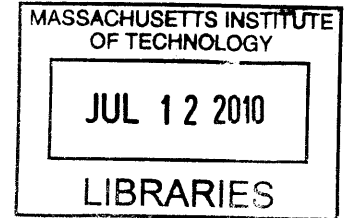# Deterministic Algorithms for the Lovász Local Lemma

by

Bernhard Haeupler

B.Sc. in Applied Mathematics, 2007
Diploma in Mathematics, 2008
M.Sc.Hon. in Applied Mathematics, 2009
Technical University Munich

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARCHIVES

JUNE 2010

Signature of author: _____

Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by: _____

David Karger
Professor of Computer Science
Thesis Supervisor

Accepted by: _____

Terry Orlando
Professor of Electrical Engineering
Chair, Committee for Graduate Students

Deterministic Algorithms for the Lovász Local Lemma

by

Bernhard Haeupler

Submitted to the Department of Electrical Engineering
and Computer Science on May 21, 2010
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

ABSTRACT

The Lovász Local Lemma [6] (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 the formula has a satisfying assignment. Recently, a randomized algorithm to efficiently construct a satisfying assignment was given by Moser [17]. Subsequently Moser and Tardos [18] 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 times $m^{\Omega(k^2)}$ and $m^{\Omega(k \cdot 1/\epsilon)}$ which are 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 the asymmetric version of LLL under the algorithmic framework of Moser and Tardos [18] and is also parallelizable, i.e., has polylogarithmic running time using polynomially many processors.

Thesis Supervisor: David R. Karger
Title: Professor of Computer Science

Bibliography of the Candidate

Since September 2008 I have been a PhD student in the Theory Group of the MIT Computer Science Department. My advisor is David Karger. In 2007-2008 I was a visiting graduate student at the Theory Group of the Princeton's Computer Science Department where I worked with Robert E. Tarjan. From 2004 to 2007 I studied in Germany at the Technical University Munich where in October 2007 I completed the mathematics diploma. From 2006 to 2008 I also studied in the elite graduate program TopMath in Applied Mathematics in which I completed the TopMath Elite Bachelor of Science (2007) and the TopMath Master of Science with Honours (2009).

Acknowledgments

Contents

1 Introduction

The Lovász Local Lemma [6] (henceforth 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 (see Section 2.1 for a more detailed exposition). The 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 [6] is non-constructive in this sense. Starting with the work of Beck [3], a series of papers [1, 5, 15, 20, 16] have sought to make the LLL constructive. We note that in most applications where the LLL is useful (e.g., [9, 11, 14]), the proof of existence of the desired structure is known only through LLL.

Algorithms for the LLL are often formulated for one of two model problems: k-SAT 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 also lead to algorithms for more general settings to which LLL applies. For k-SAT, the LLL implies that every k-CNF formula in which each clause shares variables with at most $2^k/e$ other clauses has a satisfying assignment (see Section 2.2). The objective is to find such a satisfying assignment efficiently.

Recently Moser [17] discovered 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 [18], building upon the work of Moser [17], gave an efficient randomized algorithm that works for the general version of LLL under a very general algorithmic framework (discussed in Section 2.3). They also give a randomized parallel algorithm when the LLL conditions are relaxed by an $(1 - \epsilon)$ factor. They derandomize their sequential algorithm using a similar ϵ-slack and get a running time of $m^{O((1/\epsilon)d \log d)}$where d is the maximum number of clauses sharing variables with some other clause. This running time is polynomial only under the strong condition that the degree of the dependency graph is bounded by a constant. We address the open question posed by them about derandomizing their algorithm when the degrees of the dependency graph are unbounded by giving a deterministic algorithm with a running time of $m^{O(1/\epsilon)}$.

For k-CNF, the running time of the deterministic algorithms of Moser [17] and of Moser–Tardos [18] can be shown to be $m^{\Omega(k^2)}$. In this paper, we extend this work in several ways. We give a deterministic algorithm that runs in time $\tilde{O}(m^{2(1+\frac{1}{\epsilon})})$ to find a satisfying assignment 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 that works for the more general asymmetric version of LLL in the algorithmic framework of Moser and Tardos [18]. We also give deterministic parallel algorithms in this framework.

Organization. In the next section we first give an introduction to the (non-constructive) LLL (see Section 2.1). We show its application to the (k-)SAT problem (Section 2.2) and give the Moser-Tardos framework (Section 2.3) and the Moser-Tardos algorithm (Section 2.4). In Section 3 we give the formal statement of our results and their implications for the k-SAT application. Section 4 follows with an intuitive description and summary of the new ideas in the paper. We define the main ingredient – the partial witness structure – formally in Section 5. We give the sequential deterministic algorithm and prove that it works efficiently in Section 6 and finally present the parallel algorithm and its running time analysis in Section 7.

2 Preliminaries

In this section we first give three nonconstructive versions of the LLL which are due to Erdős and Lovász [6]. We give intuition into their meaning and relations to each other. In Section 2.2 we show how to apply the LLL to the k-SAT and SAT problem. In Section 2.3 we give the algorithmic framework of Moser–Tardos [18] and in Section 2.4 we give the Moser–Tardos algorithm. Both form the foundation upon which this thesis is built.

2.1 The Lovász Local Lemma

The Lovász Local Lemma is a powerful theorem in the field of probability theory. It gives bounds on the probability of avoiding a possibly large number of "bad" events that are not "too dependent" on each other. To understand the situations in which such a theorem might be useful we imagine a random process that leads to a desired result if nothing goes wrong. In probability theory a class of outcomes of a random experiment is captured by the notion of an event (see [2] for the basics of probability theory). In our example we have a number of "bad events" for which we want none to occur. A general upper bound on how large the probabilities of these events can be is given by the

union bound over all events. Clearly the worst case situation is that all bad events are mutually exclusive. In this case the probability of something going wrong is simply the sum over the probabilities of all bad events. There is a chance that nothing goes wrong iff this sum is smaller than one. Although this is an optimal criterion in the worst case it is often too pessimistic and quickly fails if one deals with many bad events. In many of cases one thus has to exploit that bad events are usually not as highly correlated. If one looks for example at the extreme case that all bad events are independent of each other it is clear that regardless of the number of bad events there is always a non-zero chance of avoiding all of them as long as no bad event happens with probability one. The Lovász Local Lemma deals with cases between these two extreme examples and can be seen as an interpolation between the union bound for arbitrarily dependent events and the case of completely independent events. The LLL relates the probabilities of events to its dependencies and gives a condition that is sufficient for having a non-zero probability of avoiding all bad events. It is formulated in the following formal setting:

Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be events in a probability space and G be an undirected graph on $\{1, \ldots, m\}$ with the property that every event $A_i \in \mathcal{A}$ is mutually independent[1] of the set of events all events A_j with $\{i, j\} \notin E(G)$. We call this set the non-neighbors $\overline{\Gamma(A)}$ in contrast to the neighbors $\Gamma(A) := \{A_j \in \mathcal{A} \mid \{i, j\} \in E(G)\}$.

Using this setting the most general version of the LLL which will also be the one used throughout the rest of this paper is:

Theorem 1. *[6] In the above setting, suppose there exists an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that for all $A \in \mathcal{A}$, the following inequality holds for all $A \in \mathcal{A}$:*

$$\Pr(A) \leq x'(A) := x(A) \prod_{B \in \Gamma(A)} (1 - x(B)).$$

Then the probability of avoiding all events in $\mathcal{A}$ is non-zero. More precisely we have

$$\Pr\left(\bigcap_{A \in \mathcal{A}} \overline{A}\right) \leq \prod_{i=1}^{m} m(1 - x_i) > 0.$$

A good intuition for the LLL in this very general form is to interpret the x_i values as corrected probabilities in a system of independent events: In the event system given

---

[1] An event A is mutually independent of a set of events $\{A_1, \ldots, A_m\}$ iff $\Pr(A) = \Pr(A \mid f(A_1, \ldots, A_m))$ for any function f

to the LLL there are dependencies. Thus conditioning on some events not occurring can increase the probability of dependent events A to say $x(A)$. In this sense the value $x(A)$ can be seen as an upper bound on how much influences through dependent events can increase $\Pr(A)$. The LLL-condition then becomes condition which checks whether a given set of upper bounds is high enough and whether the underlying dependent event system $\mathcal{A}$ can safely be treated as an independent system in which event A has probability $x(A) > \Pr(A)$. The probability that nothing bad happens in this independent system is exactly $\prod_{A \in \mathcal{A}}(1 - x(A))$ which is positive if no corrected probability is one.

It is instructive to look at a special case in which the $x(A)$ are set to $e\Pr(A)$, i.e. we restrict the lemma to the case where the probabilities of a dependent event A blows up by at most a factor of e through the influences of its "neighbors" in G, i.e., the events depending on it. In this case the statement essentially simplifies to:

Theorem 2. *[6] Assuming again the above setting, if for each event $A \in \mathcal{A}$ we have:*

$$\sum_{A' \in \Gamma(A)} \Pr(A') < \frac{1}{4}$$

then the probability of avoiding all events in $\mathcal{A}$ is non-zero. If all probabilities are small the requirement can be weakened to

$$\sum_{A' \in \Gamma(A)} \Pr(A') < \frac{1}{e}.$$

Phrased in words this version states that it suffices that the subset of neighbors on which an event A depends has probability mass at most $\frac{1}{e}$. Note that this is essentially the union bound over a local neighborhood in G. This viewpoint demonstrates the surprising power of the LLL. Since events in a neighborhood can all depend on each other it is clearly necessary that a union bound over events in such a neighborhood is at most one because otherwise it is (in the worst case) impossible to avoid even just these events. The LLL just demands that these union bounds be only a little smaller. On the other hand it guarantees the existence of a strong global property from just this small local condition. (Note that the condition does not at all depend on the number of events.) In short the local condition that from the point of view of any event the union bound over its neighborhood implies that they are avoided with probability at least $1 - \frac{1}{e}$ is sufficient to guarantee that there is a global solution that avoids all events simultaneously.

Lastly we state a symmetric version of the LLL that assumes that all events have (roughly) the same probability (and degree in G). For this case the local union bound in

9

Theorem 2 simplifies to bounding the probability mass pd of d events with probability p and thus gives:

Theorem 3. *[6] Assuming again the above setting, if each event $A \in \mathcal{A}$ has probability at most p and at most d neighbors in G than the condition $p(d+1)e < 1$ suffices for the probability of avoiding all events in $\mathcal{A}$ to be positive.*

While this is the simplest form of the LLL it is by far the most frequently applied version.

2.2 A typical LLL-application: (k-)SAT with few variable intersections

To show how the (non-constructive) LLL is applied the following section gives a typical example application of the LLL: SAT. We will look at a CNF formula F and use the LLL to get a condition that allows us to prove satisfiability. This is done via the probabilistic method which is based on the trivial fact that a structure that is produced with non-zero probability by a random process has to exist. In this case we will draw a uniformly random assignment to the variables of F and prove that a satisfying assignment has to exist because the LLL guarantees that with positive probability such a random assignment does not violate any clause. The probability that a uniformly random assignment violates a clause $C \in F$ with $k = |C|$ variables is exactly 2^{-k}. To embed the problem into the setting of the LLL we define the probability space to be the uniform random distribution of all possible assignments and have a bad event A_C for every clause C in F that becomes true iff C is violated. The events of two clauses C and C' being violated are clearly independent if the two clauses do not share a variable. Also if we take G as the graph on the clauses or equivalently events that connects two clauses iif they share a variable it is easy to see that the mutual independence assumption on the non-neighbors is satisfied. These are all the pieces we need; plugging them into Theorem 2 gives:

Corollary 4. *Every CNF formula F which satisfies the property that for every clause $C \in F$ the set $\Gamma(C)$ of clauses sharing a variable with C satisfies*

$$\sum_{C' \in \Gamma(C)} 2^{-|C'|} < \frac{1}{e}$$

is satisfiable.

Specializing this Corollary to k-CNF gives us events with uniform probabilities of magnitude 2^{-k} and thus allows us to apply the symmetric LLL from Theorem 3:

Corollary 5. *Every k-CNF formula F in which every clause shares variables with less than $\frac{2^k}{e}$ other clauses is satisfiable.*

2.3 Algorithmic Framework

To get an algorithmic handle on the LLL, we have to move away from the abstract probabilistic setting of the original LLL and impose some restrictions on the representation and form of the probability space under consideration. In this paper we follow the general algorithmic framework for the LLL due to Moser–Tardos [18] which is inspired from Mosers earlier paper [16] on the model application k-SAT given in Section 2.2. The framework is as follows:

Every event in a finite collection of events $\mathcal{A} = \{A_1, \cdots, A_m\}$ is determined by a subset 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 the **variable set** of an event $A \in \mathcal{A}$ by $\mathrm{vbl}(A)$ and define it as the unique 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 $A, B \in \mathcal{A}$, $A \neq B$ if $\mathrm{vbl}(A) \cap \mathrm{vbl}(B) \neq \emptyset$. 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; the objective is to find a point in the probability space, or equivalently, an evaluation of the random variables from their respective domains such that none of the bad events happen. We call such an evaluation a **good evaluation**.

Moser and Tardos [18] gave a constructive proof of the general version of the LLL (Theorem 6) in this framework using Algorithm 1 below. This framework seems sufficiently general to capture most applications of LLL. Nevertheless there are a few problems in which the LLL is not applied to a product-probability space that is induced by independent random variables. The most prominent example is the classical paper [7] "Lopsided Lovász local lemma and latin transversals" by Erdős and Spencer in which the LLL is applied to the space of random permutations. The extension [12] "Using the Lovász Local Lemma in the space of random injections " by Lu and Székely also falls into this category.

2.4 The Moser–Tardos Algorithm

This section states the constructive LLL result for the general version of the LLL (Theorem 6) as given by Moser and Tardos [18] and states their algorithm (see Algorithm 1). Both build on the framework presented in Section 2.3.

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 on evaluation $(P = v_P : \forall P \in \mathcal{P})$, 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 6. *[18] If there exists an assignment of reals $x : \mathcal{A} \rightarrow (0,1)$ such that for all $A \in \mathcal{A}$, the following inequality holds:*

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

then the expected number of resamplings done by Algorithm 1 before it terminates with a good evaluation, is $O(\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)})$.

3 Our results

This section formally states the new results established in this thesis.

For the rest of this paper, we will use the following parameters defined using $x(A)$ and $x'(A)$ as in Theorem 6 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 $\overline{\mathcal{A}} := \{A \in \mathcal{A} \mid x'(A) \geq \frac{1}{4m}\}$.

12

- $w_{min} := \min_{A \in \mathcal{A}} \{-\log x'(A)\}$. [2]

We expect these parameters to be polynomial in the input size. For M this is easily true as long as $1 - x(A)$ is not to small because $|\text{vbl}(A)| < n$, $x(A) < 1$ and $1/x'(A) \leq 4m$ for any $A \in \overline{\mathcal{A}}$. We expect also expect w_{min} to be a constant or even logarithmically growing in most applications. It is easy to prove that at least $w_{min} \geq 1/M$ by showing that for all $A \in \mathcal{A}$ we have $\log \frac{1}{x'(A)} \geq 1/M$:

For all $A \in \mathcal{A} \setminus \overline{\mathcal{A}}$ we have $\log \frac{1}{x'(A)} \geq \log 4m \geq 1 \geq 1/M$.

For all $A \in \overline{\mathcal{A}}$ we have $1/M \leq 1/\left(4 \cdot \frac{2|\text{vbl}(A)|}{x'(A)} \cdot \frac{x(A)}{1 - x(A)}\right) \leq \frac{x'(A)}{8x(A)}(1 - x(A)) \leq \frac{1}{8}(1 - x'(A)) \leq \log \frac{1}{x'(A)}$.

The complexity assumptions regarding the conditional probabilities in the following theorem are same as the ones used by Moser-Tardos [18].

Theorem 7. *Let the time needed to compute the conditional probability* $\Pr[A \mid \forall i \in I : P_i = v_i]$ *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. *Suppose there is an* $\epsilon \in (0, 1)$ *and an assignment of reals* $x : \mathcal{A} \to (0, 1)$ *such that for all* $A \in \mathcal{A}$, *the following inequality holds:*

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

Then there is a deterministic algorithm that 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 8. *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 [17] and of Moser–Tardos (specialized to k-CNF) [18] with running time $m^{\Omega(k^2)}$, which is superpolynomial for $k =$

---

[2] In this paper log denotes logarithm to base 2

13

$\omega(1)$. 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 for an LLL related problem was given for the hypergraph 2-coloring problem: 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 [14]. Our theorem above applies equally well to the 2-coloring problem to state that there exists a deterministic algorithm that runs in time $\tilde{O}(m^{2(1+\frac{1}{\epsilon})})$ to 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 algorithm works under a fairly general complexity assumption regarding the events, which is valid in many applications; for example, our algorithm applies to k-CNF and hypergraph 2-coloring.

We also give a parallel deterministic algorithm. Here we make an additional assumption about the events – that their decision tree complexity is small. This assumption seems quite general: it includes applications to k-CNF and hypergraph 2-coloring.

Theorem 9. *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 for all $A \in \mathcal{A}$, the following inequality holds:*

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

*If there exists a constant c such that every event $A \in \mathcal{A}$ has **decision tree complexity**[3] 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 an EREW PRAM.

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

---

[3]Informally, we say that a function $f(x_1, \ldots, x_n)$ has decision tree complexity at most k if we can determine its evaluation by adaptively querying at most k of the n input variables.

Corollary 10. *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.*

4 Techniques

In this section, we informally describe the main ideas of our approach 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 thesis. For the sake of exposition in this section, we omit numerical constants in some mathematical expressions. Familiarity with the Moser–Tardos paper [18] is useful but not necessary for this section.

4.1 The Moser–Tardos Derandomization

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 $1/2^k \le 1/(2m)$. Thus the probability that on a random assignment, F has an unsatisfied clause is $\le 1/2$, and hence a satisfying assignment can be easily found in polynomial time using the method of conditional probabilities (see, e.g., [14]). Henceforth, we assume that $k \le \log m$. 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 algorithms of Moser and of Moser–Tardos which work in polynomial time, albeit 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 using T: instead of randomly sampling afresh each time when a new evaluation for a variable is needed, we pick its next unused value from T. The fact that the randomized algorithm terminates quickly in expectation (Theorem 6), implies that there exist small tables (i.e., small lists for each variable) on which the algorithm terminates with a satisfying assignment. The deterministic algorithm finds one such table.

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 the following 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 to find a small table with which no tree in L is consistent.

3. The list L is of polynomial size. We need this property for the method of conditional probabilities to be efficient.

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 that they use for derandomization. The *expectation lemma* states that the expected number of large (size at least $\log m$) consistent witness trees is less than $1/2$ (here randomness is over the choice of the table). We could try to use the method of conditional probabilities 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 using the method of conditional probabilities 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 probabilities 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 natural approach to resolve the issue of a large list is to find a way to implicitly maintain the list and carry out the method of conditional probabilities 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 that we now describe.

4.2 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 in the sense that 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 to 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 probabilities 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.

4.3 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 $(ed)^u \leq 2^{(1-\epsilon)ku}$ (see [10]). Thus the 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 tree 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 probabilities needs to maintain has polynomial size.

4.4 General Version

More effort is needed to obtain a deterministic algorithm for the asymmetric version of the LLL in which events are allowed to have significant variation in their probabilities and unrestricted structure.

One issue is that an event could possibly depend on all n variables. In that case, taking all variable subsets of a label for the root of a partial witness would give up to 2^n different possible labels for the roots. However, for the range lemma to hold true, we do not need to consider all possible variable subsets for the root, instead it is sufficient to have for each root event a pre-selected choice of $2n$ subsets that allows to partition each partial root into two such subsets. This pre-selected choice of variable splittings $\mathbb{B}_A$ is arbitrarily fixed for each event A in the beginning.

The major difficulty in derandomizing the asymmetric LLL is in finding a list L satisfying the three properties mentioned earlier for applying the method of conditional probabilities. The range lemma can still be applied. However, the existence of low probability events with (potentially) many neighbors may lead to as many as $O(m^u)$ partial witnesses of size in the range $[u, 2u]$. Indeed it can be shown that there are instances in which there is no setting of u such that the list L containing all witnesses of size in the range $[u, 2u]$ satisfies properties (2) and (3).

The most important ingredient for working around this 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 finds a list L that 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. It is easy to prove a similar range lemma for weight based partial witnesses which guarantees property (1) for this list. Further, the value of γ can be chosen so that the expectation lemma of Moser and

Tardos can be adjusted to lead to property (2) for L. Unfortunately one cannot prove property (3) by counting the number of partial witnesses using combinatorial enumeration methods as in [17, 18]. This is due to the possibility of up to $O(m)$ neighbors for each event A in the dependency graph. Instead we use the strong coupling between weight and probability and obtain property (3) directly from the expectation lemma.

4.5 Parallel Algorithm

For the parallel algorithm, we use the technique of limited-independence spaces, or more specifically k-wise δ-dependent probability spaces due to Naor and Naor[19] and its extensions [8, 4]. This is a general technique for derandomization. The basic idea here is that instead of using perfectly random bits in the randomized algorithm, we use a limited-independence probability space. For many algorithms it turns out that their performance does not degrade when using bits from a such probability space; but now the advantage is that these probability spaces are smaller in size and so one can enumerate all the sample points in them and choose a good one, thereby obtaining a deterministic algorithm. This tool was applied by Alon [1] to give a 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 Alon uses it: Alon starts with a random 2-coloring of the hypergraph chosen from a small size limited independence space; he then shows that at least one of the sample points in this space has the property (roughly speaking) that the monochromatic and 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.

In contrast, our algorithm is very simple: recall that for a random table the expected number of consistent partial witnesses with size in the range $[\log m/\epsilon k, 2\log m/\epsilon k]$ is at most $1/2$. Each of these partial witnesses uses at most $\frac{2\log m}{\epsilon k} \cdot k$ entries from the table. 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 $\frac{2\log m}{\epsilon}$ entries has the same joint distribution as in the original random table). So any partial witness tree is consistent with the new random table with the same probability as before. And hence the expected number of partial witnesses consistent 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 only require that the decision tree complexity of the event is small. The idea behind proof remains the same.

5 The Partial Witness Structure

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

5.1 Definitions

For every $A \in \mathcal{A}$ we fix an arbitrary rooted **binary variable splitting** $\mathbb{B}_A$. 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 distinct 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 variable splitting which are actually subsets of vbl(A). The binary variable splitting is not to be confused with the witness tree which we define next. The elements from $\mathbb{B}_A$ will be solely used to define the possible roots of partial witness trees.

A **partial witness tree** τ_S is a finite rooted tree whose vertices apart from the root are labeled by events from $\mathcal{A}$ while the root is labeled by some variable subset $S \in \mathbb{B}_A$ for some $A \in \mathcal{A}$. Each child of the root must be labeled by an event A that depends on at least one variable in S (thus a neighbor of the root event); the children of every other vertex labeled by B must be labeled by a neighboring event 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 vbl(A) for some $A \in \mathcal{A}$. In such a case, we relabel the root with A instead of vbl(A). Note that this definition of full witness tree matches the one of witness trees in [18].

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 [16], 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 at random 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 that 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 (with repetitions), which we call the **event-log**. Let the ordering of the events as they have been selected for resampling in

21

the execution of Algorithm 2 using a table T be denoted by an event-log C_T. Note that C_T 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 $S \cap \mathrm{vbl}(C(i))$ 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 $S = \mathrm{vbl}(A) \in \mathbb{B}_A$, in which case the witness tree is identified with a full witness tree with root A. For such a full witness tree, our construction matches the construction of *witness trees associated with the log* in [18].

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.

Most of these definitions are (simple) extensions of the ones given by Moser and Tardos [18]. The intuition behind the witness tree structure is the following. A witness (tree) is supposed to certify or explain why a certain resampling happened. Besides the requirement that any resampling can be explained by such a witness it has to solve the balance between being concise (i.e. there are not too many explanations) and specific enough to be unlikely to happen. This is needed to conclude via a union bound that whp. there is no valid explanation for a long running time. The witness tree construction does exactly this. It goes back in time starting from the resampling A it explains and only includes prior resamplings whose variable changes transitively lead to the resampling of A. The leveled structure of a tree is used to record necessary (partial) order information about the resamplings that is needed to eliminate any dependencies to unrelated resamplings (see beginning of the next section). Note that keeping track of the partial sequence, i.e. the full order information would lead to a super-polynomial ($O(\log M)$)! running time

instead of the $2^{O(\log M)}$ achieved here.

5.2 Proofs: Coupling and Range Lemma

The following coupling lemma is simple but an essential piece in proving unlikelihood of (large) witnesses. Its proof shows that the leveled structure of the trees captures enough of the partial resampling order to determine exactly which entry from the input table is used for which resampling. In other words: Given a witness tree one can use the T-check procedure to exactly determine which values where used in the resamplings that lead to the resampling explained by the witness tree.

Lemma 11. *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 [18].

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 vbl([u]) and 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 $\mathcal{D}(p)$ be the set of vertices $w \in \tau_S$ with depth greater than that of v such that $[w]$ depends on variable p.

When the T-check considers the vertex v labeled by B and uses the next unused evaluation of the variable P, it uses the evaluation $T(p, |\mathcal{D}(p)|)$. This is because the witness check visits the vertices in order of decreasing depth and thus has used values for p exactly $|\mathcal{D}(p)|$ times. Note for this also that vertices with equal depth are variable disjoint and thus among the vertices with depth equal to that of v, only $[v]$ depends on 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 $T(p, |\mathcal{D}(p)|)$. This is because the pointer for P was increased for events $[w]$ that were resampled before the current instance, where $w \in \mathcal{D}(p)$. Note that every

event which was resampled before t_v and that depends on $[v]$ would be present at depth greater than that of v in τ_S by construction. Hence, $\mathcal{D}(p)$ 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 $\mathrm{vbl}([v])$ when considering v, it also must find that $[v]$ happens. ∎

Next, we prove that partial witnesses allow a range lemma with an improved factor two range. Its proof makes use of the fact that the partial witness root with its variable splitting order $\mathbb{B}_A$ can always be reduced into two parts. For this proof it can be helpful to think of the witness trees as a longest-path skeleton of a DAG that captures all inter-dependencies between prior resamplings. For a DAG it clearly holds that if one looks at the sub-DAG that is transitively induced by two disjoint subsets of children of the root their (not necessarily disjoint) union remains the full DAG. This is essentially also what is used to prove the range lemma for the partial witness tree structure.

Lemma 12. *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 that 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): $\mathrm{Root}(\tau_S)$ has only one child. Let this child be labeled by v. Let t' be the largest instant before t at which $[v]$ was resampled. Note that this resampling of $[v]$ corresponds to the child v of the root of τ_S. Now, consider the partial witness tree $\tau_S' = \tau_{C_T}(t', S' = \mathrm{vbl}([v]))$. 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): $\mathrm{Root}(\tau_S)$ has at least two children. Because the children are variable disjoint and both have to share a variable with S we know that S consists of at least 2 variables. It thus also has two children in the variable splitting $\mathbb{B}_A$. In $\mathbb{B}_A$, starting from S, we now explore the descendants 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. It is easy to see that this stops at the latest when S_R and S_L are leaves in $\mathbb{B}_A$.

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. ∎

6 Deterministic Algorithm

In this section we describe our sequential deterministic algorithm and prove Theorem 7.

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. We choose $\gamma = \frac{\log M}{\epsilon}$ and show in Lemma 13 that this suffices to make the expected number of forbidden witnesses smaller than one. The splitting into high and low probability (weight) events generalizes the argument that the k-CNF problem is interesting only if $k < \log m$. Again, events with large weight have very small probability, and so the expectation of occurrence of their singleton witnesses is negligible. This makes it easy to treat them separately, thereby handling the first case in the proof of range lemma. Formally, we recall that $\overline{\mathcal{A}} = \{A \in \mathcal{A} \mid x'(A) \geq \frac{1}{4m}\}$ and define $F := F_1 \cup F_2$ where

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

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

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 probabilities: 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 probabilities quickly. To this end, we prove that the number of forbidden witnesses is polynomial using Lemma 14. Further, we prove that the table given by the method of conditional probabilities is a good table; that is, the execution of Algorithm 1 using this table terminates within polynomial number of steps. We show this as follows: Lemma 13 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 probabilities while obtaining the table). Therefore, the sequential algorithm does not encounter any of the forbidden witnesses. Putting this together with Lemma 12, 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 13. *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 11, 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

26

the random table T is $\prod_{v \in \overline{V}(\tau)} \Pr([v])$, where again $\overline{V}(\tau)$ are all non-root vertices in τ. Using this and the assumption in Theorem 7 that $\Pr([v]) \leq x'([v])^{1+\epsilon}$ 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 variable splitting $\mathbb{B}_A$. Hence, we can rewrite to get that the expectation is

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

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

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

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

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

Next we transition from partial to full witness trees by including the root again (and going from $\overline{V}$ to V).

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

Now we can use the following result of Moser–Tardos [18] that bounds the expected number of full witnesses with root A:

$$\sum_{\tau \in \Upsilon'_A} \prod_{v \in V(\tau)} x'([v]) \leq \frac{x(A)}{1 - x(A)}.$$

Their proof makes use of a Galton–Watson process that randomly enumerates proper

witness trees with root A (note that by Lemma 11 all partial witness trees are proper). Using this we can finish the proof that the expected number of witnesses $\tau \in F_2$ occurring in C_T is at most $\frac{1}{4}$:

$$E \leq \sum_{A \in \mathcal{A}} \frac{|\mathbb{B}_A|}{x'(A)} \cdot \left(\frac{x(A)}{1 - x(A)} \right) 2^{-\gamma\epsilon}$$
$$< \frac{M}{4} 2^{-\gamma\epsilon} \leq \frac{1}{4}.$$

Here the penultimate inequality follows from $|\mathbb{B}_A| < 2|\mathrm{vbl}(A)|$ and the definition of M and the last inequality follows from the choices of $\gamma = \frac{\log M}{\epsilon}$.

To calculate the expected number of all 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:

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

■

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 14. *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:

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

Here the final equality is comes from the proof of Lemma 13 which also proves the upper bound of $\frac{1}{4}$. 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

$$|F| = |F_1| + |F_2| \leq m + \frac{1}{4}M^{2(1+1/\epsilon)} < M^{2(1+1/\epsilon)}.$$

∎

Now, we are ready to prove Theorem 7.

Proof of Theorem 7. The first step of the deterministic algorithm enumerates all forbidden witnesses in F. By Lemma 14, 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 13; this invariant is maintained while picking values 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 12. 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. $\blacksquare$

From the general deterministic algorithm it is easy to obtain the corollary regarding k-CNF by using the standard reduction to the symmetric LLL and plugging in the optimum values for the parameters.

Proof of Corollary 8. 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, $\mathrm{Pr}\left(A\right) = 2^{-k}$. 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 $\mathrm{Pr}\left(A\right) \leq x'(A)^{1+\epsilon} \; \forall A \in \mathcal{A}$. Therefore, we use parameters $t_C = O(k)$, $w_{min} \approx k = \Omega(1)$, $D = 2$, $|\mathrm{vbl}(A)| = k$ and obtain a parameter of $M = O(mk) = O(m \log m)$. With these parameters the corollary follows directly from Theorem 7. $\blacksquare$

7 Parallel Algorithm

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

Definition 1. (k, δ)-*approximations[8]: 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 $\mathrm{Pr}_{\mathcal{D}}[v_I]$ 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 $\mathrm{Pr}_{\mathcal{Y}}[v_I]$ 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) in parallel: 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 1). 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 15). 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 1. *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.*

   1. 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 $k_1 + k_2$.

   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.:

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

31

Proof. For the first claim we recall that a function having decision tree complexity at most k is equivalent to saying 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 for f with depth at most k. Each one of the leaf-to-root paths in this tree corresponds to a partial assignment of values to at most k components which 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 $D^k \delta$. $\blacksquare$

Remark: 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 weaker 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 suggested property itself behaves additively.

Corollary 15. *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 1, 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 $c \log M \le c\gamma$; else if $\tau \in F_2$, then

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

$$\le 2c\gamma.$$

Lemma 13 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 1

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 14 we have $|F| \leq M^{2+2/\epsilon}$. $\blacksquare$

We obtain the proof of Theorem 9 along the outline mentioned in the beginning of the section.

Proof of Theorem 9. We use Algorithm 4 to obtain the good evaluation. Corollary 15 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 [18] and by using Lemma 4.1 of [18], if this algorithm runs for i iterations then there exists a consistent witness of height i. Such a witness has weight at least $i w_{\min}$ 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 correctness. 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 [8] (which in turn uses a construction in [19]). 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 [8] can be parallelized with the required parameters easily. $\blacksquare$

Again it is easy to obtain the k-CNF result as a corollary of the asymmetric algorithm.

Proof of Corollary 10. We apply the LLL in the same way to k-CNF as in the proof of Corollary 8 getting $M = O(m \log m)$ and $w_{min} \approx k = \Omega(1)$. Since each clause depends only on k variables a decision tree complexity of $O(k)$ is obvious. Finally using Theorem 9 and for example an algorithm of Luby [13] to compute the maximal independent set in time $t_{MIS} = O(\log^2 m)$ leads to the claimed running time. $\blacksquare$

8 Conclusion

Moser and Tardos [18] 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 those ϵ-slacks can be removed remains open.

Similar to Moser-Tardos, our sequential algorithm needs the assumption that exact conditional probabilities can be computed efficiently. This assumption can be weakened to use pessimistic estimators. The complexity assumptions of our parallel algorithm are alternative assumptions. These assumptions may also be weakened (see Remark in Section 7).

We have an improved parallel algorithm that determines a valid assignment directly from a good table needing only polynomially many non-adaptive evaluations. Its running time is faster than the parallel algorithm presented here. This parallel algorithm is furthermore 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 the sequential and parallel algorithms presented here. 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] N. Alon and J. Spencer. The Probabilistic Method. 2005.

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

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

[5] 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.

[6] 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. Erdős on his 60th birthday)*, volume 2, pages 609–627, 1975.

[7] P. Erdős and J. Spencer. Lopsided Lovász local lemma and latin transversals. *Discrete Applied Mathematics*, 30(2-3):151–154, 1991.

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

[9] U. Feige. On allocations that maximize fairness. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 287–293, 2008.

[10] D. Knuth. *The art of computer programming, volume 1: fundamental algorithms*. Addison-Wesley, Reading, MA, 1969.

[11] 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.

[12] L. Lu and L. Székely. Using the Lovász Local Lemma in the space of random injections. *Electronic Journal of Combinatorics*, 13(R63):1, 2007.

[13] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal of Compututing*, 15(4):1036–1053, 1986.

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

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

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

[17] 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, 2009.

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

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

[20] A. Srinivasan. Improved algorithmic versions of the lovász local lemma. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 611–620, 2008.