $S = D_1 \times \ldots \times D_n$ be a product space of finite domains of size at most $D = \max_i |D_i|$, $\mathcal{P}$ be an independent product distribution on S and $f, f_1, f_2 : S \to \{0, 1\}$ be boolean functions on S.*

- *If f_1 and f_2 have decision tree complexity k_1 and k_2 respectively, then the decision tree complexity of $f_1 \wedge f_2$ is at most $k_1 + k_2$.*

- *If f has decision tree complexity at most k then every (k, δ)-approximation $\mathcal{Y}$ of $\mathcal{P}$ is $D^k\delta$-indistinguishable from $\mathcal{P}$, i.e.:*

$$|E_\mathcal{Y}(f) - E_\mathcal{P}(f)| \leq D^k\delta$$

Proof. For the first claim we recall that a function having decision tree complexity at most k means exactly that we can determine its evaluation $f(x)$ for $x \in S$ by adaptively querying at at most k components of x. If this is true for f_1 and f_2 with k_1 and k_2 respectively then we can easily evaluate $f_1(x) \wedge f_2(x)$ by adaptively querying at most $k_1 + k_2$ components of x. Therefore this conjunction has decision tree complexity at most $k_1 + k_2$.

For the second claim, we fix a decision tree of f with depth at most k. Each of the leaf-to-root paths in this tree corresponds to a partial assignment of values to at most k components which already determines the result of f. The expectation of f under any distribution is simply the sum of the probabilities of the paths resulting in a 1-evaluation at the leaf. Switching from a completely independent distribution to a k-wise independent distribution does not change these probabilities since the partial assignments involves at most k variables. Similarly switching to a (k, δ)-approximation changes each of these probabilities by at most δ. There are at most D^k paths resulting in a 1-evaluation which implies that the deviation of the expectation is at most $D^k\delta$. $\square$

Remark 2: Instead of assuming for each event A, a decision tree complexity of at most $k(A) = O(\min\{\log M, -\log x'(A)\})$ it suffices to demand a less strong assumption that the boolean function for each event A can be expressed by a polynomial of degree at most $k(A)$ with coefficients and number of terms bounded by $2^{k(A)}$. In general any property that fulfills the above "additivity" and implies the property that any (k, δ)-approximation is $2^{O(k)}\delta$-indistinguishable suffices. We do not know whether this property itself behaves additively.

Corollary 12. *The expected number of forbidden witnesses consistent with a table T that was created by a (k, δ)-approximation for distribution of random tables with $k = 2c\gamma$ and $\delta^{-1} = 3M^{2+2/\epsilon}D^{2c\gamma}$ is at most $1/2 + 1/3 < 1$.*

Proof. The event that a partial witness $\tau \in F_2$ is consistent with T is exactly the conjunction of events $[v]$, $v \in \overline{V}(\tau)$; using Proposition 11, the decision tree complexity of this tree is at most $\sum_{v \in \overline{V}(\tau)} c \min\{\log M, -\log x'([v])\}$. Also, if $\tau \in F_1$, then clearly the decision tree complexity of τ is at most $c \log M \leq c\gamma$; else if $\tau \in F_2$, then

$$\sum_{v \in \overline{V}(\tau)} c \min\{\log M, -\log x'([v])\} \leq c \sum_{v \in \overline{V}(\tau)} -\log x'([v]) \leq 2c\gamma.$$

Lemma 8 shows that using the original independent distribution $\mathcal{P}$, the expected number of forbidden witnesses occurring is at most $1/2$. The second claim of Proposition 11 proves that switching to a (k, δ)-approximation changes this expectation by at most $D^k \delta = \frac{1}{3M^{2+2/\epsilon}}$ for each of the $|F|$ witnesses. To complete the proof, observe that by Lemma 9 we have $|F| \leq M^{2+2/\epsilon}$. $\square$

We obtain the proof of Theorem 4 along the outline mentioned in the beginning of the section.

Proof of Theorem 4. We use Algorithm 4 to obtain the good evaluation. Corollary 12 guarantees that there is a table $T \in Y$ for which there is no forbidden witness consistent with it. Steps 2a–3 are exactly the parallel algorithm of Moser and Tardos [13] and by using Lemma 4.1 of [13], if this algorithm runs for i iterations then there exists a consistent witness of height i. Such a witness has weight at least $i \times w_{\min}$ and thus $i < \frac{\gamma}{w_{\min}}$. Each of these i iterations takes time t_{eval} to evaluate all m clauses and time t_{MIS} to compute the independent set on the induced dependency subgraph of size at most m. This proves that after creating the probability space $\mathcal{Y}$ the algorithm terminates in $O(\frac{\gamma}{w_{\min}}(t_{MIS} + t_{eval}))$ time and the termination criterion guarantees a valid outcome. The number of processors needed for the loop is polynomial bounded by $M^{O(1)}$ for each of the $|Y|$ parallel computations.

It can be shown that Y can be constructed efficiently in parallel and that it is at most polynomially large. For this, we use the construction described in [6] (which in turn uses a construction in [14]). This construction builds a (k, δ)-approximation to a product space with maximum domain size D and $|T|$ variables. Such a space has size $|Y| = \text{poly}(2^k, \log|T|, \delta^{-1}) = M^{O(\frac{\epsilon}{\epsilon}\log D)}$ and can be constructed in parallel in time $\text{poly}(\log k + \log\log|T| + \log\log\delta^{-1}) = O(\log M)$. The construction described in [6] can be parallelized with the required parameters easily. $\square$

6 Discussion

Moser and Tardos [13] raised the open question for a deterministic LLL algorithm. We address this and give a deterministic algorithm that works under nearly the same conditions as its randomized versions. All known deterministic or (randomized) parallel algorithms need a slack in the LLL conditions. Whether our ϵ-slack can be removed remains open. Similar to Moser-Tardos, our sequential algorithm needs the assumption that exact conditional probabilities can be computed efficiently. These assumptions seem to hold for many applications and exact computations of probabilities can be performed reasonably efficiently. It is still

an interesting question how much room is there for approximate computations and whether those assumptions are needed. The complexity assumptions of our parallel algorithm are alternative assumptions which may also be weakened (see Remark 2).

We have an improved parallel algorithm that determines a valid assignment directly from a good table needing only polynomially many non-adaptive evaluations. While this is a surprising result on its own, it also speeds up the running time. Nevertheless one maximal independent set needs to be computed and it is an interesting question whether there is a parallel algorithm that avoids this.

This parallel algorithm is of interest because it extends to the Lopsided LLL, which uses a weaker notion of event-dependency. Moser-Tardos proved that their sequential algorithm extends to this case but their parallel algorithm does not. The same is true for our sequential and parallel algorithm. The improved parallel algorithm overcomes this and is the first parallel algorithm for the lopsided LLL.

References

[1] N. Alon. A Parallel Algorithmic Version of the Local Lemma. *Random Struct. Algorithms*, 2(4):367–378, 1991.

[2] J. Beck. An Algorithmic Approach to the Lovász Local Lemma. *Random Struct. Algorithms*, 2(4):343–366, 1991.

[3] S. Chari, P. Rohatgi, and A. Srinivasan. Improved Algorithms via Approximations of Probability Distributions. *J. Comput. Syst. Sci.*, 61(1):81–107, 2000.

[4] A. Czumaj and C. Scheideler. A new algorithm approach to the general Lovász local lemma with applications to scheduling and satisfiability problems (extended abstract). In *STOC*, pages 38–47, 2000.

[5] P. Erdős and L. Lovász. Problems and results on 3-chormatic hypergraphs and some related questions. In *A. Hajnal, R. Rado and V.T. Sós, editors, Infinite and Finite Sets (Colloq., Keszthely, 1973; dedicated to P. Erdós on his 60th birthday)*, volume 2, pages 609–627, 1975.

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

[7] U. Feige. On allocations that maximize fairness. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 287–293, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[8] F. T. Leighton, B. M. Maggs, and S. Rao. Packet Routing and Job-Shop Scheduling in (Congestion + Dilation) Steps. *Combinatorica*, 14(2):167–186, 1994.

[9] M. Molloy and B. Reed. *Graph Colouring and the Probabilistic Method*. Springer, 2000.

[10] M. Molloy and B. A. Reed. Further Algorithmic Aspects of the Local Lemma. In *STOC*, pages 524–529, 1998.

[11] R. A. Moser. Derandomizing the Lovász Local Lemma more effectively. *CoRR*, abs/0807.2120, 2008.

[12] R. A. Moser. A constructive proof of the Lovász local lemma. In *STOC '09: Proceedings of the 41st annual ACM Symposium on Theory of Computing*, pages 343–350, New York, NY, USA, 2009. ACM.

[13] R. A. Moser and G. Tardos. A constructive proof of the general Lovász Local Lemma. *CoRR*, abs/0903.0544, 2009.

[14] J. Naor and M. Naor. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[15] A. Srinivasan. Improved algorithmic versions of the Lovász Local Lemma. In *SODA*, pages 611–620, 2008.