ABSTRACT

Title of Dissertation ALGORITHMS AND GENERALIZATIONS FOR THE LOVÁSZ LOCAL LEMMA

David G. Harris, Doctor of Philosophy, 2015

Dissertation directed by: Professor Aravind Srinivasan, Department of Computer Science

The Lovász Local Lemma (LLL) is a cornerstone principle of the probabilistic method for combinatorics. This shows that one can avoid a large of set of "bad-events" (forbidden configurations of variables), provided the local conditions are satisfied. The original probabilistic formulation of this principle did not give efficient algorithms. A breakthrough result of Moser & Tardos led to an framework based on resampling variables which turns nearly all applications of the LLL into efficient algorithms. We extend and generalize the algorithm of Moser & Tardos in a variety of ways.

We show tighter bounds on the complexity of the Moser-Tardos algorithm, particularly its parallel form. We also give a new, faster parallel algorithm for the LLL.

We show that in some cases, the Moser-Tardos algorithm can converge even though the LLL itself does not apply; we give a new criterion (comparable to the LLL) for determining when this occurs. This leads to improved bounds for $k$-SAT and hypergraph coloring among other applications.

We describe an extension of the Moser-Tardos algorithm based on partial resampling, and use this to obtain better bounds for problems involving sums of independent random variables, such as column-sparse packing and packet-routing.

We describe a variant of the partial resampling algorithm specialized to approximating column-sparse covering integer programs, a generalization of set-cover. We also give hardness reductions and integrality gaps, showing that our partial resampling based algorithm obtains nearly optimal approximation factors.

We give a variant of the Moser-Tardos algorithm for random permutations, one of the few cases of the LLL not covered by the original algorithm of Moser & Tardos. We use this to develop the first constructive algorithms for Latin transversals and hypergraph packing, including parallel algorithms.

We analyze the distribution of variables induced by the Moser-Tardos algorithm. We show it has a random-like structure, which can be used to accelerate the Moser-Tardos algorithm itself as well as to cover problems such as MAX $k$-SAT in which we only partially avoid bad-events.

# ALGORITHMS AND GENERALIZATIONS OF THE LOVÁSZ LOCAL LEMMA

by

David G. Harris

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:
Professor Aravind Srinivasan, Chair
Professor William Gasarch
Professor MohammadTaghi HajiAghayi
Professor Joel Spencer
Professor Lawrence Washington

# Dedication

Thanks to my advisor Aravind Srinivasan,

for teaching me so much

and for tirelessly supporting my development as a mathematician

This thesis is dedicated

to my wife Joana and children Tomás and Samuel

(sorry for my vacant stares for the past three years;

this thesis is the reason why)

# Acknowledgments

I would first like to thank so many people at University of Maryland for creating such a positive learning environment. The staff in the Mathematics department and Applied Mathematics and Statistics and Scientific Computing (AMSC) programs have, since I applied to the school, have constantly guided me through all the requirements of the program, answered all my administrative questions, checked to make sure I would be able to take all the courses I needed to, and provided feedback and tips to save time. I especially want to thank Alverda McCoy, the AMSC program coordinator, for answering innumerable administrative questions.

I also have taken a number of wonderful courses and seminars. I would like to thank, in particular, Samir Khuller's introductory algorithms class and scheduling algorithms seminar, Bill Gasarch's course on automata theory and complexity seminar, Jonathan Katz's class on complexity theory, and David Kueker's classes on logic, incompleteness, and set theory.

The collaboration with my advisor Aravind Srinivasan has been phenomenal. It astonishes me how wide his command of mathematics and computer science is, and how he can suggest some many fruitful avenues to explore. At the same time he is utterly devoted to his students' development and finding ways to make their careers successful. I cannot thank him enough.

This thesis would not be possible without support from my employer, the U.S. Federal Government, especially for the first two years in which I was given time to take coursework.

I collaborated with many other researchers on the LLL in general and on the papers which make up this thesis in particular. These include Dmitris Achlioptas, Antares Chen, Vance Faber, Bernhard Haeupler, Hsin-Hao Hsu, Dana Moshkovitz,

Joel Spencer, and Jan Vondrak. Chapter 3 is co-authored with Bernhard Haeupler. Chapter 6 is co-authored with Antares Chen. Many other sections are co-authored with Aravind Srinivasan. I thank all of them for the many engaging and supportive discussions.

I also want to thank my committee members, especially Joel Spencer who had to travel from New York.

My family has been very supportive of all the time and work this thesis has required. I also thank my parents for traveling from Chicago to my dissertation defense.

Finally, I received numerous travel support awards for various conferences and support from NSF Awards CNS 1010789 and CCF 1422569.

# Contents

CHAPTER 0

# Overview

This thesis examines the Lovász Local Lemma (LLL), an important and widely-used technique in probabilistic combinatorics. This technique has been used in constructions ranging from graph theory to coding to network scheduling. The original formulation of the LLL was non-constructive — showing the existence of a favorable combinatorial configuration, but unable to construct it in efficiently.

Building on a long line of research, a breakthrough algorithm recently developed by Moser & Tardos [**87**] has turned nearly all of these non-constructive combinatorial constructions into efficient algorithms. We refer to this simple and remarkable algorithm as the MT algorithm. It has been extended to the parallel and deterministic settings as well.

The first main goal of this thesis is to intensively analyze the behavior of the MT algorithm and its variants. We describe many situations in which it can be made more efficient. For exampe, we reduce the run-time of the parallel MT algorithm from $O(\log^3 n)$ to $O(\log^2 n)$ in many situations, and we develop more efficient algorithms for problems such as monochromatic hypergraph coloring. We also consider a number of situations in which the standard form of the MT algorithm fails to give efficient algorithmic counterparts for the LLL. We show how to modify the MT algorithm so that it can match the LLL in these cases as well. Examples of this include the efficient algorithms for non-repetitive vertex coloring and Latin transversals.

The second main goal of this thesis to consider the MT algorithm as a random process in its own right, which can be more powerful than the LLL. In some cases, we show that the MT algorithm leads to combinatorial configurations which do not follow from the probabilistic LLL; for example, we show that the MT algorithm terminates

even though the LLL criterion is not satisfied. In other cases, we describe variants of the MT algorithm which do not even correspond to the LLL framework. In many cases, the only proof for the existence of certain types of combinatorial configurations (including $k$-satisfiability thresholds and traversals omitting $s$-cliques) is that they are produced by the MT algorithm or one of its extensions. For example, one key result is the following: suppose we have a $k$-SAT instance, in which each variable appears in at most $L \leq \frac{2^{k+1}(1-1/k)^k}{k-1} - \frac{2}{k}$ clauses; then the instance is satisfiable and the MT algorithm finds a solution in polynomial time. By contrast, the previous bound known for this problem was $L \leq \frac{2^{k+1}}{e(k+1)}$. This can be a signficant improvement for modest values of $k$.

### 0.1. Background

The Lovász Local Lemma (LLL), first introduced by Erdős and Lovász in [**34**], is a principle in probability theory which has become a cornerstone of the probabilistic method in combinatorics. Roughly speaking, this principle states that if one has a probability space and a set of "bad-events" in that space, then as long as the events have low probability and are "mostly independent" (more specifically, each bad-event only affects a small number of others), then there is a positive probability that none of the bad-events occur. In a sense, the bad-events act as if they truly independent from each other. Thus, the probability of avoiding all the bad-events is approximately the product of the probabilities of avoiding each of them individually. In particular, the probability of avoiding all the bad-events is positive.

For combinatorial applications, the typical proof-strategy is to design a random experiment which attempts to construct some combinatorial structure. The bad-events, in this context, are certain configurations of subsets of the variables which contradict the desirable structure. If one can show that the experiment succeeds with positive probability, it holds that the desired configuration exists. The key strength of the LLL in this context is that the conditions it requires are "local" to each bad-event.

Unlike most other techniques of the probabilistic method, this principle can be used to show scale-free structures — results which do not depend upon the number of variables involved or the overall size of the problem instance. This probabilistic principle only shows an exponentially small probability of producing the desired structures, so does not lead to efficient algorithms.

A simple example of this principle comes from *independent transversals*, a combinatorial problem introduced by Bollobás, Erdős, Szemerédi in [**18**]. In this problem, we are given a graph $G$ and a partition of the vertices into blocks of size $b$. We wish to select one vertex from each block (a transversal) without selecting any adjacent vertices (an independent set). A simple application of the LLL shows that this is possible when $b \geq 2e\Delta$, where $\Delta$ is the maximum of the graph; a more sophisticated application improves this to $b \geq 4\Delta$. Note that this result does not depend on the number of vertices, the number of blocks, or any other global properties of the graph.

Hundreds of applications of this principle have since appeared and in many cases the LLL (either applied directly or with some refinements), leads to the best combinatorial bounds. Some examples of these include solving $k$-SAT instances, in which each variable occurs in a limited number of clauses, constructing Latin transversals, and packet routing.

In Chapter 1, we define and review the LLL, and some of its classical extensions. Most of this chapter is background material and not original to this thesis. Section 1.8 appeared as "A note on near-optimal coloring of shift hypergraphs", by David G. Harris and Aravind Srinivasan, Random Structures and Algorithms (2015).

The intuition for why the LLL works is that the bad-events are approximately independent. Unfortunately, as a result of this, the probability of avoiding all the bad-events decreases exponentially with the number of bad-events. This means that the straightforward procedure to generate good configurations, namely directly simulating the random experiment, does not give polynomial-time algorithms; this again is in contrast to other standard techniques of the probabilistic method.

In Chapter 2, we examine the breakthrough LLL algorithm of Moser & Tardos [**87**], which we refer to as the *MT algorithm*. This algorithm is based on the following simple idea: whenever we detect that a bad-event is currently true, we "resample" the variables it affects, that is, we draw their values anew from the original probability distribution. This process converges in a sense, in that the resamplings die off exponentially and one has a configuration which avoids all the bad-events. This turns nearly all applications of the LLL in combinatorics into efficient randomized algorithms.

This algorithm has a number of nice properties and extensions, beyond the fact that it finds configurations which avoid the bad-events. First, it has a simple parallel algorithm. Instead of resampling bad-events one by one, we can select a maximal independent set (MIS) of bad-events, and resample all of its events in parallel. This takes approximately $O(\log^2 n)$ rounds to complete.

Next, there is a nice characterization of the distribution induced on the variables when the algorithm terminates. Roughly speaking, we can show an upper bound on the probability that any event holds on the output distribution, and this upper bound is typically not much higher than the probability of the event originally. This principle has a variety of uses. It can be used to show, for example, that one can form structures whose weight is large, and it can be used to show that that the output of the algorithm has other structural properties that were not explicitly enforced during the resampling step.

Although none of the results in Chapter 2 are new, we give a new and simplified proof for the MT algorithm. This proof highlights certain features of the algorithm that we will extend and explore in later chapters. In particular, we highlight the role of the "Witness Tree Lemma" in the Moser-Tardos proof; this deceptively simple result holds the key to many of the advanced variants and properties of the MT algorithm which we will discuss later in the thesis.

The first main goal of this thesis is to intensively analyze the behavior of the MT algorithm and its variants. We describe many situations in which it can be made more efficient. We also consider a number of situations in which the standard form of the MT algorithm fails to give efficient algorithmic counterparts for the LLL. These include situations in which the underlying probability space does not have independent variables, and cases in which there are exponentially many bad-events. We show how to modify the MT algorithm so that it can match the LLL in these cases as well.

It is somewhat of a historical accident that the probabilistic formulation of the LLL was discovered before the MT algorithm, and that the latter was considered only as a "constructive counterpart" of the former. The second main goal of this thesis to consider the MT algorithm as a random process in its own right. In some cases, we show that the MT algorithm leads to combinatorial configurations which do not follow from the probabilistic LLL; for example, we show that the MT algorithm terminates even though the LLL criterion is not satisfied. In other cases, we describe variants of the MT algorithm which do not correspond to the LLL framework. In many cases, the only proof for the existence of certain types of combinatorial configurations (including $k$-satisfiability thresholds and traversals omitting $s$-cliques) is that they are produced by the MT algorithm or one of its extensions.

## 0.2. Our contributions

In Chapter 3, which is joint work with Bernhard Haeupler, we introduce the concept of a witness dag, which is an extension of the witness tree method used in the original proof by Moser & Tardos. This is a tool that can account for some or all of the resamplings performed, in contrast to the witness tree which accounts for a single resampling. We give two major technical results which result from this: we show that the number of resamplings performed by the MT algorithm is concentrated around its mean, and we give bounds for the run-time of the parallel MT algorithm

which do not depend on the (somewhat mysterious) weighting function used in the asymmetric LLL criterion. We also use witness dags as a computational object to give a new, faster parallel algorithm for the LLL; in most cases, this reduces the runtime from $O(\log^3 n)$ to $O(\log^2 n)$.

In Chapter 4, we show that the MT algorithm can in some cases be shown to converge even though the LLL does not apply. We do this by giving a new convergence criterion, which is similar in flavor to the cluster-expansion LLL criterion of [**17**], but is usually stronger. This takes advantage of the fact that the MT algorithm applies to more specialized probability spaces than does the full LLL; nevertheless, this restricted setting is enough to cover most applications in combinatorics. We use this to develop new efficient algorithms and results for many problems such as $k$-SAT and hypergraph coloring. One key result is the following: suppose we have a $k$-SAT instance, in which each variable appears in at most $L \leq \frac{2^{k+1}(1-1/k)^k}{k-1} - \frac{2}{k}$ clauses; then the instance is satisfiable and the MT algorithm finds a solution in polynomial time. By contrast, the previous bound known for this problem was $L \leq \frac{2^{k+1}}{e(k+1)}$. This can be a significant improvement for moderate values of $k$. For example, for $k = 5$, this improves the bound from $L = 4$ to $L = 5$; for $k = 10$, it improves it from 68 to 79; for $k = 20$, this improves the bound from 36738 to 39568.

We also show a new parallel version of the MT algorithm, which exploits a phenomenon known as *lopsidependency* in the LLL setting. It was previously known that the sequential Moser-Tardos algorithm could make use of this property, but not the parallel algorithm. We develop a variant of the parallel MT algorithm in this setting, and show that it matches the performance of the sequential MT algorithm (which in turn is stronger than the LLL itself).

Chapter 4 appeared as "Lopsidependency in the Moser-Tardos framework: Beyond the Lopsided Lovász Local Lemma", by David G. Harris, Symposium on Discrete Algorithms (SODA) 2015.

In Chapter 5, we introduce a variant of the MT algorithm known as the Partial Resampling Algorithm. The key difference is that in the MT algorithm all the variables affecting a bad-event are resampled, while here we only resample a small, random subset of the variables (where the probability distribution used to select which variables to resample is carefully chosen). This leads to improved bounds and algorithms for a variety of problems, particularly those in which the bad-events are defined by sums of independent random variables. One key result concerns the problem of minimizing makespan for universal packet-routing. A seminal paper of [**73**] showed that a schedule of makespan $O(\text{congestion} + \text{dilation})$ was possible, although the hidden constant term was very large (note that both congestion and dilation are lower bounds are the makespan). By the use of partial resampling, as well as other problem-specific techniques, we are able to reduce this to a bound of $5.70(\text{congestion} + \text{dilation})$. Other applications include give applications to column-sparse packing problems and transversals with omitted $s$-cliques. Many of these results were not known, even non-constructively, prior to this work. This chapter combines two material from papers: "Constraint satisfaction, packet routing, and the Lovász Local Lemma", by David G. Harris and Aravind Srinivasan, Symposium on Theory of Computing (STOC) 2013; and "The Moser-Tardos framework with partial resampling", by David G. Harris and Aravind Srinivasan, Foundations of Computer Science (FOCS) 2013.

In Chapter 6, we discuss a variant of the Partial Resampling Algorithm applied to column-sparse covering integer programs, a generalization of set cover. We also make use here of the improved treatment of lopsidependency found in Chapter 4. This leads to improved approximation rates compared to previous algorithms. Our algorithm also generalizes to problems with an upper bound on the multiplicity of each variable or with multiple objective functions. We show a variety of hardness results and integrality gaps, which demonstrate that for a large range of parameters our algorithm has near-optimal approximation rate. In many cases our algorithm has

approximation ratios which asymptotically optimal (including having the optimal constant term), the first algorithm to achieve this.

Chapter 6 will appear as "Partial resampling to approximate covering integer programs," by Antares Chen, David G. Harris, and Aravind Srinivasan, in the Symposium on Discrete Algorithms (SODA) 2016.

In Chapter 7, we discuss the LLL in the context of random permutations. The LLL has been used here to construct Latin transversals, hypergraph packings, and other applications. The Moser-Tardos algorithm does not apply to this setting, and so this is one of the few areas that had been lacking efficient algorithms. We describe a variant of the Moser-Tardos algorithm, in which we "resample" a variable (an entry in the permutation) by swapping it with a randomly chosen element in the permutation. Although this algorithm is superficially similar to the Moser-Tardos algorithm, it proof is far more complex. We show that this Swapping Algorithm succeeds whenever the LLL criterion is satisfied for the space random permutations.

The algorithm we develop for permutations has nearly all the same nice properties as the original Moser-Tardos algorithm: its output distribution has a nice characterization, we develop a parallel algorithm, and we show that this algorithm can be combined with the partial resampling framework of Chapter 5. These lead to results which are not obtainable by either the original LLLL or more recent generalized LLL frameworks, including an asymptotically-optimal result on matrix transversals with many color occurrences and a parallel algorithm for strong coloring.

Chapter 7 appeared as "A constructive algorithm for the Lovász Local Lemma on permutations," by David G. Harris and Aravind Srinivasan, Symposium on Discrete Algorithms (SODA) 2014.

In Chapter 8, we examine the "MT distribution", which is the distribution on the variables when the Moser-Tardos algorithm terminates. We show a variety of new bounds and applications of this distribution. These include lower bounds on the

probabilities of certain events (which is not normally possible for the MT distribution); we use this to construct high-weight independent transversals. Also, we study when it is possible to partially avoid the bad-events (as opposed to the LLL, which seeks to completely avoid the bad-events). We are able to give an analog of the asymmetric LLL criterion and a parallel MT algorithm in this case. Finally, we discuss a quantitative measure of the randomness present in the MT distribution, namely its Rényi entropy; this leads to new bounds on MAX $k$-SAT and other problems. Parts of Chapter 8 appeared in "The Moser-Tardos framework with partial resampling", by David G. Harris and Aravind Srinivasan, Foundations of Computer Science (FOCS) 2013. Other parts will appear as "Algorithmic and enumerative aspects of the Moser-Tardos Distribution," by David G. Harris and Aravind Srinivasan, Symposium on Discrete Algorithms (SODA) 2016.

In Chapter 9, we study the distribution of the *intermediate* steps of the Moser-Tardos algorithm (before it terminates). We show that this distribution too has a random structure, and that this can be used to accelerate a number of algorithmic applications. This leads to sub-linear algorithms for problems such as Latin transversal. It also gives a strategy for problems in which the number of bad-events is exponential, and so the straightforward implementation of the MT algorithm (checking the bad-events one by one and resampling them if they are true), does not give polynomial time algorithms. One important problem that falls into this class is *non-repetitive vertex coloring*. Previous algorithms, such as [**48**], required an exponentially inflated number of colors for this problem, and had a small but positive probability of giving an invalid coloring. We give an algorithm which requires essentially no additional slack and is "Las Vegas": it succeeds with certainty after an expected polynomial runtime. Chapter 9 will appear as "Algorithmic and enumerative aspects of the Moser-Tardos Distribution," by David G. Harris and Aravind Srinivasan, Symposium on Discrete Algorithms (SODA) 2016.

CHAPTER 1

# The Lovász Local Lemma

## 1.1. Definition of the LLL

The Lovász Local Lemma (LLL), first introduced in [**34**], is a powerful tool in probability theory. We begin with an informal and simplified statement of this principle. Suppose we have a probability space $\Omega$ and a set $\mathcal{B}$ consisting of $m$ events in the space ("bad-events") that we are trying to avoid. We denote this set of bad-events by $\mathcal{B} = \{B_1, \ldots, B_m\}$. Now suppose that each $B \in \mathcal{B}$ has probability at most $p$ and that each $B \in \mathcal{B}$ is "dependent" with at most $d$ other bad-events $B' \in \mathcal{B}$; and if $ep(d+1) \leq 1$ (where $e = 2.71...$ is the base of the natural logarithm), then there is a positive probability that none of the events in $\mathcal{B}$ occur, that is, $P(\bigcap_{B \in \mathcal{B}} \overline{B}) > (1 - \frac{1}{d+1})^m > 0$.

This is not a proper theorem at this point, because we have not defined what it means for two bad-events to be "dependent" with each other. Defining this notion requires a fair amount of care. One cannot simply examine the probability space $\Omega$ to determine if $B$ and $B'$ are dependent with each other. Rather, one must pre-specify a set of independencies, and then verify that $\Omega$ obeys them. To formalize this, we define a *dependency graph* for $\mathcal{B}$.[1]

Definition 1.1. *Suppose $G$ is an undirected simple graph whose vertex set is $\mathcal{B}$. We say that $G$ is a* dependency graph *for $\mathcal{B}, \Omega$, if for every $B \in \mathcal{B}$ and for every $S \subseteq \mathcal{B} - N_G(B)$ (where $N_G(B)$ denotes the inclusive neighborhood of $B$, that is, set*

[1]NB: in this definition, we refer to the neighborhood of a vertex in the graph. We will always define $N(B)$ to be the *inclusive neighborhood*. This is not the standard convention, so the reader should be aware of this when comparing our results to other presentations of the LLL.

*of vertices which are equal to $B$ or are neighbors of $B$), we have*

$$P_\Omega(B \mid \bigcap_{B' \in S} \overline{B'}) = P_\Omega(B)$$

*To state this in words: conditioning on* avoiding *any set of non-neighbors of $B$, does not change the probability of $B$.*

We note that we have defined *a* dependency graph, not *the* dependency graph. There is no unique dependency graph for a set of events $\mathcal{B}$. In general, if $G$ is a dependency graph for $\mathcal{B}$, and $G'$ is obtained from $G$ by adding edges, then $G'$ is a dependency graph for $\mathcal{B}$ as well. However, there is not necessarily any unique *minimal* dependency graph either.

**The variable-assignment LLL setting.** Although the definition of a dependency graph may appear to be quite complicated in general, for most applications of the LLL in combinatorics we use a simple probability space which has a simple and natural dependency graph. Suppose that the probability space $\Omega$ is defined by $n$ variables $X_1, \ldots, X_n$. Each variable $X_i$ independently takes values from some finite set (which may be identified with the integers), such that $X_i = j$ with probability $p_{ij}$. We refer to this probability space as the *variable-assignment setting* for the LLL. Furthermore, suppose now that each $B \in \mathcal{B}$ is a Boolean function of a subset $S_B$ of the variables. In this case, we define the graph $G$ by placing an edge from $B$ to $B'$ iff $S_B \cap S_{B'} \neq \emptyset$; that is, there is a common variable which affects both the events $B, B'$. It is not hard to see that this defines a dependency graph for $\mathcal{B}$, which we define as the *canonical* dependency graph.

Now that we have defined a dependency graph, we can formalize the LLL. We begin with the simplest form, referred to as the "symmetric LLL".[2]

[2]This is not the original form of the LLL, which is now mostly of historical interest, but a later and stronger formulation of it due to Spencer in [**104**]. Also, in the usual formulation of the LLL, $d$ is defined to be the size of the exclusive neighborhood of $B$. Thus the more usual form of this statement is $ep(d+1) \leq 1$.

THEOREM 1.2 ([**104**]). *Suppose that $\Omega$ is a probability space and $\mathcal{B}$ is a set of events in that space. Suppose that $G$ is a dependency graph for $\mathcal{B}$. And suppose that each $B \in \mathcal{B}$ satisfies $P_\Omega(B) \leq p$ and $|N_G(B)| \leq d$. Furthermore suppose that $epd \leq 1$. Then we have*

$$P(\bigcap_{B \in \mathcal{B}} \overline{B}) > (1 - 1/d)^m > 0$$

For the remainder of this thesis, whenever we have fixed $\Omega, \mathcal{B}$ and some dependency graph $G$ for $\mathcal{B}$, we will always write $B \sim B'$ if $B, B'$ are neighbors in $G$ *or* $B = B'$. We will not always refer to $G$ explicitly, as we often simply assume that $G$ is the canonical dependency graph.

**1.1.1. Intuition behind the LLL.** To gain some intuition for what the symmetric LLL criterion means, suppose that the dependency graph $G$ consists of isolated $d$-cliques. Now, consider some clique $K$, and consider the probability that one the events in $K$ occurs. Each such event has probability $\leq p$ and there are $d$ such events, so by the union bound, there is a probability of $\leq pd \leq 1/e$ that an event in $K$ occurs.

Next, consider the probability that any bad-event occurs. As the cliques are isolated, the events in $K$ are *independent* from events in $K'$ for $K \neq K'$. Thus, the probability that no bad-events occur is at least $\prod_{\text{cliques } K}(1 - 1/e) = (1 - 1/e)^{m/d} > (1 - 1/d)^m > 0$.

## 1.2. The LLL and scale-free structures

To understand the power of the LLL, let us consider the problem of *independent transversals*, which we will come back to many times because of its simplicity. Suppose we are given some graph $G$ with a partition of the vertex set $V = V_1 \sqcup V_2 \cdots \sqcup V_k$. A *transversal* is simply a set $T \subseteq V$, which contains exactly one element from each of the classes $V_1, \ldots, V_k$; that is $|T \cap V_i| = 1$ for $i = 1, \ldots, k$. An *independent transversal* is simply a transversal which is also an independent set, that is, there is no edge $\langle u, v \rangle \in G$ such that both $u, v$ are elements of $T$.

There is a large literature on selecting transversals such that the graph induced on $T$ omits certain subgraphs. (This problem was introduced in a slightly varying form by [**18**]; more recently it has been analyzed in [**111, 59, 116, 60, 57**]). An independent transversal is, of course, a transversal which omits the 2-clique $K_2$; this is the most well-studied case.

Now suppose we are given bounds on the sizes of the classes and on the maximum degree of the graph; that is we know that $|V_i| = b$ for $i = 1, \ldots, k$ and the maximum degree is $\Delta$. When we can ensure that the graph $G$ has an independent transversal? Let us consider this problem via the LLL. We define the following natural probability space: for each block $V_i$, we select exactly one vertex in $V_i$ to go into $T$. For each edge $f \in G$, we have a bad-event that both its end-points are in $T$. (This is a form of the variable-assignment LLL, in that the variables $X_1, \ldots, X_n$ correspond to blocks $V_1, \ldots, V_k$ and the value of a variable $X_i$ is the vertex in $V_i$ that is selected.)

Now consider some event corresponding to an edge $f = \langle u, v \rangle$. The probability that both $u, v$ go into $T$ is $p = 1/b^2$. This event is a boolean function of the vertex selected for the blocks corresponding to $u$ and the block corresponding to $v$; the other events which are also affected by these blocks correspond to edges which have an endpoint in these blocks, and there are at most $d = 2b\Delta - 2$ such edges. Thus, the LLL criterion is satisfied if

$$eb^{-2}(2b\Delta - 2) \leq 1$$

which in turn is satisfied if $b \geq 2e\Delta$.

Haxell provides an elegant topological proof that a sufficient condition is $b \geq 2\Delta$ [**54**]; this bound is existentially optimal, in the sense that $b \geq 2\Delta - 1$ is not always admissible [**60, 116, 111**]. So the LLL has shown a bound which is of the right order of magnitude, though the constant terms are slightly sub-optimal.

Now, what is remarkable about this result is that the criterion $b \geq 2e\Delta$ does not depend on the total size of the graph $G$ in any way: it has no dependence on the number of blocks $k$, nor on the number of vertices $|V|$. Thus, we have shown a

*scale-free* property of the graph. We contrast this with other methods in the toolkit of probabilistic combinatorics. Suppose, for example, we want to use a first-moment method to show the existence of an independent transversal. The typical way to do this would be to show that there is a negligible probability that a bad-event occurs; for example, the total probability of all bad-events should sum to $< 1$. In our case, this would give a bound of roughly $|E|/b^2 < 1$, which depends on the total size of the graph.

The basic problem here is that, no matter how rare a bad-event is, when the problem size becomes sufficiently large there is likely to be some bad-event which becomes true. In the LLL, the probability of a bad-event $B$ is compared not to the *total* number of bad-events, but only to the bad-events $B'$ affected by $B$. That is, there is a dependency only on "local" information.

### 1.3. The Lopsided Lovász Local Lemma

In our definition of a dependency graph, we require that avoiding non-neighbors of $B$ does not affect the probability of $B$. In [**35**], Erdős and Spencer noted that what is really needed is that non-neighbors of $B$ does not *increase* the probability of $B$. Thus, if $B$ and $B'$ are positively correlated, in a certain technical sense, then for the purposes of the LLL this is as good as pure independence.

To give some intuition why positive correlation is as good as independence: we are trying to avoid the bad situation in which the bad-events $B \in \mathcal{B}$ completely "tile" the full probability space $\Omega$. That is, it is possible to avoid $\mathcal{B}$ iff $\Omega \neq \cup_{B\in\mathcal{B}} B$. Thus, the worst case is when the events are disjoint from each other, which is a very strong type of negative correlation. If the events overlap with each other, then this decreases the size of $\cup_{B\in\mathcal{B}} B$.

This phenomenon of allowing positive correlation among some bad-events is known as the *Lopsided Lovász Local Lemma*, or LLLL. To formalize all of this, we define the *lopsidependency graph* for $\mathcal{B}$ (also known as a *negative association graph*) as follows:

DEFINITION 1.3. *Suppose $G$ is an undirected simple graph whose vertex set is $\mathcal{B}$. We say that $G$ is a* lopsidependency graph *for $\mathcal{B}$ (on the probability space $\Omega$, which is understood), if for every $B \in \mathcal{B}$ and for every $S \subseteq \mathcal{B} - N_G(B)$, we have*

$$P_\Omega(B \mid \bigcap_{B' \in S} \overline{B}') \leq P_\Omega(B)$$

*To state this in words: conditioning on* avoiding *any set of non-neighbors of $B$, does not increase the probability of $B$.*

THEOREM 1.4 ([**35**]). *Suppose that $\Omega$ is a probability space and $\mathcal{B}$ is a set of events in that space. Suppose that $G$ is a lopsidependency graph for $\mathcal{B}$. And suppose that each $B \in \mathcal{B}$ satisfies $P_\Omega(B) \leq p$ and $|N_G(B)| \leq d$. Furthermore suppose that $epd \leq 1$. Then we have $P(\bigcap_{B \in \mathcal{B}} \overline{B}) > (1 - 1/d)^m > 0$.*

It is immediate that all dependency graphs are lopsidependency graphs so that the LLLL is always stronger than the LLL. The LLL is merely a slightly special case of the LLLL, and thus it seems a little strange that one should need to give the two principles different names. However, the variable assignment LLL is especially important in combinatorics, and there are many algorithmic complications in dealing with the types of probability spaces covered by the LLLL. Thus, it will be useful for us to distinguish between cases in which the LLL applies, and those more exotic cases in which the the LLLL is needed.

In [**76**], a rather exhaustive list of probability spaces covered by the LLLL is given; many of these are very obscure, with few applications in combinatorics. There are two settings which are somewhat useful: the space of random permutations, and the variable-assignment setting in which the bad-events are atomic.

The case of random permutations was in fact the original motivation for the LLLL, and it was introduced in [**35**] to show the existence of Latin transversals for certain types of arrays. The Latin transversal is another example which will come back to frequently, so we briefly describe it here. (See Section 7.0.1 for much more information

about this problem.) We are given an $n \times n$ matrix $A$, in which the entries of $A$ are colors. Our goal is to select $n$ cells, one from each row and column, such that no two cells receive the same color.

PROPOSITION 1.5 ([**35**]). *Suppose each color appears at most $\Delta \leq n/(4e)$ times in the matrix $A$. Then $A$ has a Latin transversal.*

PROOF. One can view the Latin transversal as equivalent to selecting a permutation $\pi \in S_n$ uniformly at random, and then choosing cell $(i, \pi(i))$ from each row. In this case, a bad-event is the event that there are two selected cells with the same color, i.e. $\pi(x_1) = y_1 \wedge \pi(x_2) = y_2$ where $A(x_1, y_1) = A(x_2, y_2)$. Each such event has probability $p = \frac{1}{n(n-1)}$.

As shown in [**35**], one can form a lopsidependency graph among such bad-events, such that events are connected iff they overlap in a row or column; for example, the event $\pi(x_1) = y_1 \wedge \pi(x_2) = y_2$ overlaps with $\pi(x_1) = y_1' \wedge \pi(x_2') = y_2'$. One can verify that each bad-event has at most $4n(\Delta - 1)$ neighbors in the lopsidepenency graph. Now apply the LLLL. □

In [**77**], this setting was formalized and slightly extended, to allow one to select multiple permutations $\pi_1, \ldots, \pi_N$ on $[n_1], \ldots, [n_N]$ letters, each permutation chosen independently and uniformly. Bad-events all have the form $B \equiv \pi_{i_1}(x_1) = y_1 \wedge \ldots \pi_{i_k}(x_k) = y_k$; that is, they are all a conjunction of conditions on the values of specific elements of the permutations. We refer to such bad-events as *atomic events*.

PROPOSITION 1.6 ([**35**], [**77**]). *Suppose $\mathcal{B}$ is a set of a atomic events in $\pi$ and the probability space $\Omega$ is defined by selecting $\pi_1, \ldots, \pi_N$ uniformly at random. Then, define the following graph $G$ on bad-events; we include an event from $B_1$ to $B_2$ iff either of the following two conditions holds:*

(1) *$B_1$ demands $\pi_i(x) = y$ and $B_2$ demands $\pi_i(x) = y'$, for some $i \in [N]$ and $x, y \neq y'$.*

(2) $B_1$ *demands* $\pi_i(x) = y$ *and* $B_2$ *demands* $\pi_i(x') = y$, *for some* $i \in [N]$ *and* $x \neq x', y$.

*Then* $G$ *is a lopsidependency graph for* $\mathcal{B}, \Omega$.

*(Note: recall that* $B \equiv \pi_{i_1} i(x_1) = y_1 \wedge \pi_{i_k} i(x_k) = y_k$*; we say that* $B$ demands $\pi_i(x) = y$ *if* $x_j = x, y_j = y, i_j = i$ *for some* $j \in [k]$.*)*

To explain the intuition behind Proposition 1.6: suppose $B_1$ demands $\pi(1) = 1$ and $B_2$ demands $\pi(2) = 2$. So, there would be no edge from $B_1$ to $B_2$ in $G$. Furthermore, $B_1$ and $B_2$ are positively correlated; for initially the probability of the event $B_1$ is $1/n$. However, if $B_2$ occurs, then $\pi(2) = 2$ and there are only $n-1$ remaining possible values for $\pi(1)$. So, conditional on $B_2$ occurring, the probability of $B_1$ has *increased*.

We will examine the case of random permutations in much more detail in Chapter 7.

Another useful and much more simple application for the LLLL was introduced by [**83**] for a construction involving hypergraph coloring.

PROPOSITION 1.7 ([**83**]). *Suppose* $\mathcal{B}$ *is a set of a atomic events and the probability space* $\Omega$ *is defined by selecting value for each variable* $x_i$ *independently with probability distribution* $p_i$*. Then, define the following graph* $G$ *on bad-events; we include an edge from* $B_1$ *to* $B_2$ *iff, for some* $i$*, we have that* $B_1$ *demands* $X_i = j$ *and* $B_2$ *demands* $X_i = j'$ *and* $j \neq j'$*. In this case, we say that* $B_1, B_2$ disagree *on variable* $x_i$.

*Then* $G$ *is a lopsidependency graph for* $\mathcal{B}, \Omega$.

*(Note that, without the condition that* $j \neq j'$*, we would have precisely the canonical* dependency *graph for the LLL.)*

We briefly discuss here how to apply this to $k$-SAT. Suppose we have a $k$-SAT instance in which variable appears in at most $L$ clauses (either positively or negatively). How large may $L$ become in order to guarantee that the instance is satisfiable?

PROPOSITION 1.8 ([**43**]). *Suppose that each variable appears at most* $L = 2^{k+1}e(k+1)$ *times. Then the* $k$*-SAT instance is satisfiable.*

PROOF. We will sketch the proof here; a much more detailed discussion of this problem, including an improved bound, will be given in Chapters 4 and 8.

Assume that each variable appears in exactly $L/2$ clauses positively and $L/2$ clauses negatively. (This is in fact the worst-case distribution, as shown in [**43**]; the proof of this fact is somewhat technically involved.) Then, we assign each variable to be True or False with probability 1/2. A bad event is that a clause is falsified; this has probability $p = 2^{-k}$.

Now, consider the dependency of a bad-event $B$. For of the $k$ variables in $B$, there are $L/2$ clauses which *disagree* with $B$ on the variable. So, $B$ is dependent with at most $kL/2 + 1$ clauses (including itself).

Now apply the symmetric LLL criterion

$$e(2^{-k})(kL/2 + 1) \leq 1$$

□

In fact, as shown in [**43**], this bound on $L$ is asymptotically tight; for any $c > 1$ and $k$ sufficiently large, there exist unsatisfiable $k$-SAT instances in which variables appears $c\frac{2^{k+1}}{e(k+1)}$ times. Thus, this problem is a simple example where the LLLL provided asymptotically tight bounds. (Note that, using just the ordinary LLL, one would have only been able to show the bound for $L = \frac{2^k}{e(k+1)}$).

### 1.4. The Asymmetric LLL and other LLL criteria

The LLL, as we have stated it, depends on only two parameters: the probabilities of the bad-events $P_\Omega(B)$; and a dependency graph $G$ for $\mathcal{B}$. The symmetric LLL gives a very simple criterion, which is boiled down to two scalar parameters: the maximum probability $p = \max P_\Omega(B)$ and the maximum dependency $d = \max |N_G(B)|$. A natural question is whether one can give other types of conditions on the probabilities $\{P_\Omega(B)\}$ and the dependency graph $G$.

In [**104**], a criterion referred to as the "Asymmetric LLL" was given which allows us to take into account the fact that different bad-events may have different probabilities and different neighborhood sizes. We will state this criterion in a somewhat non-standard form, for reasons which will be important later in this thesis.

THEOREM 1.9 ([**104**]). *Suppose there is some lopsidependency graph $G$ for $\Omega$ and suppose there is a weighting function $\mu : \mathcal{B} \to [0, \infty)$ which satisfies the following property: for all $B \in \mathcal{B}$ we have*

$$\mu(B) \geq P_\Omega(B) \prod_{B' \sim B} (1 + \mu(B'))$$

*Then the event $\cap_{B \in \mathcal{B}} \overline{B}$ has positive probability.*

*(Recall our convention that $B \sim B'$ means that $B, B'$ are neighbors in $G$. Also recall that $B \sim B$.)*

The symmetric LLL follows from this as an easy corollary: simply set $\mu(B) = ep$ for all $B \in \mathcal{B}$. The power of Theorem 1.9 comes from the fact that bad-events which have a low value of $P_\Omega(B)$ may have a large number of neighbors, and vice-versa.

In [**17**], an even stronger form of the LLL, referred to as the *cluster-expansion* criterion, was given. This can be stated as:

THEOREM 1.10 ([**17**]). *Suppose there is some lopsidependency graph $G$ for $\Omega$ and suppose there is a weighting function $\mu : \mathcal{B} \to [0, \infty)$ which satisfies the following property: for all $B \in \mathcal{B}$ we have*

$$\mu(B) \geq P_\Omega(B) \Big( \sum_{\substack{I \subseteq N_G(B) \\ I \text{ independent in } G}} \prod_{B' \in I} \mu(B') \Big)$$

*Then the event $\cap_{B \in \mathcal{B}} \overline{B}$ has positive probability.*

Observe in the formulation of Theorem 1.10 that $N_G(B)$ is the inclusive neighborhood of $B$. It is clear that there is exactly one independent set $I$ which contains $B$ itself, namely $I = \{B\}$.

Since we will be dealing frequently with the cluster-expansion criterion, the following notation is convenient. For any bad-event $B \in \mathcal{B}$, we define

$$\theta_G(B) = \sum_{\substack{I \subseteq N_G(B) \\ I \text{ independent}}} \prod_{B' \in E} \mu(B')$$

Thus, we can state the cluster-expansion criterion more simply as

$$\mu(B) \geq P_\Omega(B)\theta_G(B);$$

we can think of $\theta_G(B)$ as the "inflation factor" compared to the original probability distribution. (We will sometimes write $\theta(B)$, if $G$ is clear from context.)

We note one strange fact about the cluster-expansion criterion: it is not necessarily monotone in $G$. Thus, it may be that $G$ is a lopsidependency graph for $\Omega$, and $G$ does not satisfy the cluster-expansion criterion, but $G' \supseteq G$ (which is necessarily a lopsidependency graph) does.

We will give a brief application of the cluster-expansion criterion to independent transversals. We include this here because it illustrates a few proof-techniques which will frequently appear in our analysis.

PROPOSITION 1.11 ([**17**]). *Suppose that $G$ has a partition of the vertices into blocks of size $b$, and the average degree of each block is $d$. Then, if $b \geq 4d$, there exists an independent transversal of $G$.*

PROOF. We select each vertex with probability $1/b$. We have a bad-event for each edge, that both end-points are selected. We define the weighting function by setting $\mu(B) = \alpha$ for all $B$; here $\alpha > 0$ is a parameter to be specified.

Now, let us fix some bad-event $B$ defined by an edge $f = \langle u, v \rangle$. We need to describe the neighborhood structure of $B$. Any edge $f'$ which is affected by $f$ has one end-point in the block of $u$ or one end-point in the block of $v$. However, note that if $f', f''$ both have an end-point in the block of $u$, then the corresponding bad-events are connected. Hence, in an *independent* set of bad-events which neighbor $f$, there

may be *at most one* edge $f'$ in the block of $u$ and *at most one* edge $f''$ in the block of $v$.

We want to compute $\sum_{I \subseteq N_G(B)} \prod_{B' \in I} \mu(B')$, where the sum is taken over independent sets. We set aside the special case of $I = \{B\}$. In any other independent set $I$, we may select one or zero edges $f'$ in the block of $u$, and we may select one or zero edges $f''$ in the block of $v$. If we select an edge $f'$, we multiply in a factor of $\mu(f') = \alpha$; if we select zero edges $f'$, we multiply in a factor of 1. We do a similar process for $f''$.

This type of summation will occur frequently in analyzing the LLL: we are given lists $L_1, \ldots, L_l$, and we form a set $I$ by selecting one element each list. For each resulting set $I$, we multiply together terms of the form $f(i, j)$, where $i$ is the item selected from list $j \in [l]$. We then sum this product over all possible choices for $I$.

Whenever we encounter this type of summation, we use the following useful identity:

$$\sum_{i_1 \in L_1, \ldots, i_l \in L_l} f(i_1, 1) \times \cdots \times f(i_l, l) = \prod_{j=1}^{l} \sum_{i \in L_j} f(i, j)$$

In our case, we have two lists corresponding to the two blocks. The items in the list are the $\leq bd-1$ edges (apart from $f$ itself), as well as the null item (corresponding to selecting no edge from that block). The former terms have value $f(i, j) = \alpha$; the latter term has value $f(i, j) = 1$.

Adding in the case of $I = \{B\}$, which contributes $\mu(B) = \alpha$, we thus compute

$$\sum_{\substack{I \subseteq N_G(B) \\ I \text{ independent in } G}} \prod_{B' \in I} \mu(B') \leq \alpha + (1 + (bd-1)\alpha) \times (1 + (bd-1)\alpha)$$

and so the cluster-expansion criterion reduces to

$$\alpha \geq (1/b^2)(\alpha + (1 + (bd-1)\alpha)^2) \tag{1}$$

We will frequently encounter inequalities that are similar to (1). While (1) is simple enough to solve directly via the quadratic formula, in other cases we will

encounter polynomials of higher degree. We will solve (1) by using a maneuver which, although overkill in this situation, will be useful later on. First, we arrange this expression:

$$\alpha - (1/b^2)(\alpha + (1 + (bd-1)\alpha)^2) \geq 0 \tag{2}$$

We want to show that this has a positive root $\alpha > 0$. Now, observe that a necessary and sufficient condition for this is that if we maximize the LHS of (2) over all $\alpha > 0$, the resulting maximum value is positive. We thus differentiate with respect to $\alpha$ and set the resulting expression equal to zero, finding the critical point $\alpha_0$:

$$\alpha_0 = \frac{1 + b^2 - 2bd}{2(bd-1)^2}$$

A necessary and sufficient condition for (2) to be satisfiable if it is satisfied at $\alpha = \alpha_0$. Plugging $\alpha = \alpha_0$ into (2) yields

$$\frac{(b^2-1)(3 + b^2 - 4bd)}{4b^2(bd-1)^2} \geq 0$$

which is solvable iff $b \geq 2d + \sqrt{4d^2 - 3}$; this holds for $b \geq 4d$.

□

### 1.5. Shearer's criterion

The cluster-expansion form of the LLL has perhaps the best balance between strength and ease of calculations. However, it is not the strongest form of the LLL. In [**102**], Shearer gave the strongest possible criterion that can be stated in terms of just $G$ and $P_\Omega(B)$ alone. This criterion is stated in terms of the *independent set polynomial*.

DEFINITION 1.12. *Suppose that $G$ is a given graph on vertex set $V$ and $\vec{p} \in [0,1]^V$. We define the* independent set polynomial *with respect to any $I \subseteq V$ as*

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

*Note that $Q(I, G, p) = 0$ if $I$ is not an independent set.*

THEOREM 1.13 ([**102**]). *Suppose that $G$ is a simple undirected graph with vertex set $\{1, \ldots, m\}$, and $\vec{p} \in [0,1]^m$. Then:*

(1) *Suppose that for all sets $I \subseteq [m]$ which are independent with respect to $G$, we have $Q(I, G, p) > 0$. Then, for* any *probability space $\Omega$ and any set of events $\mathcal{B} = \{B_1, \ldots, B_m\}$ satisfying $P(B_i) = p_i$ and such that $G$ is a lopsidependency graph for $\mathcal{B}$, there is a positive probability that none of the events in $\mathcal{B}$ occur.*

(2) *Suppose that there is some independent set $I \subseteq [m]$ such that $Q(I, G, p) \leq 0$. Then, there exists a probability space $\Omega$ and any set of events $\mathcal{B} = \{B_1, \ldots, B_m\}$ satisfying $P(B_i) = p_i$ such that $G$ is a dependency graph for $\mathcal{B}$ and such that, with probability one, at least one of the events in $\mathcal{B}$ occurs.*

*If $G, p$ satisfy the condition $Q(I, G, p) > 0$ for all independent $I$, we say that they* satisfy the Shearer criterion*; otherwise we say they* violate *the Shearer criterion.*

We list some useful monotonicity properties of the Shearer criterion:

PROPOSITION 1.14 ([**102**]). *Suppose that $p(B) \leq p'(B)$ for all $B \in \mathcal{B}$. (We write this more compactly as $p \leq p'$.) Then if $G, p'$ satisfies the Shearer criterion, so does $G, p$.*

PROPOSITION 1.15 ([**102**]). *Suppose that the edges of $G'$ are a subset of the edges of $G$. Then if $G, p$ satisfies the Shearer criterion, so does $G', p$.*

Although the Shearer criterion is the most powerful LLL criterion, it is extremely difficult to work with mathematically. The main reason for this is that the independent-set polynomials are alternating sums; thus, if we want to show that $Q(I, B, P_\Omega) > 0$

for some $I$, then individual terms $P_\Omega$ will appear both negatively and positively. Thus, if one has an upper bound on $P_\Omega(B)$ (which is all that is needed for the LLL), then one cannot directly use this to bound $Q(I, B, P_\Omega)$

Even showing very simple properties of the Shearer criterion become technically challenging. Consider Proposition 1.15, for example. Although it is "merely" an algebraic property of the independent set polynomials, it is difficult to show directly. The simplest proof is the following indirect argument: for any $\mathcal{B}, \Omega$ of which $G'$ is a lopsidependency graph, so is $G$. So by Theorem 1.13, $\Omega$ avoids all the bad-events $\mathcal{B}$. Since this is true for arbitrary $\mathcal{B}, \Omega$ with lopsidependency graph $G'$, by Theorem 1.13 $G', p$ satisfies the Shearer criterion. This convoluted proof uses the very powerful Theorem 1.13 twice, but it is much easier than trying to compute $Q(I, G', p)$ in terms of $Q(I, G, p)$.

Similarly, Theorems 1.9, 1.10 can be derived as consequences of Theorem 1.13; it would not be correct to call them corollaries (even though the derivation is merely algebraic) because it is technically much more difficult to prove them this way than to prove them directly.

For these reasons, one rarely uses the Shearer criterion in combinatorics.

## 1.6. A variable-based LLL criterion

When using the cluster-expansion criterion, one must specify a value $\mu(B)$ for every bad-event $B$, and one must enumerate the collection of independent sets of neighbors for each $B$. When the bad-events are relatively homogeneous (falling into just one or a few categories), then this can be relatively easy. However, when the set of bad-events is more diverse, then this can require specifying an excessively large number of parameters.

In this section, we reduce the number of parameters dramatically by rephrasing the LLL criterion to give a constraint in terms of each *variable.* Suppose that we are in the variable-assignment LLL setting, and that each bad-event is atomic, i.e.

it is a conjunction of terms of the form $X_i = j$. (In the variable-assignment LLL setting, any bad-event can be written as the disjoint union of a finite number of such atomic events.) We often identify a bad-event $B$ with the set of atomic conditions it demands; that is, we write $B = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ to mean that $B$ is true iff $X_{i_1} = j_1, \ldots, X_{i_k} = j_k$.

Instead of specifying a vector of probabilities $p_{ij}$ to define the space $\Omega$ and a weighting function $\mu$ for the bad-events, we instead suppose we are given an assignment of non-negative real numbers $\lambda_{i,j}$, for each variable $X_i$ and each possible value of the variable $j$. We also define $\lambda_i = \sum_j \lambda_{i,j}$. The vector $\lambda$ should be thought of as an "inflated" version of the probability vector $p$; roughly speaking, $\lambda_{i,j}$ is the probability that $X_i = j$ conditional on avoiding the bad-events. We will derive a probability distribution $p$ and a weighting function $\mu$ from this vector to satisfy the cluster-expansion criterion.

THEOREM 1.16. *Suppose that for each variable $i \in [n]$ we satisfy*

$$\lambda_i - \prod_{B: i \in S_{B'}} \prod_{i', j') \in B} \lambda_{i',j'} \geq 1$$

*Then there is a vector of probabilities $p$ which satisfies the LLL*

PROOF. Set $p_{i,j} = \frac{\lambda_{i,j}}{\lambda_i}$. Define the weighting function $\mu : B \to [0, \infty)$ by $\mu(B) = \prod_{(i,j) \in B} \lambda_{i,j}$.

Clearly this is a valid probability distribution. We now must show that it satisfies the LLL criterion. Consider some $B \in \mathcal{B}$. To compute $\theta(B)$, observe that for any variable $i$ involved in $B$, an independent set of neighbors of $B$ contains one or zero bad-events $B'$ affected by $i$. Thus, we have:

$$P_\Omega(B)\theta_G(B) \leq P_\Omega(B) \prod_{(i,j) \in B} (1 + \sum_{B': i \in S_{B'}} \mu(B'))$$

$$= \prod_{(i,j) \in B} \frac{\lambda_{i,j}}{\lambda_i} \prod_{(i,j) \in B} (1 + \sum_{B': i \in S_{B'}} \prod_{(i',j') \in B'} \lambda_{i',j'}$$

$$\leq \prod_{(i,j)\in B} \frac{\lambda_{i,j}}{\lambda_i} \prod_{(i,j)\in B} \lambda_i$$

by hypothesis

$$\leq \prod_{(i,j)\in B} \lambda_{i,j} = \mu(B)$$

□

This type of accounting gives a simplified and slightly weakened form of the cluster-expansion criterion. We illustrate with independent transversals, for which we used the cluster-expansion LLL directly in Proposition 1.11.

PROPOSITION 1.17 ([**17**]). *Suppose that $G$ has a partition of the vertices into blocks of size $b$, and the average degree of each block is $d$. Then, if $b \geq 4d$, there exists an independent transversal of $G$.*

PROOF. We set $\lambda_{i,j} = \alpha$, where $\alpha$ is a scalar to be determined. Now, for each variable (block) $i$, there are $b$ values for the variable and $bd$ events involving that variable which are atomic events on exactly two variables hence we have

$$\begin{aligned}\sum_j \lambda_{i,j} - \sum_{B: i\in S_{B'}} \prod_{(i',j')\in B} \lambda_{i',j'} &= b\alpha - bd\alpha^2 \\ &= \frac{b}{4d} \qquad \text{for } \alpha = \frac{1}{2d} \\ &\geq 1 \qquad \text{for } b \geq 4d\end{aligned}$$

Observe that this derivation is much simpler, both conceptually and algebraically, than the proof given in Proposition 1.11 □

## 1.7. The iterated LLL

One powerful technique is based on multiple applications of the LLL; this is often referred to as the *iterated LLL* or the *semi-random method*. The basic idea is that instead of completely fixing the combinatorial structure, we determine it only partially.

For example, if we are selecting an independent transversal, instead of immediately choosing every vertex to either go into the transversal or not go into the transversal, we might only make this decision for a small random selection of the vertices — most of the remaining vertices remain undetermined. Every time we make this partial decision, some of the combinatorial structure can be "frozen" and resulting dependencies disappear. Thus, at each stage of this process, we are decreasing the dependency parameter $d$ in the LLL.

One powerful example of this technique appears in the seminal construction of [**73**] for universal packet routing; we will encounter this construction in much greater depth in Section 5.5. That example features multiple (unbounded) number of applications of the LLL. See [**85**] for several other applications of this technique. In this section, we will give a much simpler example of this principle. This is characterized by only two applications of the LLL.

## 1.8. An application of the iterated LLL: near-optimal coloring of shift hypergraphs

One of the first applications of the LLL is in fact an affirmative answer to an *infinitary* question of Strauss: for a given $k$, does there exist a finite $m$ such that for any set $S$ of $m$ integers, there is a $k$-coloring of the integers such that every integer translate of $S$ (i.e., sets of the form $S + t$, for $t \in \mathbf{Z}$) meets every color class? We let $m(k)$ denote the smallest such value of $m$, if it exists.

By combining the LLL with a compactness argument, it was shown in [**34**] that $m(k) \leq (3 + o(1))k \ln k$. Following this, the work of [**4**] showed, among other things, that $m(k) \geq (1 - o(1))k \ln k$, and also presented an "efficient" version of the upper bound, by showing that the required coloring can in fact be made periodic with a short period. Answering one of the main open questions of [**4**], we prove in this section that $m(k) \leq (1 + o(1))k \ln k$ (ours is also an efficiently-computable periodic coloring as in [**4**]). Our approach is very similar to that of [**38**].

We follow the approach of [**4**] and reduce the problem to a certain hypergraph-coloring problem: how small an $m = m(k)$ can we exhibit, so that for every $m$-uniform, $m$-regular hypergraph $H$ there exists a $k$-coloring of the vertices such that every edge meets every color class. (Briefly, each vertex corresponds to an integer; every edge corresponds to a translation of $S$.) Thus, we use this hypergraph-coloring terminology from now on. A short calculation using the LLL shows that if $m = (3 + o(1))k \ln k$, then there is a positive probability that a random coloring causes every edge to meet every color class [**4**].

THEOREM 1.18. *Suppose $m \geq (1 + \epsilon)k \ln k$, where $\epsilon(k) \geq (4 + \omega(1)) \ln^{-1/2} k$; and suppose $k$ is sufficiently large. Then, the vertices of any $m$-uniform, $m$-regular hypergraph can be colored using $k$ colors, such that each edge meets every color class. Furthermore, such a coloring can be found in randomized polynomial time.*

We assume that $\epsilon(k) = (4 + v) \ln^{-1/2} k$, where $v > 0$ is some fixed constant. We assume that $k$ is sufficiently large, which may depend on $v$. Finally, we ignore all rounding effects; in this vein, we suppose $m = (1 + \epsilon)k \ln k$ exactly.

**1.8.1. Phase I.** In Phase I, we choose a coloring using $k' = k/\ln k$ colors; each vertex receives each color uniformly at random. On average, each edge $f$ receives each color an average of $\mu = (1 + \epsilon) \ln^2 k$ times.

For each edge $f$ and each color $c$, we have a bad-event that $f$ receives the color more than $m_1 = \mu(1 + \delta)$ times or less than $m_0 = \mu(1 - \delta)$ times, where $\delta = 4/\sqrt{\ln k}$. For $k$ sufficiently large, we have $\delta < 1$ and the probability of this event can be estimated by the Chernoff bound; it is at most $p \leq 2e^{-\mu\delta^2/3} \leq 2k^{-16/3(1+\epsilon)}$. Similarly, each bad-event $c, f$ depends on other bad-events $c', f'$ iff the edges $f, f'$ intersect; hence the dependency of a bad-event is at most $d \leq k' \times m \times m \leq (1+\epsilon)^2 k^3 \ln k$. For $k$ sufficiently large the LLL criterion is

$$e \times 2k^{-16/3(1+\epsilon)} \times (k^3 \ln k(1 + \epsilon)^2) \leq 1$$

Simple calculus now shows that, when $k$ is sufficiently large, this holds for all $\epsilon > 0$. Note that in this phase, we are not taking advantage of the $\epsilon$-slack in our estimate for $m$.

**1.8.2. Phase II.** In the second phase of the construction, we subdivide each of the initial colorings from Phase I into $\ln k$ sub-colors. The total number of colors thus produced is $k' \times \ln k = k$ as desired. The critical property here is that distinct colorings from Phase I no longer affect each other in any way. This greatly reduces the dependency when applying the Lovász Local Lemma.

Now consider an edge $f$ and a color $c$ (the color $c$ includes both the coloring from Phase I and Phase II): a bad event is that $f$ does not see the color $c$. The probability of this event can be computed as follows. The edge sees the Phase-I color corresponding to $c$ at least $m_0$ times. The total probability that none of appearances is equal to $c$, is at most $p \leq (1 - k'/k)^{m_0}$.

Next, consider the dependency of an event. Again, each event $c, f$ affects $c', f'$ iff $c, c'$ have a common Phase-I color *and* $f, f'$ intersect in some vertex which shares this Phase-I color. As each Phase-I color appears at most $m_1$ times in $f$, the total dependency is thus at most $d \leq (k/k')m_1 m$.

The LLL criterion is thus satisfied if $e(1 - k'/k)^{m_0}((k/k')m_1 m) \leq 1$. Routine calculations show that this is satisfied for $k$ sufficiently large if $\epsilon \geq (4 + o(1)) \ln^{-1/2} k$.

CHAPTER 2

# The Moser-Tardos algorithm

## 2.1. Basic algorithm

The Lovász Local Lemma can be used in a variety of situations to show that certain combinatorial configurations exist. It also gives a simple random process for producing these configurations: namely, draw a sample from $\Omega$. However, the probability that $\Omega$ has the desired properties, while non-zero, is typically exponentially small. Thus, while the LLL gives existential proofs, it does not lead directly to efficient algorithms.

For many years since the introduction of the LLL, there was a major gap between constructive and non-constructive results. A variety of algorithms, some quite complicated and ad-hoc, were introduced to bridge this gap. These algorithms often showed qualitatively similar results to the LLL (in particular, the probability of a bad-event would be compared to the number of bad-events), although constant factors were much worse. This line of research culminated with the work of Moser & Tardos [**87**], giving an amazing simple construction to turn nearly all applications of the LLL into polynomial-time algorithms. We refer to this as the *Moser-Tardos Algorithm (or MT algorithm)*. It can be summarized as follows:

1. Draw $X_1, \dots, X_n \sim \Omega$.
2. While there is some true bad-event $B \in \mathcal{B}$:
    3. Select some true $B \in \mathcal{B}$ arbitrarily.
    4. For each variable $i \in S_B$, draw $X_i$ independently from its distribution under $\Omega$.

If this algorithm terminates, then it produces a configuration which necessarily avoids all bad-events. We refer to step (4), in which all the variables involved in $\mathcal{B}$ are drawn independently from the distribution $\Omega$, as *resampling* $B$.

We note that this algorithm, phrased in terms of variables, is inherently specific to the variable-assignment LLL setting. It is not clear what it would even mean to give a constructive algorithm for the full LLL, which is phrased in terms of probability spaces.

Also, we note that this algorithm is somewhat under-specified, in that if there are multiple bad-events which are currently true then one is allowed to resample *any* of them. This freedom can be useful in a variety of contexts. We refer to the choice of which bad-event to resample as the *resampling rule*.

In [**87**], it is shown that this algorithm terminates under the same conditions as the asymmetric LLL:

THEOREM 2.1 ([**87**]). *Suppose there is a weighting function $\mu : \mathcal{B} \to [0, \infty)$ which satisfies the following property: for all $B \in \mathcal{B}$ we have*

$$\mu(B) \geq P_\Omega(B) \prod_{B' \sim B} (1 + \mu(B'))$$

*Here, we write $B \sim B'$ to denote that $B$ and $B'$ are either equal to each other or are connected in the canonical dependency graph, viz. $S_B \cap S_{B'} \neq \emptyset$.*

*Then, the MT algorithm terminates with probability one; the expected number of resamplings it performs is at most $\sum_{B \in \mathcal{B}} \mu(B)$.*

Aside from the restriction to the canonical dependency graph, this is precisely the same as Proposition 1.9, which gives conditions for there to exist positive probability of avoiding $\mathcal{B}$.

In the MT setting, we always work with the canonical dependency graph for the events, which has an edge between $B$ and $B'$ if $B$ and $B'$ depend on common variables. It is useful to develop notation to this variable-intersection setting. Namely, for *any* events $E, E'$ which are defined on subsets of the variables, we write $E \sim E'$ if they share a common variable. In this case, note that $B \sim B$. Given a set of bad-events

$\mathcal{B}$, and any event $E$ (not necessarily a bad-event), we can write

$$N(E) = \{B \in \mathcal{B} \mid B \sim E\} \tag{3}$$

Note that, for a bad-event $B$, that $N(B)$ in the sense of (3) coincides with the definition of $N_G(B)$ we have given where $G$ is the canonical dependency graph.

Setting $\mu(B) = ep$, we obtain the following simple symmetric criterion:

PROPOSITION 2.2 ([**87**]). *Suppose that for each $B \in \mathcal{B}$ we have $P_\Omega(B) \leq p$ and $|N(B)| \leq d$ and suppose that $epd \leq 1$. Then the MT algorithm terminates with probability one; the expected number of resamplings it performs is at most $epm$.*

There are two probability spaces at play when we we are analyzing the MT algorithm. First, there is the space $\Omega$; this is the space in which the variables are drawn independently. Second, there is the "ambient" probability space in which we draw the random bits used to drive the MT algorithm itself. In this second probability space, which does not receive any specific name, the variables $X_i$ are not independent. We will always refer to probabilities pertaining to the first space as $P_\Omega$ while the probabilities from the second space are referred to simply as $P$. Thus, for example, when we refer to the expected number of resamplings performed by the MT algorithm, this expectation is taken with respect to the ambient probability space. When we refer to the probabilities of the bad-events, we refer to their probabilities within the space $\Omega$. We can think of the MT algorithm as "simulating" $\Omega$.

We have seen that there are stronger criteria than the asymmetric LLL. Later papers have shown that the MT algorithm meets these as well. First, in [**94**], Pegden gave a straightforward proof that the MT algorithm matches the cluster-expansion criterion:

THEOREM 2.3 ([**94**]). *Suppose there is a weighting function $\mu : \mathcal{B} \to [0, \infty)$ which satisfies the following property: for all $B \in \mathcal{B}$ we have*

$$\mu(B) \geq P_\Omega(B)P_\Omega(B)\Big( \sum_{\substack{I \subseteq N_G(B) \\ I \text{ independent} \\ \text{in the canonical } G}} \prod_{B' \in I} \mu(B')\Big)$$

*Then, the MT algorithm terminates with probability one; the expected number of resamplings it performs is at most $\sum_{B \in \mathcal{B}} \mu(B)$.*

In [**65**], Kolipaka & Szegedy extended this to show that the MT algorithm matches, essentially, the full Shearer criterion:

DEFINITION 2.4 (The measure of a bad-event in the Shearer setting). *Let $G$ denote the canonical dependency graph for $\Omega, \mathcal{B}$. Suppose that $P_\Omega, G$ satisfies the Shearer criterion. For any bad-event $B$, define the* measure *of $B$ to be $\tilde{\mu}(B) = \frac{Q(G,\{B\},P_\Omega)}{Q(G,\emptyset,P_\Omega)}$, where $G$ is the canonical dependency graph.*

THEOREM 2.5 ([**65**]). *Let $G$ denote the canonical dependency graph for $\Omega, \mathcal{B}$. Suppose that $P_\Omega, G$ satisfies the Shearer criterion.*

*Then the MT algorithm terminates with probability one; the expected number of resamplings it performs is at most $\sum_{B \in \mathcal{B}} \tilde{\mu}(B)$.*

Harvey & Vondrak also showed the connection between the Shearer criterion and the cluster-expansion LLL criterion:

THEOREM 2.6 ([**51**]). *Suppose that a weighting function $\mu$ satisfies the cluster-expansion LLL criterion for the canonical dependency graph $G$. Then for all $B \in \mathcal{B}$ we have $\tilde{\mu}(B) \leq \mu(B)$.*

**2.1.1. Proof of the MT algorithm.** Moser & Tardos give a well-written description for their algorithm; we recommend reading [**87**] for more details. In this section, we will give a self-contained proof which has a few key differences from [**87**].

We will prove Theorem 2.3 in this section; the stronger Theorem 2.5 is beyond our scope here.

To begin, suppose we execute the MT algorithm. It may terminate after $T$ steps, or it may proceed for an infinite time. At each time-step $t$, we resample some bad-event $B_t$. We refer to the listing $B_1, B_2, \ldots$, as the *execution log*. The bad-events $B_1, B_2, \ldots$ are not necessarily distinct.

The basic analytical tool for understanding the MT algorithm is the *witness tree.* This is a compact description of "why" each $B_t$ was resampled; more precisely, it provides a history of all the variables that are relevant to this resampling. There is a separate witness tree for each time $T$, which we denote $\hat{\tau}^T$.

We form $\hat{\tau}^T$ recursively, according to the following rule. We begin with $\hat{\tau}^T_{\geq T}$, which has a singleton node labeled by $B_T$. We next go backward in time through $t = T-1, \ldots, 1$. For each $B_t$, we see if there is some node $v \in \hat{\tau}^T_{\geq t+1}$ labeled by some $B' \sim B_t$. If there is no such node, then we do not update $\hat{\tau}^T$, thus $\hat{\tau}^T_{\geq t} = \hat{\tau}^T_{\geq t+1}$. If there are such nodes $v \in \hat{\tau}^T_{\geq t+1}$, then we select one vertex $v \in \hat{\tau}^T_{\geq t+1}$ *of greatest possible depth* (breaking ties arbitrarily), and we form $\hat{\tau}^T_{\geq t}$ by adding a single node as a child of $v$, labeled by $B_t$. At the end of this process, we set $\hat{\tau}^T = \hat{\tau}^T_{\geq 1}$.

We distinguish now between two closely related senses of the term "witness tree." First, there is the random variable $\hat{\tau}^T$, as we have defined it above; it is a random variable. Second, one might refer to a specific rooted tree, whose nodes are labeled by bad-events; these are the possible values that the random variables $\hat{\tau}$ can take. We refer to the latter, which are ordinary (not random) variables, as *tree-structures* and denote them by $\tau$. Thus, we might fix some tree-structure and ask whether it is the case that $\hat{\tau}^T = \tau$.

In many discussion of witness trees, the precise value of $T$ is irrelevant. Thus, we will often omit the superscript. When we write $\hat{\tau}$, then we mean $\hat{\tau}^T$ for some unspecified $T$.

When discussing these trees, we often say that the root of the tree is at the "top" and the deep layers of the tree are at the "bottom". The top of the tree corresponds to later events, the bottom of the tree to the earliest events.

We note one important property of the witness produced in this fashion:

PROPOSITION 2.7. *In any witness tree $\hat{\tau}^T$, there cannot be two nodes $v, v'$ at the same depth, labeled by $B, B'$, such that $B \sim B'$. In other words, the layers of $\hat{\tau}^T$ receive labels which are* distinct *and* independent *with respect to the canonical dependency graph.*

PROOF. We prove that this holds for $\hat{\tau}^T_{\geq j}$ by induction on $j$. When $j = T$ this is clear. Now suppose that when forming $\hat{\tau}_{\geq j}$ we add a node $v$ labeled by $B$, and there is already another node $v'$ labeled by $B'$ which is at depth $k$. In this case, $v$ would be eligible to be placed as a child of $v'$. As the rule for forming witness trees always places nodes at the greatest possible depth, this implies that $v$ is placed at depth $\geq k + 1$. In particular, $\hat{\tau}_{\geq j}$ does not contain $B, B'$ at the same depth $k$. □

By convention, we will only be interested in tree-structures which are possible values of $\hat{\tau}$. Thus, we may assume that all tree-structures $\tau$ have the property that the nodes in each layer of $\tau$ receive distinct labels. (That is, any tree which does not satisfy this property, will not be considered a "tree-structure.") We may adopt this convention at various times: whenever we show that witness trees have a certain structural property, then we will assume that all tree-structures under consideration have this property as well.

For a fixed tree-structure $\tau$, we say that $\tau$ *appears* if $\hat{\tau}^T = \tau$ for some $T > 0$. The most important structural result concerning the MT algorithm, which will appear in numerous contexts and variants throughout our work, is the following:

LEMMA 2.8 (Witness Tree Lemma). *Suppose $\tau$ is a tree-structure whose nodes are labeled $B_1, \ldots, B_t$. Then the probability $\tau$ appears is at most*

$$P(\tau \text{ appears}) \leq \prod_{j=1}^{t} P_\Omega(B_j)$$

*We refer to the value $\prod_{j=1}^{t} P_\Omega(B_j)$ as the* weight *of $\tau$, and we write it $w(\tau)$.*

PROOF. We will prove this using by using a coupling construction introduced in [**87**]; this coupling construction has significant limitations, but it is a good starting point. We will also assume that all of the events $\mathcal{B}$ are atomic events. (Every event in the space $\Omega$ is equivalent to a union of atomic events, so this actually does not lose generality.)

As the bad-events are atomic, each $B$ can be written as $B_j \equiv X_{i_1} = w_1 \wedge \cdots \wedge X_{i_k} = w_k$ and this has probability $P_\Omega(B) = P_\Omega(X_{i_1} = w_1) \times \cdots \times P_\Omega(X_{i_k} = w_k)$. So we can factor $w(\tau)$ as a product of terms of the form $P_\Omega(X_i = w)$.

Suppose that, before we run the MT algorithm, we construct a "resampling table" $R$. This table contains, for each variable $i \in [n]$, an infinite stream of values which are all drawn from the distribution $p_i$. We denote these by $R(i, j)$ for $j \geq 1$. All the entries in this table are independent. In the initial stage of the MT algorithm, we set $X_i = R(i, 1)$. The first time that the MT algorithm needs to resample variable $i$, it sets $X_i = R(i, 2)$, and in general for the $j$th resampling of variable $i$ sets $X_i = R(i, j+1)$. It is clear that that resulting algorithm has the probabilistic behavior as the MT algorithm, so this is a valid coupling.

Now, given the tree-structure $\tau$, we will show certain necessary conditions on the entries of $R$. Consider any variable $i$. All of the events $B_1, \ldots, B_t$ which contain $i$ must, by Proposition 2.7, occur on distinct layers of the tree. So, we can sort all of the occurrences of variable $i$ from deepest to shallowest as $B_{k_1}, \ldots, B_{k_r}$. Because we have assumed that the events are atomic, we may assume that these events demand respectively $X_i = v_1, \ldots, X_i = v_r$.

Now we claim that in order for $\tau$ to appear, it must be the case that the events $B_{k_1}, \ldots, B_{k_r}$ are resampled in that order, and that furthermore that these are the first $r$ events to be resampled which involve the variable $i$. The reason for this is that all of the events $B_{k_1}, \ldots, B_{k_r}$ are neighbors in the dependency graph (or are equal to each other), and in forming witness trees, earlier events are always eligible to be placed as children of later ones if they are dependent. So, for example, $B_{k_{r-1}}$ is eligible to be placed as a child of $B_{k_r}$, and so is placed below $B_{k_r}$ in $\hat{\tau}^T$. By a similar token, any event $B'$ which involves variable $i$ will satisfy $B' \sim B_{k_r}$, so if it occurs before $B_{k_r}$, it will be placed somewhere in the tree below $B_{k_r}$.

So, we have seen that the first bad-event to be resampled involving variable $i$ involves setting $X_i = v_1$. At that time, variable $i$ still has the variable it was assigned initially, $R(i, 1)$. Thus, a necessary condition for $\tau$ to appear is that $R(i, 1) = v_1$. By similar reasoning, we see that we must have $R(i, j) = v_j$ for $j = 1, \ldots, r$.

As all these entries of $R$ are independent, then the total probability of these events (for a fixed value of $i$) is $\prod_{j=1}^{r} P(R(i, j) = v_j) = \prod_{j=1}^{r} P_\Omega(X_i = v_j)$.

But, now consider how variable $i$ appears in the expansion of $w(\tau)$. As we have noted, this can be factored into a product of terms $P_\Omega(X_{i'} = w')$. For each $j = 1, \ldots, r$, there is a corresponding term in $w(\tau)$ $P_\Omega(X_i = v_r)$. Hence, the contribution of variable $i$ to $w(\tau)$, is also equal to $\prod_{j=1}^{r} P_\Omega(X_i = v_j)$.

We have given a necessary condition for variable $i$, in terms of the resampling table $R$, and shown an upper bound on the probability of this condition holding. As the entries of $R$ are independent, we can multiply across all values of $i$ to upper-bound the probability that $\tau$ appears. The resulting upper-bound is equal to $w(\tau)$. $\square$

We now discuss the second part of the MT proof, following the Witness Tree Lemma.

PROPOSITION 2.9. *We cannot have $\hat{\tau}^T = \hat{\tau}^{T'}$ for $T \neq T'$.*

PROOF. Suppose that $T < T'$, and that $\hat{\tau}^T = \hat{\tau}^{T'} = \tau$. Suppose that $\tau$ its root node labeled $B$ and includes $r$ nodes labeled $B$. Then in order for $\hat{\tau}^T = \tau$ there must be exactly $r-1$ times before $T$ in which $B$ was resampled, and that in addition $B$ was resampled at time $T$. Similarly in order for $\hat{\tau}^{T'} = \tau$ there must be exactly $r-1$ times before $T'$ in which $B$ was resampled. But, we know there are at least $r$ such times: all the times before $T$ in which $B$ was resampled, plus time $T$ itself. □

PROPOSITION 2.10. *We have*

$$\mathbf{E}[\#\textit{resamplings in } MT] \leq \sum_{\textit{tree structures } \tau} w(\tau)$$

PROOF. By Proposition 2.9, every resampling performed by MT produces a *distinct* witness tree. This implies that we can write the overall running time of the MT algorithm as

$$\#\text{ resamplings} \leq \sum_{\text{tree structures } \tau} [\tau \text{ appears}]$$

where we use the Iverson notation (here and throughout the paper), i.e. $[\mathcal{P}]$ is equal to one if predicate $\mathcal{P}$ holds and is zero otherwise.

Combining this with Lemma 2.8 gives the key convergence result

$$\mathbf{E}[\#\text{resamplings}] \leq \sum_{\text{tree structures } \tau} P(\tau \text{ appears}) \leq \sum_{\text{tree structures } \tau} w(\tau)$$

□

Obtaining an upper bound on the sum $\sum_{\text{tree structures } \tau} w(\tau)$ has certain technical similarities to computing extinction probabilities for Galton-Watson processes. This analogy is exploited in the proof appearing in [**87**]. However, we prefer to avoid this route, and show an upper bound on $\sum_{\text{tree structures } \tau} w(\tau)$ directly.

For any $h \geq 0$ and any $B \in \mathcal{B}$, we define

$$T_h(B) = \sum_{\substack{\text{tree-structures } \tau \\ \text{with height} \leq h \\ \text{with root node labeled } B}} w(\tau)$$

We recursively use the weighting function $\mu$ as an upper bound on $T_h(B)$ as follows:

PROPOSITION 2.11. *Suppose that $\mu$ satisfies Pegden's criterion (which is identical to the cluster-expansion criterion), namely that for all $B \in \mathcal{B}$ we have $\mu(B) \geq P_\Omega(B)\theta_G(B)$ where $G$ is the canonical dependency graph. Then, for all $B \in \mathcal{B}$ and $h \geq 0$, we have $T_h(B) \leq \mu(B)$.*

PROOF. We prove this by induction on $h$. When $h = 0$, this follows vacuously as $T_0(B) = 0$ and $\mu(B) \geq 0$.

We now show the induction step. Let $\tau$ be a tree-structure with a root node $v$ labeled by $B$. Let $v_1, \ldots, v_t$ be the children of this root node (possibly $t = 0$), with labels $B_1, \ldots, B_t$. Then $\tau$ can be written as the union of its root node $v$ plus tree-structures $\tau_1, \ldots, \tau_t$, of height $\leq h-1$ rooted in $B_1, \ldots, B_t$ respectively. Furthermore, $w(\tau) = P_\Omega(B)w(\tau_1)\ldots w(\tau_t)$.

All of $v_1, \ldots, v_t$ are at depth 1 in $\tau$, and hence by Lemma 2.7 $B_1, \ldots, B_t$ must be distinct, form an independent set, and be neighbors of $B$ in the dependency graph (or equal to $B$.) For any fixed choice of $B_1, \ldots, B_t$, the set of possible values for $w(\tau_1)\ldots w(\tau_t)$ is at most $T_{h-1}(B_1)\ldots T_{h-1}(B_t)$; by induction this is at most $\mu(B_1)\ldots\mu(B_t)$. Summing over all valid choices for $B_1, \ldots, B_t$, we have

$$\begin{aligned} T_h(\tau) &\leq P_\Omega(B) \sum_{\{B_1,\ldots,B_t\}} \mu(B_1)\ldots\mu(B_t) \\ &\leq P_\Omega(B) \sum_{\substack{I \subseteq N(B) \\ I \text{ independent in } G}} \prod_{B' \in I} \mu(B') \\ &\leq \mu(B) \end{aligned}$$

completing the induction. □

PROPOSITION 2.12. *Suppose that $\mu$ satisfies Pegden's criterion. Then for any $B \in \mathcal{B}$ we have*

$$\sum_{\substack{\text{tree-structures } \tau \\ \text{with root node labeled } B}} w(\tau) \leq \mu(B)$$

PROOF. Observe that any term $w(\tau)$ appears as a summand of $T_h$, for some finite $h$. Thus

$$\sum_{\substack{\text{tree-structures } \tau \\ \text{with root node labeled } B}} w(\tau) = T_\infty(B) = \lim_{h\to\infty} T_h(B) \leq \lim_{h\to\infty} \mu(B) = \mu(B)$$

□

We can define a key parameter to analyze the MT runtime:

DEFINITION 2.13 (Work parameter). *Given a weighting function $\mu$, define the* work parameter

$$W = \sum_{B\in\mathcal{B}} \mu(B)$$

THEOREM 2.14. *Suppose that $\mu$ satisfies Pegden's criterion. The expected # resamplings of MT is at most $W$.*

PROOF. We have already seen that $\mathbf{E}[\#\text{resamplings}] \leq \sum_{\text{tree-structures } \tau} w(\tau)$. So

$$\sum_{\text{tree-structures } \tau} w(\tau) \leq \sum_{B\in\mathcal{B}} \sum_{\substack{\text{tree-structures } \tau \\ \text{with root node labeled } B}} w(\tau) \leq \sum_{B\in\mathcal{B}} \mu(B) = W$$

□

In particular, this shows that the expected number of resamplings in the MT algorithm is $< \infty$; so the MT algorithm terminates with probability one.

COROLLARY 2.15 ([**87**]). *Suppose that the symmetric LLL criterion epd $\leq 1$ is satisfied. Then the MT algorithm terminates with probability one; the expected number of resamplings it performs is at most epm.*

PROOF. Setting $\mu(B) = ep$ for all $B$ satisfies the cluster-expansion criterion. Then $\sum_B \mu(B) \leq epm$. □

This concludes the proof of convergence for the MT algorithm. We will next examine several ways this result can be extended.

### 2.2. The MT distribution

After running the MT algorithm, we are guaranteed that the bad-events in $\mathcal{B}$ cannot possibly occur. In other words, we know that the configuration produced by the MT has the property that no $B \in \mathcal{B}$ is true. For a variety of reasons, we might want to know more about such configurations other than that they exist. Suppose for instance that we have some weights on our variables, and we define the objective function on a solution $\sum_i w(X_i)$; in this case, if we are able to estimate the probability that a variable $X_i$ takes on value $j$ in the MT algorithm, then we may be able to show that configurations with a good objective function exist. A second example is when the number of bad-events becomes too large, perhaps exponentially large. In this case, the MT algorithm cannot test them all. However, we may still be able to ignore a subset of the bad events, and argue that the probability that they are true at the end of the MT algorithm is small even though they were never checked.

Recall that we have defined $N(E)$ for any event which is a boolean function of a subset $S_E$ of the variables, not just for a bad-event. If we are given a weighting function $\mu$, in the setting of the asymmetric LLL, it will be convenient to extend the definition of $\theta$ to cover arbitrary events (not just bad-events). We also suppress the dependence on $G$, which is always assumed to be the canonical dependency graph.

$$\theta(E) = \sum_{\substack{I \subseteq N_G(E) \\ I \text{ independent}}} \prod_{B \in I} \mu(B)$$

In [**48**], it was shown how to adapt the proof of the MT algorithm to give a bound on the probability of any event in terms of $\theta(E)$:

THEOREM 2.16 ([**48**]). *Suppose that $\mu$ satisfies Pegden's criterion. Then for any event $E$ we have*

$$P(E \text{ is true in output of MT algorithm}) \leq P_\Omega(E)\theta(E)$$

PROOF. We will show a bound on the probability that $E$ is *ever* true during the execution of the MT algorithm; this automatically shows the desired result. Suppose that we monitor the execution of the MT algorithm; consider the first time $T$ that $E$ is true. We may build a witness tree $\hat{\tau}$ for this occurrence of $E$, in the usual manner: we place a root node labeled by $E$, and we continue adding nodes labeled by $B_T, \ldots, B_1$ as in the MT algorithm. (One can consider $E$ to be an "extra" bad-event, which is treated differently during the execution of the MT algorithm: instead of resampling $E$ when $E$ is true, we instead immediately halt the MT algorithm.)

It is not hard to see that the Witness Tree Lemma continues to hold for such trees rooted in $E$. Also, we then have

$$P(E) \leq \sum_{\text{tree-structures } \tau \text{ rooted in } E} P(\tau \text{ appears}) \quad \leq \sum_{\text{tree-structures } \tau \text{ rooted in } E} w(\tau)$$

Now, consider some tree-structure $\tau$ rooted in $E$. Its root node labeled $E$ contributes $P_\Omega(E)$ to its weight. It has children labeled $B_1, \ldots, B_t$ with subtrees $\tau_1, \ldots, \tau_t$, and $w(\tau) = P_\Omega(E)w(\tau_1)\ldots w(\tau_t)$. $B_1, \ldots, B_t$ must be distinct and independent by Proposition 2.7. Note that $E$ never occurs in the subtrees $\tau_1, \ldots, \tau_t$; these are simply trees whose nodes are labeled by $\mathcal{B}$. As shown in Proposition 2.11, for a fixed choice of $B_1, \ldots, B_t$, the total weight of all such tree-structures rooted in $B_1, \ldots, B_t$ respectively is at most $\mu(B_1)\ldots\mu(B_t)$. Now, sum over all valid choices of $B_1, \ldots, B_t$ (i.e. independent sets of neighbors of $E$) as in Proposition 2.11. □

The accounting method of Section 1.6 yields a particularly nice form for Theorem 2.16.

THEOREM 2.17. *Suppose we are given a vector $\lambda$ satisfying Theorem 1.16. Then, for any atomic event $E \equiv X_{i_1} = j_1 \wedge \cdots \wedge X_{i_k} = j_k$ we have*

$$P(E \text{ is true in output of MT algorithm}) \leq \prod_{s=1}^{k} \lambda_{i_s,j_s}$$

PROOF. Each independent set of neighbors of $E$ contains one or zero bad-events involving each variable $i_s$, so:

$$\begin{aligned} P_\Omega(E)\theta(E) &\leq P_\Omega(E) \prod_s (1 + \sum_{B' : i_s \in S_{B'}} \mu(B')) \\ &\leq \prod_s \frac{\lambda_{i_s,j_s}}{\lambda_{i_s}}(\lambda_{i_s}) \qquad \text{by (37)} \\ &= \prod_s \lambda_{i_s,j_s} \end{aligned}$$

□

Chapters 8,9, we will explore the MT distribution in much more detail. We note that Theorem 2.16 only provides an *upper bound* on the probability that $E$ occurs in the output of MT. In general, it is not possible to show a lower bound on the probability of an arbitrary event $E$ — if $E \in \mathcal{B}$, then necessarily $P(E) = 0$. There are some limited circumstances in which lower bounds may be possible, which we discuss in Chapter 8.

## 2.3. A parallel algorithm

The MT algorithm as described is sequential. In [**87**], Moser & Tardos described a parallel (RNC) variant of it:

1. Draw $X_1, \ldots, X_n \sim \Omega$.
2. While there is some true bad-event $B \in \mathcal{B}$:
    3. Select a maximal independent set $\mathcal{V} \subseteq \mathcal{B}$ of currently-true bad-events.
    4. For each variable $i \in \bigcup_{B \in V} S_B$, draw $X_i$ independently from its distribution under $\Omega$.

Step (4) can be thought of as resampling, in parallel, all of the bad-events in $\mathcal{V}$. Recall that in the MT sequential algorithm, there was complete freedom in determining the resampling rule. Thus, it would be a valid instantiation of the sequential MT algorithm to select an MIS of currently true bad-events and then resample then one by one. In this sense, the parallel algorithm is an instance or a simulation of the sequential MT algorithm, with a very strange resampling rule. This immediately shows that the parallel MT algorithm terminates with probability one.

We want to show that this parallel algorithm terminates quickly, say in a logarithmic number of steps. Unfortunately, the LLL criterion by itself is not sufficient to guarantee this. Rather, we will need a slightly stronger criterion, which we refer to as an *$\epsilon$-multiplicative slack condition:*

PROPOSITION 2.18. *Suppose that $\mu$ satisfies the following criterion:*

$$\mu(B) \geq (1+\epsilon) P_\Omega(B) \theta_G(B)$$

*where $G$ is the canonical dependency graph.*

*Then, with high probability, the Parallel MT algorithm terminates after $O(\frac{\log W}{\epsilon})$ steps.*

*In particular, this algorithm can be implemented in time $O(\frac{\log^2 m \log W}{\epsilon})$.*

PROOF. We will give a brief sketch of this proof, which can be found in more detail in [**87**]. First, observe that if we resample a bad-event $B$ during the $r$th round of the parallel MT algorithm, then the witness tree $\hat{\tau}^T$ will have height $r$ exactly. The reason for this is that there must be some $B' \sim B$ which was resampled during the $r-1$st round; for, if there was not, then $\{B\} \cup \mathcal{V}_{r-1}$ would be an independent set, contradicting the maximality of $\mathcal{V}_r$.

Next, we can show, along the same lines as Proposition 2.11, that the total weight of all tree-structures of height $h$ rooted in $B$ is at most $\mu(B)(1+\epsilon)^{-h}$.

Thus the probability that there is a resampling at round $r$ is at most the total probability that any tree-structure of height $r$ appears. This is at most $\sum_B \mu(B)(1+\epsilon)^{-r} = W(1+\epsilon)^{-r}$. So for $r = \Omega(\frac{\log W}{\epsilon})$ this probability is negligible.

The individual steps of the Parallel MT algorithm can typically be performed in polylogarithmic time. The most costly step is usually finding the MIS itself, which can cost $\log^2 m$ time using the algorithm of Luby [**80**]. Thus, the overall algorithm runs in time $O(\frac{\log^2 m \log W}{\epsilon})$.

□

There are many ways to improve this analysis and give better bounds on the parallel run-time; we discuss these in Chapter 3.

The following heuristic may give intuition as to why we need this extra slack condition for the Parallel MT algorithm. Suppose that our probability space is defined by setting $X_i = 0$ with probability $\epsilon$, and $X_i = 1$ with probability $1-\delta$. And suppose that $\mathcal{B} = \{X_i = 1 \mid i = 1, \ldots, n\}$. That is, we have $n$ highly-biased coins and we want to ensure that all the coins come up heads. It is not hard to see that Proposition 2.18 is satisfied with $\epsilon = \delta/2$ and $\mu(B) = 4/\delta$ (for small $\delta$).

In this case, the canonical dependency graph $G$ consists of $n$ isolated points, each with probability $1-\delta$. In each stage of the parallel MT algorithm, we simply randomly flip all of variable $i$ which are currently equal to 1. At each stage of this process, the expected number of variables $i$ satisfying $i = 1$ decreases by a factor of $1-\delta$, and it is not hard to see that this process requires $\Theta(\delta^{-1} \log n)$ rounds to drive all the variables equal to 0. We also observe that this is precisely the number of steps predicted by Proposition 2.18 — here $\epsilon^{-1} \log W = \Theta(\delta^{-1} \log n)$.

This example of $n$ independent biased coins, although highly simplistic, is a very useful extremal case in analyzing the MT algorithm, particularly the parallel MT algorithm. We recommend that the reader always consider this example first in understanding any estimates of the parallel MT.

**2.3.1. Shearer bounds on the parallel MT algorithm.** The weighting functions $\mu$ and $W$ plays a somewhat mysterious role in the LLL, and it can be confusing to have them appear in the complexity bounds for the parallel MT algorithm. To make this even more confusing, a weighting function which satisfies the criteria for the Asymmetric LLL will not necessarily satisfy the criterion for the parallel MT algorithm (which requires an $\epsilon$ slack). Consider the example again of $n$ independent coins with probability $p = (1+\epsilon)^{-1}$. It is fairly simple to see that the resampling process terminates after $O(\frac{\log n}{\epsilon})$ rounds with high probability, yet there is no finite weighting function $\mu$ which can satisfy Proposition 2.18.

Indeed, as shown in [**65**], one can directly compute a criterion for the convergence of the parallel MT algorithm from the Shearer criterion, without the use of a weighting function $\mu$. Note that the Shearer criterion states that we have $Q(I, G, p) > 0$ for all independent sets $I$. The $\epsilon$ in the running time for the parallel MT algorithm is derived essentially from *how far* $Q(I, G, p)$ is bounded away from zero.

Definition 2.19. *We say that the Shearer criterion is satisfied with $\epsilon$-multiplicative slack, if the vector of probabilities $(1+\epsilon)P_\Omega$ satisfies the Shearer criterion.*

Observe that since the Shearer criterion is an open set condition on $p$, it follows that if $p$ satisfies the Shearer criterion, it must be satisfied with $\epsilon$-multiplicative slack for some $\epsilon > 0$. In [**65**], Kolipaka & Szegedy showed that if the Shearer criterion is satisfied, then the run-time of the Parallel MT algorithm can be bounded in terms of the slack $\epsilon$ alone (and not the measure $\tilde{\mu}$).

Proposition 2.20 ([**65**]). *Whp, the number of rounds executed by the parallel MT algorithm is at most $O(\epsilon^{-1}\log(m/\epsilon))$.*

For the most part, we will work with the asymmetric LLL criterion and its variants, as the Shearer criterion is too difficult technically. However, Theorem 2.6 will also allow us to work with the Shearer criterion when that is more convenient; any result we show for the Shearer criterion applies also to the MT algorithm.

**2.3.2. Alternative parallel LLL algorithms.** The computation of a maximal independent set is a relatively costly which dominates the runtime of this parallel algorithm. In [**26**], Chung et al. gave several alternative algorithms for the symmetric LLL which either avoid this step or reduce its cost. Although the main focus of [**26**] was obtaining distributed algorithms for the LLL, these algorithm have reduced time complexity as well.

They give one algorithm, based on bad-events choosing random priorities and resampling a bad-event if it has earlier priority than its neighbors, which runs in $O(\epsilon^{-1} \log n)$ distributed rounds and $O(\epsilon^{-1} \log^2 n)$ time. Unfortunately, this algorithm of [**26**] requires a stronger criterion than the LLL: namely, in the symmetric setting, it requires that $epd^2 \leq (1 - \epsilon)$. In many applications of the LLL, particularly those based on Chernoff bounds for the sum of independent random variables, satisfying the stricter criterion $epd^2 \leq (1 - \epsilon)$ leads to qualitatively similar results as the symmetric LLL. In other cases, the criterion of [**26**] loses much critical precision leading to weaker results. In particular, their bound essentially corresponds to the state of the art [**86**] before the break-through result of Moser and Moser-Tardos [**87**].

Another algorithm given by Chung et al. requires only the standard symmetric LLL criterion and runs in $O(\epsilon^{-1} \log^2 d \log n)$ rounds. Recently, a key subroutine used by this algorithm was improved by [**44**], leading to a reduction to $O(\epsilon^{-1} \log d \log n)$ rounds. When $d$ is polynomial in $n$, however, this does not improve on the MT algorithm.

Neither of the algorithms given by Chung et al. generalize to the asymmetric LLL setting.

In Chapter 3, we will give a new parallel algorithm for the LLL.

## 2.4. Lopsidependency

The MT algorithm and criterion we have given so far, is defined for the canonical *dependency* graph. We have already seen instances in which the LLLL is applied

to more general types of probability spaces. In general, these probability spaces do not necessarily even have "variables" or are representable in any succinct way. Thus, it is difficult to see what it would even mean to extend the MT algorithm to cover the full generality of the LLLL. However, the simplest form of the LLLL is the variable-assignment LLLL. Moser & Tardos showed that this was already by the MT algorithm.

THEOREM 2.21. *Suppose that all the events in $\mathcal{B}$ are atomic events, and let $G$ be the canonical* lopsidependency *graph among them. Suppose there is a weighting function $\mu : \mathcal{B} \to [0, \infty)$ which satisfies the cluster-expansion criterion $\mu(B) \geq P_\Omega(B)\theta_G(B)$.*

*Then, the MT algorithm terminates with probability one; the expected number of resamplings it performs is at most $\sum_{B \in \mathcal{B}} \mu(B)$.*

In this context, it is useful to redefine our rule for defining $\sim$ on variable-defined events. Namely, suppose that $E, E'$ are atomic events. Then we say that $E \sim E'$ iff $E, E'$ disagree on a variable. By convention, we also define $B \sim B$. Thus, again $N(B)$ is the inclusive neighborhood of a bad-event $N(B) = N_G(B) \cup \{B\}$. We note that, unlike for the standard LLL, the rule that $B \sim B$ is really a special case: while $B$ overlaps on variables with itself, it does not disagree on any variables.

We prove Theorem 2.21 via witness trees in a similar manner to the usual MT algorithm. We form witness trees, we use the same rule as we did for the ordinary MT algorithm; that is, we add a node labeled by $B$ as a child of $B'$ if $B \sim B'$. However, in this case, we are using the alternate lopsided definition of $\sim$. We next show that the Witness Tree Lemma holds with this modification. The remainder of the proof of MT, such as Proposition 2.11 and Theorem 2.14, follow immediately,

LEMMA 2.22. *Suppose $\tau$ is a tree-structure, whose nodes are labeled $B_1, \ldots, B_t$. Then the probability that $\tau$ appears is at most $\prod_{j=1}^{t} P_\Omega(B_j)$*

PROOF. We first note that the proof of Lemma 2.8 does not hold here. In the previous proof, we assumed that for any variable $i$, all of the occurrences of variable $i$ appear at distinct layers of the tree, so they can be unambiguously sorted by time.

However, one can slightly modify this idea. The layers of the witness tree are still independent. This means that if there are two bad-events $B, B'$ at the same layer of the tree and $i \in S_B \cap S_{B'}$, then $B$ and $B'$ *agree* on variable $i$, say they both demand that $X_i = j$.

Now, suppose we consider some fixed variable $i$. Looking at the layers of $\tau$ from its greatest depth $h$ to its root, suppose that there are $r_j$ occurrences of variable $i$ at depth $j$ in $\tau$. And furthermore, suppose that all the occurrences of variable $i$ at depth $j$ demand that $X_i = v_j$. (If $r_j = 0$, then $v_j$ can be chosen arbitrarily).

In this case, we observe that the first $r_h$ entries of $R(i, \cdot)$ must be equal to $v_h$; the next $r_{h-1}$ entries of $R(i, \cdot)$ must be equal to $v_{h-1}$, and so forth. These contribute a total probability of $\prod_{j=h}^{1} P_\Omega(X_i = v_j)^{r_j}$; this is also precisely the term involving variable $i$ that appears in the expansion of $w(\tau)$. □

While many aspects of the MT algorithm algorithm carry over immediately to the lopsidependent setting, the parallel algorithm is an exception. The main problem is that if $B, B'$ are "independent" (in the sense of the canonical lopsidependency graph), then they may still share a common variable, and so they cannot be resampled in parallel (they are not computationally independent). It is somewhat frustrating that we have a parallel algorithm for the LLL, and we have a sequential algorithm for the LLLL, but we cannot combine the two.[1]

In Chapter 4, we will examine the behavior of the MT algorithm in the variable-assignment LLLL setting much more closely. We will show that in many cases the MT algorithm converges even though the LLLL criterion is not satisfied.

[1]In [**23**], there is a brief discussion about a parallel deterministic algorithm for the variable-assignment LLLL. However, no algorithm is provided, nor are there any definite claims made for its performance.

## 2.5. The role of the Witness Tree Lemma

Lemma 2.8, the Witness Tree Lemma, plays a central role in the proof of the MT algorithm. What makes it so powerful is that an individual witness tree $\tau$ only contains information about a small subset of the resamplings performed by the overall algorithm. In the proof of this lemma, we ignore all these other resamplings and are able to obtain strong probabilistic bounds on the resamplings represented by $\tau$, despite the unknown contribution of other resamplings.

The Witness Tree Lemma is fairly straightforward to prove for the MT algorithm. We will encounter many variants of the MT algorithm later in this work for which we will need modified forms of this lemma. Some of these will have extremely difficult and subtle proofs. Once we have proved the corresponding version of the Witness Tree Lemma, the remainder of the convergence proof (from Proposition 2.9 onward), as well as the MT distribution, remains almost unchanged. The results on the parallel algorithm do not always generalize so cleanly, however one important result (that the height of a witness tree is bounded by $O(\frac{\log W}{\epsilon})$ with high probability) does generalize as well.

In analyzing (variants of) the MT algorithm, there is a tension in how much information to include in a witness tree. What makes the Witness Tree Lemma potentially difficult to prove in general is the influence of the bad-events we have ignored. Every time we ignore a bad-event, our probabilistic reasoning must "argue around" it. Thus, the more information we include in a witness tree, the easier the Witness Tree Lemma becomes to prove.

However, note that in Proposition 2.10, we are taking a union-bound over all tree-structures. If we keep track of more information in a witness tree, then the space of all possible tree-structures grows. This, we will be taking a union-bound over a large space, and this leads to weaker probabilistic bounds and weaker control over the behavior of the MT algorithm.

We have already seen an example of this strategy, in Section 2.4, where we change the definition of $\sim$ in terms of variable disagreement instead of just variable overlap. This means that the witness trees include fewer nodes. The Witness Tree Lemma is only slightly harder to prove in this case than for the ordinary MT algorithm: we can still use the same type of coupling argument, based on a resampling table.

One can decide to keep track of *all* the resamplings performed by the MT algorithm (in which the "witness tree" is really a forest). Achlioptas & Ilioupoulous [**1**], Harvey & Vondrak [**51**], and Kolmogorov [**68**] have described generalized LLL algorithms, which uses this technique to prove convergence. Because of the very large space of "witness forests" which result, these lead to exponentially weaker bounds on the behavior of their algorithms. These are not able to show the MT distribution, parallel algorithms, or others. We will discuss this issue much more in detail, in the context of an algorithm for the LLL on permutations, in Section 7.5.3.

In Chapter 3, we will explore how to generalize the witness tree. This leads to the concept of a "witness dag", a much general structure to keep track of some or all of the resamplings performed by the MT algorithm. These are useful for developing stronger bounds on the number of resamplings performed by the parallel and sequential MT algorithms.

CHAPTER 3

# The witness dag: a tool for improved bounds and parallel algorithms for the Lovász Local Lemma

In this chapter, we introduce a new theoretical structure to analyze the behavior of the MT algorithm, which we refer to as the *witness dag*. This provides an explanation or history for some or all of the resamplings that occur. This generalizes the notion of a witness tree which only provides the history of a single resampling. We use this tool to show stronger bounds on the runtime of the sequential and parallel algorithms.

For the remainder of this chapter, we will always assume, unless stated otherwise, that our probability space satisfies the Shearer criterion with $\epsilon$-multiplicative slack. We will occasionally drop this assumption when we are discussing the symmetric LLL criterion, but this will always be derived as a corollary of results on the full Shearer criterion.

We we deal exclusively in this chapter with the canonical dependency graph (not the lopsidependency graph). Thus, we simplify our notation for the Shearer criterion by writing $Q(I, p)$ to mean $Q(G, I, p)$ where $G$ is the canonical dependency graph.

We will also make extensive use of the coupling construction we have seen for the proof of the Witness Tree Lemma, based on a resampling table $R$. Recall that in this construction, we have a table of values $R(i, t)$, where $i$ ranges over the variables $1, \ldots, n$ and $t$ ranges over the natural numbers $1, 2, \ldots$. Each cell $R(i, t)$ is drawn independently from the distribution on the variable $i$, that $R(i, t) = j$ with probability $p_{ij}$, independently of all other cells. The intent of this table is that, instead of choosing new values for the variables in "on-line" fashion, we precompute the future values of all the variables. The first entry in the table $R(i, 1)$, is the initial value for the variable $X_i$; on the $t^{\text{th}}$ resampling, we set $X_i = R(i, t + 1)$.

**3.0.1. Overview.** In Section 3.1, we define and analyze the witness dag. We use this tool to show stronger bounds on Parallel MT algorithm:

THEOREM 3.1. *With high probability, the Parallel MT algorithm terminates after* $O(\epsilon^{-1} \log n)$ *rounds.*

*Suppose we have an oracle which can determine all the bad-events true on a given configuration, in time* $O(\log^2 n)$. *The total complexity is* $O(\epsilon^{-1} \log^3 n)$ *time and* $\epsilon^{-1} n^{O(1)}$ *processors.*

These bounds are all independent of the LLL weighting function $\tilde{\mu}(B)$ and the number of bad-events $m$. These significantly improve on qualitatively similar bounds shown in Kolipaka & Szegedy [**65**], which show that Parallel MT algorithm terminates, with constant probability, after $1/\epsilon \log(n/\epsilon)$ rounds.[1]

In Section 3.2, we show a new, stronger concentration result for the runtime of the sequential MT algorithm. This bound improves on similar concentration bounds shown in [**65**] and [**1**].

THEOREM 3.2. *Suppose that the asymmetric LLL criterion is satisfied with* $\epsilon$*-multiplicative slack. With high probability, the total number of resamplings made the MT algorithm is at most* $O(\sum_B \mu(B) + \epsilon^{-1} \log^2 n)$.

*Suppose that the symmetric LLL criterion epd* $\leq 1$ *is satisfied. Then, whp, the number of resamplings is at most* $O(n + d \log^2 n)$.

*We obtain a similar result for when the Shearer criterion is satisfied as well.*

In Section 3.3 and 3.4, we develop a new parallel algorithm for the LLL. The basic idea of this algorithm is to select a random resampling table and then precompute all possible resampling-paths compatible with it. Surprisingly, this larger collection, which in a sense represents all possible choices for the trajectory of the MT algorithm, can still be computed relatively quickly (in approximately $O(\epsilon^{-1} \log^2 n)$ time). Next,

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

we find a *single* MIS of this larger collection, which will allow us to determine the complete set of resamplings necessary. It is this reduction from $\epsilon^{-1} \log n$ separate MIS algorithms to just one that is the key to our improved runtime.

It requires some definitions to state this result in its full "asymmetric" form, so we defer that until later. For the symmetric LLL, we can give a simpler statement of our new algorithm:

THEOREM 3.3. *Suppose that we can determine if any bad-event $B$ is true on a given configuration in time $O(\log n)$. Suppose that the symmetric LLL criterion $ep(1+\epsilon)d \leq 1$ is satisfied for some $\epsilon > 0$.*

*Then we find a configuration avoiding $\mathcal{B}$ using $\tilde{O}(\epsilon^{-1} \log(mn) \log n)$ time and $(mn)^{O(1)}$ processors.*

In Section 3.5, we show how this algorithm can be derandomized, to give an NC algorithm which also requires $O(\epsilon^{-1} \log^2(mn))$ time under a stronger LLL criterion $epd^{1+\epsilon} \leq 1$. This improves on an algorithm of [**23**] which requires $O(\epsilon^{-1} \log^3(mn))$ time. (These results can be extended to an asymmetric setting, but there are many more technical conditions on the precise form of $\mathcal{B}$.)

### 3.1. The witness dag

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

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

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

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

Given a full execution of the MT algorithm, one can form a witness dag $\hat{G}$ which we refer to as the *full witness dag*, as follows. Suppose that we resample bad-events $B_1, \dots, B_t$. Then $\hat{G}$ has vertices $v_1, \dots, v_t$ which are labeled $B_1, \dots, B_t$. We place an edge from $v_i$ to $v_j$ iff $i < j$ and $B_i \sim B_j$. It is not hard to see that this graph is indeed a witness dag as we have defined it. We emphasize that $\hat{G}$ is a random variable, and we distinguish between this notion and that of a witness dag (which is a non-random variable). The full witness dag (under different terminology) was analyzed by Kolipaka & Szegedy in [**65**], and we will use their results in numerous places. However, we will also consider partial witness dags, which record information about only a subset of the resamplings. As we will see, these partial witness dags can be useful even when we wish to analyze the full set of resamplings.

As tree-structures and single-sink witness dags are closely related, we will often use the notation $\tau$ for a single-sink witness dag.

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

Definition 3.4 (Path of a variable). *Let $G$ be a witness dag. For any $i \in [n]$, let $G[i]$ denote the subgraph of $G$ induced on all vertices $v$ labeled by $B$ with $i \in S_B$.*

*Because of the comparability condition, $G[i]$ is linearly ordered by $\prec$; thus we refer to $G[i]$ as the* path *of variable $i$.*

DEFINITION 3.5 (Configuration of $v$). *Let $G$ be a witness dag and $R$ a resampling table. Let $v \in G$ be labeled by $B$. For each $i \in S_B$, let $y_{v,i}$ denote the number of vertices $w \in G$ such that $w \prec v$.*

*We now define the* configuration of $v$ *by*

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

DEFINITION 3.6 (Compatibility of dag $G$ with resampling table $R$). *For a witness dag $G$ and a resampling table $R$, we say that $G$ is* compatible *with $R$ if, for all nodes $v \in G$ labeled by $B \in \mathcal{B}$, it is the case that $B$ is true on the configuration $X_G^v$.*

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

We can easily extend key results of Moser & Tardos from tree-structures to witness dags:

DEFINITION 3.7 (Weight of a dag). *Let $G$ be any witness dag, whose nodes are labeled by bad-events $B_1, \ldots, B_s$. We define the* weight *of $G$ to be $w(G) = \prod_{k=1}^{s} P_\Omega(B_k)$.*

PROPOSITION 3.8. *Let $G$ be any witness dag. For a random resampling table $R$, $G$ is compatible with $R$ with probability $w(G)$.*

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

The following result shows how witness dags and resampling tables are related to the MT algorithm:

PROPOSITION 3.9. *Suppose we run the MT algorithm, taking values for the variables from the resampling table $R$. Then $\hat{G}$ is compatible with $R$.*

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

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

These two results give us immediately the main lemma from Moser & Tardos:

COROLLARY 3.10. *Let $G$ be a witness dag. Then the probability that $\hat{G} = G$ is at most $w(G)$.*

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

DEFINITION 3.11 (Prefix graph). *For any vertices $v_1, \ldots, v_l \in V$, let $G(v_1, \ldots, v_l)$ denote the subgraph of $G$ induced on all vertices which have a path to at least one of $v_1, \ldots, v_l$.*

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

Using definition 3.11, we can give a more compact definition of the configuration of a node:

PROPOSITION 3.12. *For any witness dag $G$ and $v \in G$, we have*

$$X_G^v(i) = R(i, |G(v)[i]|)$$

PROOF. Suppose that $v$ is labeled by $B$. The graph $G(v)[i]$ contains precisely $v$ itself and the other nodes $w \in G$ with $w \prec v$. □

PROPOSITION 3.13. *Suppose $G$ is compatible with $R$ and $H$ is a prefix of $G$. Then $H$ is compatible with $R$.*

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

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

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

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

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

**3.1.3. Counting tree-structures and witness dags.** If we are given a collection of witness dags $\mathcal{G}$, we define the *total weight* of $\mathcal{G}$ as $\sum_{G \in \mathcal{G}} w(G)$. In this section, we count the total weight of certain classes of witness dags. In light of Corollary 3.10, these will upper-bound the expected number of resamplings.

PROPOSITION 3.14 ([**65**]). *Let $B \in \mathcal{B}$. Then the total weight of all witness dags with a single sink node labeled $B$, satisfies*

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

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

Now observe that by the comparability condition each set $I_j$ is an independent set, and for each $B' \in I_{j+1}$ there is some $B'' \sim B', B'' \in I_j$. Also, the mapping from $G$ to $I_0, \ldots, I_j$ is injective. We thus may sum over all such $I_1, \ldots, I_\infty$ to obtain an upper bound on the weight of such witness dags. In [**65**] Theorem 14, this sum is shown to be $Q(\{B\}, P_\Omega)/Q(\emptyset, P_\Omega)$ (although the notation they use is slightly different.) □

We will now take advantage of the $\epsilon$-multiplicative slack in our probabilities.

DEFINITION 3.15 (Adjusted weight). *For any witness dag $G$, we define the* adjusted weight *with respect to rate factor $\rho$ by*

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

PROPOSITION 3.16. *Let $B$ be any bad-event. Then for $\rho \in [0, \epsilon]$, we have*

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

PROOF. The probabilities $(1+\epsilon)P_\Omega$ satisfy the LLL criterion, and by Proposition 1.14 so must $(1+\rho)P_\Omega$. Now apply Proposition 3.14 to the probabilities $(1+\rho)P_\Omega$. □

PROPOSITION 3.17. *For any variable $i \in [n]$ and any $\rho \in [0, \epsilon)$, we have*

$$\sum_{B \sim i} \frac{Q(\{B\}, (1+\rho)P_\Omega)}{Q(\emptyset, (1+\rho)P_\Omega)} \leq \frac{1+\rho}{\epsilon - \rho}$$

PROOF. Let $V = \{B \mid i \in S_B\}$. Now, consider the set of probabilities $p$ given by

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

Note that $p \leq (1+\epsilon)P_\Omega$ and so by Proposition 1.14 we have $Q(\emptyset, p) > 0$. But, now consider that we have

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

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

$$\sum_{B \in V} \frac{Q(\{B\}, (1+\rho)P_\Omega)}{Q(\emptyset, (1+\rho)P_\Omega)} = \frac{\sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega)}{Q(\emptyset, p) + \frac{(\epsilon-\rho)}{1+\rho} \sum_{B \in V} Q(\{B\}, (1+\rho)P_\Omega)}$$

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

□

COROLLARY 3.18 ([**65**]). *The total weight of all tree-structures satisfies*

$$\sum_{\textit{tree-structures } \tau} w(\tau) \leq n/\epsilon$$

PROOF. We have

$$\begin{aligned}
\sum_{\text{tree-structures } \tau} w(\tau) &\leq \sum_i \sum_{\substack{\tau \text{ has a single sink node} \\ \text{labeled by some } B\sim i}} w(\tau) \\
&\leq \sum_i \sum_{\substack{\tau \text{ has a single sink node} \\ \text{labeled by some } B\sim i}} \frac{Q(\{B\},P_\Omega)}{Q(\emptyset,P_\Omega)} \qquad \text{by Proposition 3.16} \\
&\leq \sum_i \epsilon^{-1} \qquad \text{by Proposition 3.17} \\
&= n/\epsilon
\end{aligned}$$

□

PROPOSITION 3.19. *For $r \geq 1+1/\epsilon$, the expected number of single-sink witness dags compatible with $R$ containing more than $r$ nodes is at most $enr(1+\epsilon)^{-r}$*

PROOF. Summing over such dags:

$$\begin{aligned}
&\sum_{\substack{\tau \text{ has single sink node} \\ |\tau|\geq r}} P(\tau \text{ compatible with } R) \\
&= \sum_{\substack{\tau \text{ has single sink node} \\ |\tau|\geq r}} w(\tau) \\
&\leq (1+\rho)^{-r} \sum_{\substack{\tau \text{ has single sink node} \\ |\tau|\geq r}} w(\tau)(1+\rho)^{|\tau|}
\end{aligned}$$

$$
\begin{aligned}
&\text{for any } \rho \in [0, \epsilon] \\
&\leq (1+\rho)^{-r} \sum_{\substack{\tau \text{ has single sink node} \\ |\tau| \geq r}} a_\rho(\tau) \\
&\leq (1+\rho)^{-r} \sum_{i \in [n]} \sum_{B | i \in S_B} \sum_{\substack{\tau \text{ has single sink node } B \\ |\tau| \geq r}} a_\rho(\tau) \\
&\leq (1+\rho)^{-r} n \frac{1+\rho}{\epsilon - \rho} \qquad \text{by Propositions 3.16, 3.17}
\end{aligned}
$$

Now take $\rho = \epsilon - (1+\epsilon)/r$. By our condition $r \geq 1 + 1/\epsilon$ we have $\rho \in [0, \epsilon]$ and so Proposition 3.17 applies. Hence the expected number of such witness dags is at most $n \frac{r^r}{(r-1)^{r-1}(1+\epsilon)^r} \leq enr(1+\epsilon)^{-r}$. □

COROLLARY 3.20. *Whp, all single-sink witness dags compatible with $R$ contain $O(\frac{\log(n\epsilon^{-1})}{\epsilon})$ nodes. Whp all but $\frac{10 \log n}{\epsilon}$ single-sink witness dags compatible with $R$ contain $\leq \frac{10 \log n}{\epsilon}$ nodes.*

PROOF. This follows immediately from Markov's inequality and Proposition 3.19. □

COROLLARY 3.21. *Whp, all witness dags compatible with $R$ have height $O(\frac{\log n}{\epsilon})$.*

PROOF. Suppose that there is a witness dag $G$ of height $T$. Then for $i = 1, \ldots, T$ there is a single-sink witness dag of height $\geq i$, and all such dags are distinct. (Select a node $v$ of height $i$, and set $G_i = G(v_i)$. ) In particular, for $i = T/2$, we have that there are $\Omega(T)$ single-sink witness dags of height $\Omega(T)$ compatible with $R$. By Corollary 3.20, this implies $T = O(\frac{\log n}{\epsilon})$. □

With this Corollary 3.21, we are able to give a better bound on the complexity of the Parallel MT algorithm. The following Proposition 3.22 is remarkable in that the complexity is phrased solely in terms of the number of variables $n$ and the slack $\epsilon$, and is otherwise independent of $\mathcal{B}$.

PROPOSITION 3.22. *With high probability, the Parallel MT algorithm terminates after* $O(\epsilon^{-1} \log n)$ *rounds.*

*Suppose we have an oracle which can determine all the bad-events true on a given configuration, in time* $O(\log^2 n)$. *The total complexity of the Parallel MT algorithm is* $O(\epsilon^{-1} \log^3 n)$ *time and* $\epsilon^{-1} n^{O(1)}$ *processors.*

PROOF. An induction on $i$ shows that if the Parallel MT Algorithm runs for $i$ steps, then $\hat{G}$ has depth $i$, and it is compatible with $R$. But Corollary 3.21 shows that, whp, this implies that $i = O(\frac{\log n}{\epsilon})$. This implies that the total time needed to identify true bad-events is $O(i \log^2 n) \leq O(\frac{\log^3 n}{\epsilon})$.

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

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

We contrast this with a qualitatively similar result in Kolipaka & Szegedy, which shows that Parallel MT algorithm terminates, with constant probability, after $\epsilon^{-1} \log(m/\epsilon)$ rounds.

### 3.2. Concentration for the number of resamplings

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

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

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

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

We contrast our result with Kolipaka & Szegedy [**65**], which shows that the MT algorithm terminates after $O(n^2/\epsilon + n/\epsilon \log(1/\epsilon))$ resamplings with constant probability. In [**1**], a similar type of concentration result is shown: they show that in the symmetric LLL setting, their algorithm (which is a variant/generalization of the MT algorithm) performs $O(n/\epsilon)$ resamplings whp.

It will be more technically convenient, in this section, to work with the Shearer criterion. In this case, we use a slightly different parameter than $W$; namely we use

$$\tilde{W} = \sum_{B \in \mathcal{B}} \tilde{\mu}(B).$$

As we have seen in Theorems 2.5, 2.6, $W \leq \tilde{W}$ and $\tilde{W}$ is an upper bound on the expected number of resamplings performed by the MT algorithm. So it will suffice to show concentration around $\tilde{W}$ resamplings.

PROPOSITION 3.23. *Given any distinct bad-events $B_1, \ldots, B_s$, the total weight of all witness dags with $s$ sink nodes labelled $B_1, \ldots, B_s$, is at most $\prod_{i=1}^{s} \tilde{\mu}(B_i)$.*

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

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

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

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

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

$$\begin{aligned}
\sum_{G \text{ has } s \text{ sink nodes } B_1, \ldots, B_s} w(G) &\leq \sum_{\tau_1, \ldots, \tau_s} w(\tau_1) \ldots w(\tau_s) \\
&= \prod_{i=1}^{s} \sum_{\tau \text{ has single sink node } B_i} w(\tau) \\
&\leq \prod_{i=1}^{s} \tilde{\mu}(B_i) \qquad \text{by Proposition 3.14}
\end{aligned}$$

□

THEOREM 3.24. *With high probability, the total number of resamplings made the MT algorithm is at most $O(\tilde{W} + \epsilon^{-1} \log^2 n)$*

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

$$\begin{aligned}
\sum_{G \text{ has } s \text{ sink nodes}} P(G \text{ compatible with } R) &\leq \sum_{G \text{ has } s \text{ sink nodes}} w(G) \\
&\leq \sum_{B_1,\ldots,B_s \text{ distinct}} \sum_{\substack{G \text{ has sink nodes} \\ \text{labeled } B_1,\ldots,B_s}} w(G) \\
&\leq \sum_{B_1,\ldots,B_s \text{ distinct}} \tilde{\mu}(B_1)\ldots\tilde{\mu}(B_s) \qquad \text{by Proposition 3.23} \\
&\leq \frac{1}{s!}(\sum_{B\in\mathcal{B}} \tilde{\mu}(B))^s \\
&= \frac{\tilde{W}^s}{s!}
\end{aligned}$$

Now, suppose that the MT algorithm runs for $t$ time-steps. Let $\hat{G}$ be the full witness dag of the resulting execution. Each resampling at time $i \in \{1,\ldots,t\}$ corresponds to some vertex $v_i$ in $\hat{G}$.

By Proposition 3.20, all but $\frac{10\log n}{\epsilon}$ single-sink witness dags contain $\leq \frac{10\log n}{\epsilon}$ nodes. As our goal is to show an event happens whp, we assume for the rest of this proof that this event has happened. Now, let $X$ denote the set $\{i \mid |\hat{G}(v_i)| \leq h\}$ where $h = \frac{10 log n}{\epsilon}$. Under our assumption, we must have $|X| \geq t - \frac{10\log n}{\epsilon}$; as we are seeking to show that $t \geq \frac{\log^2 n}{\epsilon}$ we may assume that $|X| \geq t/2$. For each $i = 1,\ldots,|X|$ let $H_i = \hat{G}(x_i)$ where $x_j$ denotes the $i$th element of $X$ in order.

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

The subgraph $\hat{G}(x_{i_1},\ldots,x_{i_s})$ must contain exactly $s$ sink nodes $x_{i_1},\ldots,x_{i_s}$ (as each $x_{i_j}$ cannot have a path to any $x_{i_{j'}}$.) Furthermore, for any such choice of $i_1,\ldots,i_j$,

the resulting witness dags $\hat{G}(x_{i_1}, \ldots, x_{i_s})$ are distinct. Finally, by Proposition 3.13 each such witness dag is compatible with $R$.

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

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

By Proposition 3.27 (which we defer to after this proof), under the assumption that $|H_j| \le h$ for all $j$, this expression is at least $\binom{t/2-(s-1)h}{s} \ge \frac{(t/2-sh)^s}{s!}$.

Hence, we have shown that the number of witness dags with $s$ sink nodes compatible with $R$ is at least $\frac{(t/2-sh)^s}{s!}$. Recalling that the expected number of such dags is at most $W^s/s!$, the latter by Markov's inequality has probability at most $W^s/(t/2-sh)^s$.

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

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

□

Corollary 3.25. *The MT algorithm performs $O(n/\epsilon)$ resamplings whp.*

Proof. We have $\tilde{W} = \sum_B \mu(B) \le \sum_i \sum_{B \sim i} \mu(B) \le n/\epsilon$. Thus, by Theorem 3.24, with high probability the total number of resamplings made the MT algorithm is at most $O(\epsilon^{-1} n + \epsilon^{-1} \log^2 n) = O(n/\epsilon)$ □

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

Corollary 3.26. *Suppose the symmetric LLL criterion epd $\le 1$ applies. Then, whp, the number of resamplings is $O(n + d \log^2 n)$.*

PROOF. Set $\mu(B) = \frac{1}{d}$ for all $B \in \mathcal{B}$. Now a simple calculation shows that we satisfy the asymmetric LLL condition for $\epsilon = e((d-1)/d)^{d-1} - 1 = \Omega(1/d)$.

Also, observe that $\tilde{\mu}(B) \leq \mu(B) = 1/d$, and so $\tilde{W} \leq m/d$. We also may observe that $m \leq nd$. So, by Theorem 3.24, the total number of resamplings is, with high probability $O(n + d\log^2 n)$. □

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

PROPOSITION 3.27. *Suppose that $A$ is a set of positive integers of cardinality $|A| = t$. Suppose that for each $j \in \mathbf{Z}$ there is a set of positive integers $I_j$, with $|I_j| \leq h$ for all $j$. Define*

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

*Then we have*

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

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

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

as claimed.

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

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

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

and the induction is proved. □

### 3.3. Mutual consistency of witness dags

In Section 3.1, we have seen conditions for witness dags to be compatible with a *given* resampling table $R$. In this section, we examine when a set of witness dags can be mutually consistent, in the sense that they could all be prefixes of some (unspecified) full witness dag.

DEFINITION 3.28 (Consistency of $G, G'$). *Let $G, G'$ be witness dags. We say that $G$ is* consistent *with $G'$ is, for all variables $i$, either $G[i]$ is an initial segment of $G'[i]$ or $G'[i]$ is an initial segment of $G[i]$, both of these as labeled graphs. (Carefully note the presence of the quantifier here: If $n = 2$ and $G[1]$ is an initial segment of $G'[1]$ and $G'[2]$ is an initial segment of $G[2]$, then $G, G'$ are compatible.)*

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

PROPOSITION 3.29. *Suppose $H_1, H_2$ are prefixes of $G$. Then $H_1$ is consistent with $H_2$.*

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

DEFINITION 3.30 (Merge of two consistent dags). *Let $G, G'$ be consistent witness dags. Then we define the* merge $G \vee G'$ *as follows. If either $G$ or $G'$ has a node $v$ with an extended label $(B, k)$, then we create a corresponding node $w \in G \vee G'$ labeled by $B$. We refer to the* corresponding label *of $w$ as $(B, k)$.*

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

PROPOSITION 3.31. *Suppose that, when forming $G \vee G'$, that $v \in G \vee G'$ has corresponding label $(B, k)$. Then $v$ has extended label $(B, k)$.*

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

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

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

By Proposition 3.31, for every vertex $v \in G$ or $v \in G'$, there is a vertex in $G \vee G'$ with the same extended label. We will abuse notation slightly, so that we refer to this vertex in $H$ also by the name $v$.

PROPOSITION 3.32. *Let $G, G'$ be consistent witness dags and let $H = G \vee G'$. If there is a path $v_1, \ldots, v_l$ in $H$ and $v_l \in G$, then also $v_1, \ldots, v_l \in G$.*

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

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

PROPOSITION 3.33. *Let $G, G'$ be consistent witness dags and let $H = G \vee G'$. Then $H$ is a witness dag and both $G$ and $G'$ are prefixes of it.*

PROOF. Suppose that $H$ contains a cycle $v_1, \ldots, v_l, v_1$, and suppose $v_1 \in G$. Then by Proposition 3.32 the cycle $v_1, \ldots, v_l, v_1$ is present also in $G$, which is a contradiction.

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

Finally, we claim that $G = H(v_1, \ldots, v_l)$ where $v_1, \ldots, v_l$ are the vertices of $G$. It is clear that $G \subseteq H(v_1, \ldots, v_l)$. Now, suppose $w \in H(v_1, \ldots, v_l)$. Then there is a path $w, x_1, x_2, \ldots, x_l, v$ where the vertices $x_1, \ldots, x_l$ lie in $H$ and $v \in G$. By Proposition 3.32, this implies that $w, x_1, \ldots, x_l, v \in G$. So $w \in G$ and we are done. □

PROPOSITION 3.34. *The operation $\vee$ is commutative and associative.*

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

PROPOSITION 3.35. *Suppose $G_1, G_2$ are consistent with each other and with some witness dag $G_3$. Then $G_1 \vee G_2$ is consistent with $G_3$.*

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

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

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

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

The notation suggests that this may depend on the ordering $G_1, \ldots, G_l$, but because of associativity and commutativity this can be well-defined in terms of the unordered set $\{G_1, \ldots, G_l\}$.

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

PROPOSITION 3.36. *The witness dags $G_1, \ldots, G_l$ are pairwise consistent iff there is some witness dag $H$ such that $G_1, \ldots, G_l$ are all prefixes of $H$.*

PROOF. For the forward direction: let $H = G_1 \vee \cdots \vee G_l$. By Proposition 3.33, each $G_i$ is a prefix of $H$. For the backward direction: by Proposition 3.14, any $G_{i_1}, G_{i_2}$ are both prefixes of $H$, hence consistent. □

PROPOSITION 3.37. *Let $G_1, G_2$ be consistent witness dags and $R$ a resampling table. Then $G_1 \vee G_2$ is compatible with $R$ iff both $G_1$ and $G_2$ are compatible with $R$.*

PROOF. For the forward direction: let $v \in G_1$ labeled by $B$. By Proposition 3.32, we have $G_1(v) = (G_1 \vee G_2)(v)$. Thus for $i \in S_B$ we have $|G_1(v)[i]| = |(G_1 \vee G_2)(v)[i]|$. This implies that $X^v_{G_1} = X^v_{G_1 \vee G_2}$. By hypothesis, $B$ is true on $X^v_{G_1 \vee G_2}$ and hence $X^v_{G_1}$. As this is true for all $v \in G_1$, it follows that $G_1$ is compatible with $R$. Similarly, $G_2$ is compatible with $R$.

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

## 3.4. A new parallel algorithm for the LLL

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

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

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

**3.4.1. Collectible witness dags.** The goal of our algorithm is to enumerate the single-sink witness dags (nearly equivalently, the tree-structures.) We will build them up node-by-node. However, in order to do so, we must keep track of a slightly more general type of witness dags, namely, those derived by removing the root node from a single-sink witness dag. Such witness dags have multiple sink nodes, which are all at distance two in the dependency graph. Although this is a much larger set than the set of single-sink witness dags, it is still small enough to enumerate. This is very close to the concept of partial witness trees introduced in [**23**].

DEFINITION 3.38 (Collectible witness dag). *Suppose we are given a witness dag $G$, whose sink nodes are labeled $B_1, \ldots, B_s$. We say that $G$ is* collectible to $B$ *if* $B \sim B_1, \ldots, B \sim B_s$.

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

PROPOSITION 3.39. *Define*

$$W' = \sum_B \tilde{\mu}(B)/P_\Omega(B)$$

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

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

Now

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

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

in the sum. Hence, we have

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

□

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

PROPOSITION 3.40. *Suppose that each bad-event $B$ has $P_\Omega(B) \leq p$ and is dependent with at most $d$ other bad-events. And suppose that the symmetric LLL criterion $epd \leq 1$ is satisfied.*

*Then $W' \leq me$.*

PROOF. Observe that the Asymmetric LLL criterion is satisfied by setting $\mu(B) = eP_\Omega(B)$ for all $B \in \mathcal{B}$. So $W' \leq \sum \frac{eP_\Omega(B)}{P_\Omega(B)} \leq me$. □

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

**3.4.2. Algorithmically enumerating witness dags.** In the Moser-Tardos setting, the witness trees were not actually part of the algorithm but were a theoretical device for analyzing it. In our algorithm, we will operate directly on witness dags.

The following algorithm draws a random resampling table and then builds a list of witness dags compatible with it:

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

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

PROPOSITION 3.41. *Let $G \in F_k$ for any integer $k \geq 1$. Then $G$ is compatible with $R$.*

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

Now for the induction step. Suppose first $G$ was formed by $G = G_1 \vee G_2$, for $G_1, G_2 \in F_{k-1}$. By induction hypothesis, $G_1, G_2$ are compatible with $R$. So by Proposition 3.35, $G$ is compatible with $R$. Second suppose $G$ was formed in step (5), so by definition it must be compatible with $R$. □

PROPOSITION 3.42. *Suppose that $G$ is a collectible witness dag with $k$ nodes compatible with $R$. Then $G \in F_k$.*

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

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

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

Finally, observe that $G = G' \vee G''$. Clearly every vertex in $G$ appears in $G'$ or $G''$. Also, by Proposition 3.29, $G, G'$ are both prefixes of $G$ and hence are compatible with $R$.

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

□

PROPOSITION 3.43. *Suppose that one can determine, in $O(\log n)$ time, whether any given bad-event is true on an assignment of its variables.*

*With high probability, this procedure enumerates all single-sink witness dags compatible with $R$ in time $\tilde{O}(\epsilon^{-1}(\log \epsilon^{-1} n)(\log(W'\epsilon^{-1} n))$ and using $(W'\epsilon^{-1} n)^{O(1)}$ processors.*

PROOF. We have shown that $F_k$ contains all the collectible witness dags compatible with $R$ using at most $k$ nodes. Furthermore, by Corollary 3.20, whp all the

single-sink witness dags compatible with $R$ contain at most $O(\epsilon^{-1}\log(\epsilon^{-1}n))$ nodes. Hence, for $K = O(\epsilon^{-1}\log(\epsilon^{-1}n))$, we have that with high probability $F_K$ contains all such single-sink witness dags.

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

We describe in Section 3.4.5 further details about how these witness dags can be processed. Given that there are $M = (W'n)^{O(1)}$ total dags and that bad-events can be checked in $O(\log n)$ time, we can implement individual steps using $(Mm\epsilon^{-1}n)^{O(1)}$ processors and $\tilde{O}(\log(M\epsilon^{-1}mn))$ time. Observe that $W' \geq m$, so this can be simplified to $(W'\epsilon^{-1}n)^{O(1)}$ etc. □

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

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

for all $i \in [n]$.

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

PROPOSITION 3.44. *With high probability, no bad-event $B \in \mathcal{B}$ is true on the configuration $X^*$.*

PROOF. Suppose that $B$ is true on $X^*$. Now define the witness dag $H$ by adding to $G$ a new sink node $v$ labeled by $B$. Observe that $G$ is a prefix of $H$. By Proposition 3.29 $H, G$ are consistent.

We claim that $H$ is compatible with $R$. By Proposition 3.13, $G$ is compatible with $R$ so this is clear for all the vertices of $H$ except for its sink node $v$. For this vertex, observe that for each $i \in S_B$ we have $X^v_H(i) = R(i, |H[i]|) = R(i, |G[i]| + 1) = X^*(i)$. By Proposition 3.13, this implies that $H(v)$ is compatible with $R$ as well.

So $H(v)$ is a single-sink witness dag compatible with $R$. By Proposition 3.43, with high probability $H(v) \in F_K$. So $H(v)$ is a node of $\mathcal{G}$. Observe that $H(v)$ and all the witness dags $G' \in \mathcal{I}$ are prefixes of $H$. By Proposition 3.36, $H(v)$ is pairwise consistent with all of them. As $\mathcal{I}$ was chosen to be a maximal independent set, this implies that $H(v) \in \mathcal{I}$.

By Proposition 3.33, this implies that $H(v)$ is a prefix of $G$. This implies that $|G[i]| \geq |H(v)[i]|$ for any variable $i$. But for $i \in S_B$ we have $|H(v)[i]| = |H[i]| = |G[i]| + 1$, a contradiction. □

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

THEOREM 3.45. *Suppose we satisfy the Shearer criterion with multiplicative $\epsilon$-slack. Suppose we can check, for any bad-event $B$ and any configuration, if $B$ is true in time $O(\log n)$. Then there is an algorithm to find a configuration avoiding $\mathcal{B}$, running in time $\tilde{O}(\epsilon^{-1}(\log \epsilon^{-1} n) \log(W' \epsilon^{-1} n))$ and using $(W' \epsilon^{-1} n)^{O(1)}$ processors.*

PROOF. By Proposition 3.43, we can enumerate all the single-sink witness dags using $\tilde{O}(\epsilon^{-1}(\log \epsilon^{-1} n)(\log(W' \epsilon^{-1} n))$ time and $(W' \epsilon^{-1} n)^{O(1)}$ processors.

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

Finally, one can form the configuration $X^*$ as indicated in Proposition 3.44 in time $\log(W\epsilon^{-1}n)$ and using $(W\epsilon^{-1}n)^{O(1)}$ processors. This configuration avoids all bad-events, as desired. □

COROLLARY 3.46. *Suppose that we can determine if any bad-event $B$ is true on a given configuration in time $O(\log n)$. Suppose that each bad-event $B$ has $P_\Omega(B) \leq p$ and is dependent with at most $d$ other bad-events. And suppose that the symmetric LLL criterion $ep(1+\epsilon)d \leq 1$ is satisfied, for some $\epsilon > 0$.*

*Then there is an algorithm to find a configuration avoiding $\mathcal{B}$ running in time $\tilde{O}(\epsilon^{-1} \log(mn) \log n)$ and using $(mn)^{O(1)}$ processors.*

PROOF. We have $W \leq \sum_{B \in \mathcal{B}} ep \leq O(m/d)$. By Proposition 3.40, we have $W' \leq me$. Now note that $m \leq nd$, so $W \leq O(n)$.

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

Now apply Theorem 3.45. □

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

Suppose we are given a problem instance on $n$ variables whose distributions are all Bernoulli-$q$, where $q \in [0, 1]$ is a parameter to be chosen. The space $\mathcal{B}$ consists of $\sqrt{n}$ bad-events, each of which is a conjunction of $\sqrt{n}$ variables, and all these events are completely disjoint from each other. The probability of each event is $p = q^{\sqrt{n}}$; we choose $q$ so $p = 1 - \epsilon$.

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

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

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

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

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

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

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

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

Next, we enumerate in parallel over all $B \in \mathcal{B}$. Suppose we are given a fixed $B$ and a fixed $G$; we want to determine if $G$ is collectible to $B$. We can check, in parallel,

whether the sink nodes of $G$ overlap the variables in $B$; this takes time $O(\log n)$ and $n^{O(1)}$ processors. We then check if every sink node of $G$ overlapped in some variable; this takes another $O(\log n)$ time and $n^{O(1)}$ processors.

Finally, we need to determine if we can form a new graph $G'$ by adding a new sink node $v$ labeled $B$ to $G$. In addition to the graph-theoretic structure needed for this, we need to check if $B$ is true on the configuration $X^v$. This will be possible under our assumption that we can check if a bad-event is true in time $O(\log n)$.

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

### 3.5. A deterministic variant

In [**23**], a deterministic parallel (NC) algorithm is given for the LLL. This algorithm requires an additional slack compared to the Parallel MT algorithm (which in turn requires additional slack compared to the sequential algorithm). Although [**23**] gives a general asymmetric criterion, it is quite technical and has many parameters. We will discuss the simpler symmetric setting. In that case, the algorithm of [**23**] requires that

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

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

In the parallel algorithm as given, we assume that $R$ is drawn from a completely independent probability distribution. Such probability distributions have exponentially large support. A key result from [**23**] is that substantially less independence is required.

DEFINITION 3.47. *We say a probability space $\Omega'$ is $k$-wise, $\epsilon$-approximately independent, if for all subsets of variables $X_{i_1}, \ldots, X_{i_k}$, and all possible valuations $j_1, \ldots, j_k$, we have*

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

THEOREM 3.48 ([**36**]). *There are $k$-wise, $\epsilon$-approximately independent probability spaces which have a support of size poly$(\log n, 2^k, \epsilon^{-1})$.*

PROPOSITION 3.49 ([**23**]). *Suppose that $\mathcal{B}$ consists of atomic events.*

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

*Note that this event only depends on the first $R(i, x)$ for $x \leq \log m / \log d$. Hence, this event only depends on polynomially many entries in $R$. By Theorem 3.48, probability spaces with the required level of independence on this number of elements exist which are supported on only $m^{O(1/\epsilon)}$ events*

We note now one major difference between the RNC algorithm of Section 3.4 and the NC algorithm in this section. In Section 3.4, we only enumerated witness dags compatible with $R$. Potentially, there could be many collectible witness dags not compatible with $R$, but we never need to deal with them. For our NC algorithm, we will enumerate all single-sink witness dags, and then we later check whether they are compatible with $R$. The stronger slack condition $epd^{1+\epsilon} < 1$ is needed to ensure that this process remains bounded.

We give a deterministic algorithm to enumerate single-sink dags (with no restriction on their compatibility with any resampling table $R$):

1. Initialize $F_1$ by creating, for each $B \in \mathcal{B}$, a single-node dag with a vertex labeled by $B$.

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

PROPOSITION 3.50. *Suppose that $G$ is a single-sink witness dag containing $k$ nodes. Then $G \in F_k$.*

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

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

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

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

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

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

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

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

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

PROPOSITION 3.51. *The total number of single-sink witness dags containing $k$ vertices is at most $m(ed)^k$. In particular, for $k \leq K = \frac{c \log m}{\epsilon \log d}$ this is $m^{O(1/\epsilon)}$.*

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

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

PROPOSITION 3.52. *Suppose $R$ is drawn from a probability distribution which is $m^{-c/\epsilon}$-approximately, $\log m$-wise independent. Then, for $c$ sufficiently large, the expected total number of single-sink witness dags compatible with $R$ is $m^{O(1)}$.*

PROOF. The event that a $k$-node witness dag is compatible with $R$ is a conjunction of events corresponding to the vertices in $G$. Each such event depends on at most $d$ variables, so in total this is an atomic event depends on at most $kd \leq \log m$ terms.

So, by Definition 3.47, this event also has probability $\leq w(G) + m^{-c/\epsilon}$. Now sum over all single-sink witness dags. The term $w(G)$ sums to $O(m)$ and the term $m^{-c/\epsilon}$ sums to $O(1)$ for $c$ sufficiently large. □

THEOREM 3.53. *There is a deterministic algorithm running in time $\tilde{O}(\epsilon^{-1} \log^2 m)$ and using $m^{O(1/\epsilon)}$ processors to find a configuration avoiding $\mathcal{B}$.*

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

For each $R$, we filter down the set $F_k$ to the smaller set $F'_k$ consisting of witness dags compatible with $R$. We proceed as for the randomized algorithm: we define a graph $\mathcal{G}$, whose nodes correspond to such single-sink witness dags compatible with $R$, with an edge between dags if they are pairwise inconsistent. By Propositions 3.49, 3.52, there is a positive probability that the two events jointly occur:

(1) $F'_K$ contains $m^{O(1)}$ nodes

(2) There are no witness dags outside $F'_K$ that are compatible with $R$.

For a fixed resampling table $R$, we can find a maximal independent subset $\mathcal{I} \subseteq \mathcal{G}$, using time $O(\log^2 m)$ and using $m^{O(1)}$ processors. Let $G = \bigvee \mathcal{I}$. As in Proposition 3.44, defining $X^*(i) = R(i, |G[i]| + 1)$ gives a configuration avoiding all bad-events.

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

CHAPTER 4

# Lopsidependency in the Moser-Tardos framework: Beyond the Lopsided Lovász Local Lemma

As noted by [**65**], the MT algorithm applies to a more restrictive model than the LLL, so Shearer's LLL criterion is not necessarily tight in this restrictive class. [**65**] gave some toy examples of this situation. Notwithstanding this, most researchers have considered Shearer's criterion to be the ultimate form of the LLL. Other forms of the LLL and algorithms such as MT are attempts to match this bound. In this chapter, we do not change the MT algorithm in any way. However, we give a alternate criterion for it to converge. In our opinion, the surprising fact is that this new criterion can go beyond the Shearer criterion. In the LLLL framework, lopsidependency is as good as pure independence; we show that if bad-events agree on a variable, this gives *better bounds* than if they were independent!

We reiterate that the Shearer criterion is the strongest possible criterion that can be given *for the level of generality to which it applies*. Our new criterion depends in a fundamental way on the decomposition of bad-events into variables; it cannot be stated in the language of probability and dependency graphs.

We recall the interpretation of lopsidependency for the variable assignment setting. We assume that all events are atomic, that is, each bad-event has the form $B \equiv (x_{i_1} = j_1) \wedge \cdots \wedge (x_{i_k} = j_k)$. It will be convenient to identify the bad-event $B$ with the set of tuples $\{(i_1, j_1), \ldots, (i_k, j_k)\}$; in this case, if we say that $(i, j) \in B$, we mean that $B$ demands $x_i = j$.

We say that bad events are connected $B \sim B'$ if $B = B'$ or if $B, B'$ disagree on some variable; that is, if we have $(i, j) \in B, (i, j') \in B'$ and $j \neq j$. We use the notation now $(i, j) \sim (i', j')$ iff $i = i', j \neq j'$. Some related notations will be to write

$(i,j) \sim B$ iff there is some $j' \neq j$ with $(i,j') \in B$, and to write $i \sim B$ iff there is some $(i,j) \in B$.

We state our new criterion as follows.

DEFINITION 4.1 (Orderability). *Given an event $E$, we say that a set of bad-events $Y \subseteq N(B)$ is* orderable *to $E$, if either of the conditions hold:*

(O1) $Y = \{E\}$, *or*

(O2) *there is some ordering $Y = \{B_1, \ldots, B_s\}$, with the following property. For each $i = 1, \ldots, s$, there is some $z_i \in E$ such that $z_i \sim B_i, z_i \not\sim B_1, \ldots, B_{i-1}$.*

*Note that $\emptyset$ is orderable to $E$, as indeed it satisfies condition (O2).*

THEOREM 4.2. *In the variable-assignment setting, suppose there is $\mu : \mathcal{B} \to [0, \infty)$ satisfying the following condition:*

$$\forall B \in \mathcal{B}, \mu(B) \geq P_\Omega(B) \sum_{\substack{Y \\ \text{orderable} \\ \text{to } B}} \prod_{B' \in Y} \mu(B')$$

*then the MT terminates with probability 1. The expected number of resamplings is at most $W = \sum_B \mu(B)$.*

The LLLL cannot guarantee under these conditions that a satisfactory configuration even exists; for this reason, we view this criterion as going beyond the LLLL. This criterion is about as easy to work with as the original MT criterion — in some cases, in fact, it can yield significantly simpler calculations. In Section 4.1.3, we compare this to other LLLL criteria.

In Section 4.3, we give some applications of this new criterion. We summarize the most important ones here:

(1) **SAT with bounded variable occurrences** Consider the following problem: we have a SAT instance, in which each clause contains $k$ distinct variables. We are also guaranteed that each variable occurs in at most $L$ clauses,

either positively or negatively. How can large can $L$ be so as to guarantee the existence of a solution to the SAT instance?

As shown in [**43**], the LLLL gives an asymptotically tight bound for this problem, namely $L \leq \frac{2^{k+1}}{e(k+1)}$. However, there still is room for improvement, especially when $k$ is small. As $L$ is growing exponentially, it is arguably the case that large $k$ is not algorithmically relevant anyway. We are able to improve on [**43**] to show that when

$$L \leq \frac{2^{k+1}(1-1/k)^k}{k-1} - \frac{2}{k}$$

then the SAT instance is satisfiable, and the MT algorithm finds a satisfying occurrence in polynomial time. This is always better than the bound of [**43**], and when $k$ is small the improvement can be substantial.

(2) **Hypergraph coloring**. Suppose we are given a $k$-uniform hypergraph, in which each vertex participates in at most $L$ edges. We wish to $c$-color the vertices, so that no edge is monochromatic (all vertices receiving the same color). This problem was in fact the inspiration for the original LLL [**34**]. There are many types of graphs and parameters for which better bounds are known, but the LLL gives very simple constructions and also provides the strongest bounds in some cases (particularly when $c, k$ are fixed small integers). Strangely, depending on whether $c$ or $k$ is large, one can obtain better bounds using the standard LLL or the LLLL. Thus one can show the bounds:

(4) $$L \leq \frac{c^k}{k} \max\left(\frac{(1-1/k)^{k-1}}{c}, \frac{1}{(c-1)e}\right)$$

Our approach gives the simpler and stronger criterion:

$$L \leq \frac{c^k(1-1/k)^{k-1}}{k(c-1)}.$$

Our new criterion is always better than (4), interpolating smoothly between the regimes when $c$ or $k$ is large. This illustrates an advantage of our technique — despite the daunting form of our new LLLL criterion, in practice it typically gives formulas which are more computationally tractable.

**Comparison of Shearer criterion to Moser-Tardos.** It is challenging to directly compare the Shearer criterion with the MT algorithm, because they apply to such different contexts: generic probability spaces in the former case and variable configurations in the latter. However, in Section 4.4, we analyze $k$-SAT satisfiability using the Shearer criterion. We show that the analysis of [**43**], which used the asymmetric LLL, could not be much improved by using a stronger form of the LLL. In particular, our proof directly based on our new MT criterion is stronger than would be possible from Shearer's criterion, let alone the LLL.

**A new parallel algorithm.** We have discussed in Chapter 2 how the LLL, but not the LLLL, has a parallel algorithm. We remedy this situation in Section 4.2 by introducing a new parallel randomized algorithm for the variable-assignment LLLL, which requires only a multiplicative slack compared to our new criterion for the sequential algorithm. (Showing that this algorithm is compatible with the new LLLL criterion requires non-trivial arguments).

THEOREM 4.3. *Suppose there is* $\mu : \mathcal{B} \to [0, \infty)$ *satisfying the following condition:*

$$\forall B \in \mathcal{B}, \mu(B) \geq (1+\epsilon) P_\Omega(B) \sum_{Y \text{ orderable to } B} \prod_{B' \in Y} \mu(B')$$

*then our new parallel algorithm algorithm terminates with probability 1. Suppose that the size of each bad-event is at most* $M$*. Then our parallel algorithm terminates in time* $\epsilon^{-1} M \log W \log^{O(1)} n(M + \log^{O(1)} m)$ *and* $(nm)^{O(1)}$ *processors with high probability.*

We list a few applications of these new parallel algorithms:

(1) **SAT with bounded variable occurrences** We have a SAT instance, in which each clause contains at least $k$ variables. We are also guaranteed that each variable occurs in at most $L$ clauses, either positively or negatively. Then, under the condition $L \leq \frac{2^{k+1}(1-1/k)^k}{(k-1)(1+\epsilon)} - \frac{2}{k}$ the parallel MT algorithms find a satisfying assignment in time $\frac{k^{O(1)} \log^{O(1)} n}{\epsilon}$.

(2) **Hypergraph coloring**. Suppose we are given a $k$-uniform hypergraph, in which each vertex participates in at most $L$ edges. We wish to $c$-color the vertices, so that no edge is monochromatic (all vertices receiving the same color). Then, under the condition $L \leq \frac{c^k(1-1/k)^{k-1}}{(1+\epsilon)(c-1)k}$ then the parallel MT algorithm finds a good coloring in time $\frac{k^{O(1)} \log^{O(1)} n}{\epsilon}$.

## 4.1. The variable-assignment LLLL

**4.1.1. Overview of our new analysis.** As we have said, we do not change the MT algorithm in any way. The only change is to the analysis. In particular, we dispense with the use of the resampling table which played a prominent role in the proof of [**87**]. (See Lemma 2.8). Recall that the idea of the resampling table is that, at the very beginning of the algorithm, you draw an infinite list of all the future values for each variables. Then resampling table entry $R(i, j)$ gives the $j$th value for each variable $X_i$. Initially, you set $X_i = R(i, 1)$; when you need to resample $i$, you set $X_i = R(i, 2)$, and so forth. After drawing $R$, the remainder of the MT algorithm becomes deterministic. One can determine, for each tree-structure $\tau$, necessary conditions to hold on $R$ in order for $\tau$ to appear.

In the MT algorithm, the choice of which bad-event to resample can be arbitrary, and can even be under the control of an adversary. In fact, MT shows something even stronger than this: even if the choice of which bad-event to resample depends on $R$, then the MT algorithm must still converge with high probability. Thus, the LLLL criterion is strong enough to show convergence even the resampling is determined by a *clairvoyant adversary*.

Our analysis is also based on witness trees, but we assume that variables are assumed to be resampled in an *on-line* fashion. The choice of which bad-event to resample can be arbitrary, but must depend *solely on the prior state (not the future state) of the system.*

Because we impose this restriction on the resampling rule, we can use stochasticity to analyze our witness trees, in a way which is not possible with the MT framework. The basic idea is that, whenever we resample some bad-event $B$, the distribution for the new values for its variables is the same as the law of $\Omega$, *even conditional on all prior state.* This is simply not true in the MT framework: the choice of resampling rule could produce a dependency on the future.

We will eventually take a union-bound over tree-structures, so it is critical to prune the space of witness trees as much as possible. In other words, we will need the most succinct possible explanation of each resampling. Let us consider a simple example of how we can use our stronger stochasticity assumption to analyze more succinct witness trees. Suppose that we have some bad-event $B$, including $(i, j)$, and we want to explain why we eventually resampled $X_i = j$. Suppose there were two earlier events $B_1, B_2$ which included $(i, j')$. These events would be placed as children of $B$ in the standard MT witness tree. This means that we encounter $B_1, B_2$ (in an unspecified order), and when we encounter the second one we select $X_i = j$.

Now, it is not necessary to describe the earlier of the two events $B_1, B_2$ here. It would be sufficient to only include the latter, because that is when we are making the key prediction (namely, that $X_i$ is resampled to value $j$). By only retaining the latest occurrence of each variable, we still have all the information we need to deduce the resamplings. We know that, whenever the resample the latter of $B_1, B_2$, the new values for the variables in it must have the same distribution as in $\Omega$. The reason that this is true is that the choice of whether to resample $B_1$ or $B_2$ first *cannot depend on the new values of the variables.* This information is now "compressed" more tightly, and we will have to work much harder to show that it is sufficient.

**4.1.2. Forming witness trees.** The key principle behind our new criterion is the following: When building witness trees, we will maintain the following key invariant: for any node $v$ in the tree labeled by $B$, the children of $v$ receive distinct labels $B_1, \ldots, B_s$ such that $\{B_1, \ldots, B_s\}$ is an orderable set for $B$.

We now describe how to form a witness tree for a resampling at time $T$. Suppose we have listed the execution log consisting of all the bad-events that were resampling $B_1, \ldots, B_T$. This is referred to as the *execution log.* Starting at time $T-1$, we proceed backward through the execution log. For each bad-event $B$ encountered, we see if there is some node $v \in \tau$ for which $B$ is eligible. If so, we add $B$ to the *deepest* such position (breaking ties arbitrarily). We give the following more precise definition of eligibility:

DEFINITION 4.4 (Eligibility). *Suppose we have a tree-structure $\tau$ and a node $v \in \tau$ labeled by $B$. Suppose the children of $v$ receive distinct labels $B_1, \ldots, B_s$. Then we say a bad-event $B'$ is* eligible *for $v$ if $B' \neq B_1, \ldots, B_s$ and if $\{B_1, \ldots, B_s, B'\}$ is orderable for $B$.*

One simple definition we will use often:

DEFINITION 4.5. *Consider any variable $i$, and consider a tree-structure $\tau$ with a node $v$. We say that $v$* involves *$i$, if $v$ is labeled by some bad-event $B$, and $(i, j) \in B$ for some $j$.*

We list some easy properties of the witness trees produced in this manner:

PROPOSITION 4.6. (1) *Consider any bad-event $B$. Consider the leaf nodes of $\hat{\tau}$ labeled by $B$; all such nodes must have* distinct *depths in the tree.*

(2) *Consider any variable $i$, and, among all the nodes $v \in \hat{\tau}$ involving $i$, consider the set of such nodes which are greatest depth in the tree. While it is possible that there are multiple such nodes $v_1, \ldots, v_r$, all such nodes must be labeled by $B_1, \ldots, B_r$ which* agree *on variable $i$.*

PROOF. The earlier bad-event would have been eligible to be a child of the later bad-event, and hence would have been placed either there or deeper in the tree. □

In light of Proposition 4.6, we may define the *active value* for each variable:

DEFINITION 4.7 (The active value of a variable). *Consider any variable $i$, and, among the set of nodes $v \in \tau$ involving $i$, consider the nodes at greatest depth in the tree. All such nodes contain $(i, j)$ for some common value $j$. We denote by $A_i(\tau)$, the* active value *of variable $i$, by this common value $j$.*

*If variable $i$ does not appear in $\tau$, we define $A_i(\tau) = \top$, the sure value. By convention, we use $X_i = \top$ as a shorthand for the sure event (the entire probability space). For example, $P_\Omega(X_i = \top) = 1$.*

We note that these types of witness trees look very different from the standard MT construction. For example, the layers in the tree (and even the children of a common parent) do not necessarily form an independent set; there can be multiple copies of a single bad-event in a given layer.

Suppose we are given a tree $\tau$ and a time $t_1$; we want to estimate the probability that $\hat{\tau}^{t_1} = \tau$. This is the key to the MT proof strategy. (We will omit the superscript $t_1$ in the following; it should be understood.) We can imagine running the MT algorithm and see whether, so far, it appears that it is still possible for $\hat{\tau}^{t_1} = \tau$. This is a kind of dynamic process, in which we see what conditions are still imposed in order to achieve this tree. One key point in our rule for forming witness trees is that, as we run the MT algorithm, we will be able to deduce not just $\hat{\tau}^{t_1}$ but also $\hat{\tau}^{t_1}_{\geq t_0}$ for all $t_0 \geq 1$.

PROPOSITION 4.8. *Suppose we are given the partial witness tree $\hat{\tau}_{\geq t}$, and we encounter a bad-event $B$ at time $t$. Then $\hat{\tau}_{\geq t+1}$ is uniquely determined, according to the following rule: if there is a leaf node labeled by $B$, select the deepest such leaf node $v$ (by Proposition 4.6 it is unique) and we have $\hat{\tau}_{\geq t+1} = \hat{\tau}_{\geq t} - v$. Otherwise we have $\hat{\tau}_{\geq t+1} = \hat{\tau}_{\geq t}$.*

PROOF. First, suppose that $\hat{\tau}_{\geq t}$ did contain such a node $v$. It must be that $v \notin \hat{\tau}_{\geq t+1}$. For, if so, then when forming $\hat{\tau}_{\geq t}$ from $\hat{\tau}_{\geq t+1}$, we would have placed $B$ as a child of $v$; that is, $\hat{\tau}_{\geq t}$ would include an additional copy of $B$. So $\hat{\tau}_{\geq t+1}$ is missing the node $v$ from $\hat{\tau}_{\geq t}$. As each time step can only affect a single node in the witness tree, it must be that $\hat{\tau}_{\geq t+1} = \hat{\tau}_{\geq t} - v$.

Second, suppose that $\tau$ contained no such node $v$. When forming $\hat{\tau}_{\geq t}$ from $\hat{\tau}_{\geq t+1}$, we either make no changes or add a single node labeled by $B$. In the latter case, $\hat{\tau}_{\geq t}$ would contain a leaf node labeled by $B$, which has not occurred. Hence it must be that $\hat{\tau}_{\geq t} = \hat{\tau}_{\geq t+1}$ as claimed. □

The other key point is that, from the partial tree $\hat{\tau}_{\geq t}$, we can deduce some information about the variables:

PROPOSITION 4.9. *Consider any variable $i$. At time $t$ of the MT algorithm, we must have $X_i = A_i(\hat{\tau}_{\geq t})$.*

PROOF. Suppose $B$ is a node of greatest depth containing variable $i$, and we have $(i, j) \in B$, where $j = A_i(\hat{\tau}_{\geq t})$. Suppose $X_i = j' \neq j$ at time $t$.

In order to include $B$ in the witness tree $\hat{\tau}$, we must eventually resample $B$, which implies that eventually we must have $X_i = j$. As $X_i = j'$ at time $t$, this implies that we must first encounter some bad-event $B' \ni (i, j')$. But then $B'$ would be eligible to be placed as a child of $B$, and so would be placed there or lower. This contradicts that $B$ is the greatest-depth occurrence of variable $i$. □

These propositions together allow us to prove the Witness Tree Lemma:

LEMMA 4.10 (Witness Tree Lemma). *Let $\tau$ be a tree-structure with nodes labeled $B_1, \ldots, B_s$. Then the probability of that $\tau$ appears is bounded by*

$$P(\tau \text{ appears}) \leq w(\tau)$$

PROOF. The first step of the MT algorithm is to draw all the variables independently from $\Omega$. We may consider fixing the variables to some arbitrary (not random)

values, and allowing the MT algorithm to run from that point onward. We refer to this as *starting at an arbitrary state of the MT algorithm.* We prove by induction on $\tau$ the following: for any tree-structure $\tau$, and starting at an arbitrary state of the MT algorithm, then we have

$$P(\tau \text{ appears}) \leq \frac{\prod_{B \in \tau} P_\Omega(B)}{\prod_i P_\Omega(X_i = A_i(\tau))} \tag{5}$$

First, the base case when $\tau = \emptyset$. Then this is vacuously true, as the RHS of (5) is equal to 1.

Next, for the induction. Suppose that $\hat{\tau}^t = \tau$ for some $t > 0$. Then, a necessary condition of this is that, at some time $t' < t$, we must resample some $B \in \mathcal{B}$ such that a leaf node of $\tau$ is labeled by $B$. (If not, then $\hat{\tau}^t$ could never acquire such a leaf node.) Suppose that $t'$ is the earliest such time and that $v$ is a leaf node labeled by $B$. We condition now on a specific value for $t'$ and $B$.

For each variable $i$ appearing in $B$, we must resample variable $i$ to take on value $A_i(\tau - v)$ (recall our convention that if $A(\tau - v) = \top$, then this is automatically true.) This has probability $P_\Omega(X_i = A_i(\tau - v))$. Next, starting at the state of the system at time $t'+1$, we must satisfy $\hat{\tau}^{t-1} = \tau - v$. We need to estimate the probability that this occurs, conditional on a fixed choice of $t', B$. Crucially, the inductive hypothesis gives an upper bound on this probability *conditional on any state of the MT algorithm.* In particular, this upper bound applies even when we condition on $t', B$. Thus, we may multiply the two probabilities to obtain:

$$\begin{aligned} P(\tau \text{ appears}) &\leq \prod_{i \sim B} P_\Omega(X_i = A_i(\tau - v)) \frac{\prod_{B' \in \tau - v} P_\Omega(B')}{\prod_i P_\Omega(X_i = A_i(\tau - v))} \\ &= \frac{\prod_{B' \in \tau} P_\Omega(B')}{P_\Omega(B) \prod_{i \not\sim B} P_\Omega(X_i = A_i(\tau - v))} \end{aligned}$$

If $i \not\sim B$ then $A_i(\tau - v) = A_i(\tau)$, while for each $i \sim B$ we have $(i, A_i(\tau)) \in B$. Hence the denominator is equal to $\prod_i P_\Omega(X_i = A_i(\tau))$ and the induction holds.

Next, we claim that the probability $\tau$ appears is at most $w(\tau)$. (Note that this differs subtly from the induction; in the induction, we are showing a bound which applies to any starting state of the system; here, we are claiming that this bound holds when the MT algorithm begins with a random initialization.) To see this, note that a necessary condition for $\tau$ to appear is that in the initial sampling of all relevant variables, each variable $i$ must take on value $A_i(\tau)$. Conditional on this event, the probability that $\tau$ appears is still given by the inductive hypothesis, so we have

$$P(\tau \text{ appears}) \leq \prod_i P_\Omega(X_i = A_i(\tau)) \times \frac{\prod_{B \in \tau} P_\Omega(B)}{\prod_i P_\Omega(X_i = A_i(\tau))} = w(\tau)$$

□

Finally, we show that each event in the execution log of the MT algorithm has a distinct witness tree. This is almost a triviality in the standard analysis of the MT algorithm, but here it is surprisingly subtle. For instance, there may be multiple resamplings of a bad-event $B$, and the later occurrences may have smaller witness trees. Nevertheless, all such trees are unique:

PROPOSITION 4.11. *Let $t_1 < t_2$; then $\hat{\tau}^{t_1} \neq \hat{\tau}^{t_2}$.*

PROOF. Suppose $\hat{\tau}^{t_1} = \hat{\tau}^{t_2}$. Proposition 4.8 shows that the sequence of trees $\hat{\tau}_t$ is uniquely determined by the original value $\hat{\tau}$ and by the sequence of resamplings encountered the execution of the MT algorithm. As $\hat{\tau}^{t_1} = \hat{\tau}^{t_2}$ initially, we must have that $\hat{\tau}^{t_1}_{\geq t} = \hat{\tau}^{t_2}_{\geq t}$ for all $t \geq 1$. But now substitute $t = t_2$; in this case, $\hat{\tau}^{t_1}_{\geq t}$ is the null tree and $\hat{\tau}^{t_2}_{\geq t}$ consists of a single node. So this is a contradiction. □

THEOREM 4.12. *Suppose there is $\mu : \mathcal{B} \to [0, \infty)$ satisfying the following condition:*

$$\forall B \in \mathcal{B}, \mu(B) \geq P_\Omega(B) \sum_{Y \text{ orderable to } B} \prod_{B' \in Y} \mu(B')$$

*then the MT terminates with probability 1. The expected number of resamplings of a bad-event $B$ is at most $\mu(B)$.*

PROOF. First, by induction on tree-height, one can show that the total weight of all tree-structures rooted in a bad-event $B$, is at most $\mu(B)$. This follows since the children of the root node form an orderable set for $B$.

Next, by Proposition 4.11, each resampling of $B$ corresponds to a distinct witness tree. Hence, by the Lemma 4.10, the expected number of witness trees rooted in $B$ is at most the sum of the weights of all such trees. Hence the expected number of resamplings of $B$ is at most $\mu(B)$. □

**4.1.3. Comparison to other LLL criteria.** We have already encountered Pegden's criterion for the MT to converge. This criterion works for both the canonical dependency graph and the canonical lopsidependency graph. One counter-intuitive aspect is that sometimes a *denser* dependency graph gives a stronger criterion. (For the Shearer criterion, this can never occur). In particular, ignoring lopsidependency can give better bounds.

Strictly speaking, Pegden's criterion is incomparable to ours. However, in practice, the usual method of accounting for independent sets of neighbors comes from analyzing, for each variable $i$, the total set of all bad-events in which $i$ could participate. An independent set of neighbors of $B$ can contain one or zero bad-events involving each variable. Any higher-order interaction — such as finding groups of variables participating jointly in bad-events — is usually too complicated to analyze and is disregarded.

When we account for the dependency graph solely in terms of variable intersection, then we can replace the somewhat confusing concept of "orderable set" with a simpler (albeit slightly weaker) notion.

DEFINITION 4.13. *Given an event $E$, we say that a set of bad-events $Y \subseteq N(B)$ is* assignable *to $E$, if either $Y = \{E\}$,* or *there is an* injective *function $f : Y \to E$, such that for all $B \in Y$, we have some $B \ni z \sim f(B) \in E$*

PROPOSITION 4.14. *If $Y$ is orderable to $B$, then it is assignable to $B$ (but not necessarily vice-versa)*

PROOF. Let $Y = \{B_1, \ldots, B_s\}$, so that for each $i = 1, \ldots, s$ there is some $z_i \in E$ with

$$z_i \sim B_i \qquad z_i \not\sim B_1, \ldots, B_{i-1}$$

Now define $f(B_i) = z_i$. We claim that $f$ is injective. For, suppose $z_i = z_j$ and $i < j$. Then $z_i \sim B_i$, so $z_j \sim B_i$, which is a contradiction. □

When we sort bad-events by their variables, we obtain the following criteria; these are respectively the LLLL criterion and Pegden's LLL criterion:

PROPOSITION 4.15. (1) *If for all bad-events $B$ we have*

$$\mu(B) \geq P_\Omega(B)\Big(\mu(B) + \prod_{(i,j)\in B} \prod_{j'\neq j} \prod_{B'\ni(i,j')} (1+\mu(B))\Big)$$

*then the MT algorithm converges.*

(2) *If for all bad-events $B$ we have*

$$\mu(B) \geq P_\Omega(B) \prod_{(i,j)\in B} \Big(1 + \sum_{j'} \sum_{B'\ni(i,j')} \mu(B')\Big)$$

*then the MT algorithm converges.*

Our criterion blends these two conditions and is stronger than either of them:

PROPOSITION 4.16. *If for all bad-events $B$ we have*

$$\mu(B) \geq P_\Omega(B)\Big(\mu(B) + \prod_{(i,j)\in B} (1 + \sum_{j'\neq j} \sum_{B'\ni(i,j')} \mu(B'))\Big)$$

*then the MT algorithm terminates with probability 1; the expected number of resamplings of $B$ is at most $\mu(B)$.*

PROOF. For the bad-event $B$, we have the criterion

$$\mu(B) \geq P_\Omega(B) \sum_{\substack{Y \text{ assignable} \\ \text{to } B}} \prod_{B' \in Y} \mu(B')$$

We enumerate the assignable sets $Y$ as follows. First, we may take $Y = \{B\}$; this accounts for the term $\mu(B)$ in the RHS of Proposition 4.16. Next, for each variable $(i, j) \in B$, we may select either zero or one bad-event $B' \ni (i, j')$ for some $j' \neq j$. These account for respectively the terms 1 and $\sum_{j' \neq j} \sum_{B' \ni (i,j')} \mu(B')$ in Proposition 4.16. □

It is in this sense that we view our criterion as being stronger than the original MT lopsidependency criterion and stronger than Pegden's criterion.

### 4.2. Parallel algorithm for MT

We have seen in Chapter 2 a parallel algorithm for the LLL. This algorithm depends on the fact that, in the standard MT framework, bad-events which are unconnected do not share any variables. Hence they do not interact in any way and can be resampled in parallel. This is no longer the case for the LLLL. MT algorithm; so in that case, frustratingly, we do not have any corresponding parallel algorithms.

In this section, we introduce a new parallel algorithm corresponding to the lopsidependent MT setting, which achieves our new criterion up to a multiplicative slack. We assume that each bad-event uses at most $M \leq \text{polylog}(n)$ terms. We also suppose that the number of bad-events is polynomially bounded, although this can be relaxed quite a bit. Finally, we require a multiplicative slack in the LLLL criterion.

Here is the basic idea. Suppose we have a large number of bad-events which are currently true. Due to the LLLL criterion, there may be many "unconnected" bad-events which are simultaneously true, yet they intersect in variables and cannot be resampled in parallel. However, suppose we resample a given variable; with good probability, it will change its value, thereby falsifying *all* of the bad-events which contain it, even those we did not explicitly resample.

This argument can break down if we have $p_{ij} \approx 1$ for any variable $i$ and value $j$ (this situation is rare; see Section 4.3.5 for an example). In that case, resampling the variable $i$ a single time is not likely to flip its value; we must resample it multiple times. Much of the complication of our parallel algorithm comes from dealing with this somewhat pathological case. We will begin by stating a simple parallel algorithm which assumes $p_{ij} < 1 - \Omega(1)$ for all $i, j$; we then modify it to remove this condition.

**4.2.1. The parallel algorithm: warm-up exercise.** We present the following Parallel MT Algorithm (Simplified):

1. Draw all variables independently from the distribution $\Omega$.
2. While there is some true bad-event, repeat the following for rounds $t = 1, 2, \ldots,$:
    3. Let $\mathcal{V}_{t,1}$ be the set of bad-events which are true at the beginning of round $t$.
    4. Repeat the following for a series of sub-rounds $s = 1, 2, \ldots,$ until $V_{t,s} = \emptyset$.
        5. Select a maximal disjoint set $I_{t,s} \subseteq V_{t,s}$. (This can be done using a parallel MIS algorithm).
        6. Resample all $B \in I_{t,s}$.
        7. Update $V_{t,s+1}$ as: $V_{t,s+1} = V_{t,s} - I_{t,s} -$All bad events which are no longer true.

THEOREM 4.17. *Suppose that we satisfy the condition*

$$\forall B \in \mathcal{B}, \mu(B) \geq P_\Omega(B)(1+\epsilon) \sum_{Y \text{ orderable to } B} \prod_{B' \in Y} \mu(B')$$

*Suppose further that we satisfy the condition*

$$\forall i, j, P_\Omega(X_i = j) < 1 - \psi$$

*Then whp the Parallel MT Algorithm (Simplified) terminates in time* $\psi^{-1}\epsilon^{-1} \log W \log^{O(1)}(nm)$ *using* $(nm)^{O(1)}$ *processors.*

PROOF. We provide only a sketch, as we will later introduce a more advanced algorithm. In each round $t, s$, note that every bad-event $B \in \mathcal{V}_{t,s}$ contains some resampled variable. Such a variable switches to a new value with probability $\geq \psi$, in which case $B$ is removed from $V_{t,s+1}$. Hence the expected size of $\mathcal{V}_{t,s}$ is decreasing as $(1 - 1/\psi)^s$. So, for $s = \Omega(\psi^{-1} \log n)$, we have $\mathcal{V}_{t,s} = \emptyset$ and the round $t$ is done.

Next, suppose that a bad-event is resampled in round $t$. One can show that the witness tree for this resampling must have height $t$ exactly. One can also compute the total weight of all such trees, and show that this is decreasing as $(1+\epsilon)^{-t}$. Taking a union-bound over such trees, this implies that the probability of having $\geq t$ rounds is at most $(1 + \epsilon)^{-t} W$.

This implies that, whp, our algorithm requires $\psi^{-1}\epsilon^{-1} \log^{O(1)} n \log W$ rounds. In each round, one must select an MIS of the currently-true bad-events, which takes at most $\log^{O(1)} m$ time and $m^{O(1)}$ processors. □

It will take a lot more work to drop the dependency of the running time on $\psi$. To do this, we will need to resample a variable multiple times in the round. Thus, we must replace the step of selecting a maximal disjoint set of bad-events with a maximal set in which no variable occurs too many times (which depends on its probabilities $p_i$). We must also deal with the possibility that we get inconsistent results when we resample a variable multiple times in a round.

Before we move on to the general case, we will need a subroutine to solve a problem we refer to as the *vertex-capacitated maximal edge packing problem*. We believe this may be a useful building block for other parallel algorithms.

### 4.2.2. Vertex-capacitated maximal edge packing.

DEFINITION 4.18. *Suppose we are given a hypergraph $G$, with $m$ edges of size $\leq k$, on a vertex set $V$. For each $v \in V$, we are given a* capacity *$C_v$ in the range $\{0, \ldots, m\}$. We wish to select a subset $L \subseteq E$ of the edges, with the property that each vertex appears in at most $C_v$ edges of $L$, and such that $L$ is a maximal subset*

*of $E$ with that property. Such a set $L$ is referred to as a* vertex-capacitated maximal edge packing (VCMEP).

Such a set can be found easily by a sequential algorithm. Note that if $C_v = 1$ for all $v$, this is equivalent to finding a maximal independent set of the line graph of $G$.

THEOREM 4.19. *There is parallel algorithm to find a VCMEP in time $k \times \log^{O(1)}(m+n)$.*

PROOF. We will repeatedly add edges until we have reached such a maximal set. At round $i$, suppose we have selected so far edges $L_i$, and we begin with $L_0 = \emptyset$.

Now form the residual graph and residual capacities; we abuse notation so that these are also denoted $G, C$. One can form an integer program corresponding to the vertex-capacitated *maximum* edge packing (i.e. packing of highest cardinality) for the residual. We let $M_i$ denote the size of the maximum packing which can be obtained by extending $L_i$. This integer program has variables $x_f$ corresponding to each edge $f \in G$, along with constraints that $\sum_{v \in f} x_f \leq C_v$ for each vertex $v$. Now relax the integer program to a positive linear program. As shown by [**79**], there is a parallel algorithm running in time $\log^{O(1)}(n+m)$ which can find a solution $x'$ which is at least $(1-\epsilon)$ times the optimum solution, where $\epsilon > 0$ is some sufficiently small constant. In turn, this solution is at least $(1-\epsilon)(M_i - |L_i|)$.

We now round this fractional solution $x'$ as follows: each edge is selected with probability $x'_f/(2k)$; if any vertex constraint $v$ is violated, then all edges containing $v$ are de-selected. We define $L_{i+1}$ to be $L_i$ plus any selected edges.

Define the potential function $\Phi_i = M_i - |L_i|$. Note that if $\Phi_i = 0$, then $L_i$ must be a maximal set of edges. We claim that, conditional on the state at the beginning of round $i$, we have $\mathbf{E}[\Phi_{i+1}] \leq (1 - \Omega(1/k))\Phi_i$.

For, consider some edge $f$; it is selected with probability $x'_f/(2k)$; suppose we condition on that event. Consider any vertex $v \in f$. The expected number of times that other edges incident to $v$ are selected is $\sum_{f' \ni v, f' \neq f} x'_f/(2k) \leq (C_v - x'_f)/(2k)$.

By Markov's inequality, the probability that the actual number exceeds $C_v$, in which case $f$ is de-selected, is at most $\frac{C_v - x'_f}{2kC_v}$. Hence the total probability that $f$ is selected is at least

$$\begin{aligned} P(f \text{ selected}) &\geq \frac{x'_f}{2k}(1 - \sum_{v \in f} \frac{C_v - x'_f}{2kC_v}) \\ &\geq \frac{x'_f}{2k}(1 - k \times \frac{1}{2k}) \geq \frac{x'_f}{4k} \end{aligned}$$

Summing over all such edges, we have that

$$\begin{aligned} \mathbf{E}[\Phi_{i+1}] &= \mathbf{E}[M_{i+1}] - \mathbf{E}[L_{i+1}] \\ &\leq M_i - |L_i| - \sum_f \frac{x'_f}{4k} \\ &\leq M_i - |L_i| - (\frac{(1-\epsilon)|M_i| - L_i}{4k}) \\ &\leq \Phi_i(1 - \Omega(1/k)) \end{aligned}$$

Hence, for $i \geq \Omega(k \log(m+n))$, we have $\mathbf{E}[\Phi_i] \leq n^{-\Omega(1)}$. This implies that $\Phi_i = 0$ with high probability, which in turn implies that $L_i$ is a maximal packing with high probability. □

**4.2.3. The parallel algorithm.** We now present our full parallel algorithm. We will suppose that each bad-event uses at most $M$ terms. We also suppose that the number of bad-events is polynomially bounded, although this can be relaxed quite a bit. Finally, we suppose there is a slack in the LLL condition,

$$\forall B \in \mathcal{B}, \mu(B) \geq P_\Omega(B)(1+\epsilon) \sum_{\substack{Y \text{ orderable} \\ \text{to } B}} \prod_{B' \in Y} \mu(B')$$

1. Draw all variables independently from the distribution $\Omega$.
2. While there is some true bad-event, repeat the following for rounds $t = 1, 2, \ldots,$:

3. Let $\mathcal{V}_{t,1}$ be the set of bad-events which are true at the beginning of round $t$. Let $a_i$ be the value of variable $X_i$ at the beginning round $t$. Note that each bad-event in $\mathcal{V}_{t,1}$ is a conjunction of terms $X_i = a_i$. For notation throughout the rest of this algorithm, for each variable $i$ let $q_i = P_\Omega(X_i \neq a_i)$.
4. Repeat the following for a series of sub-rounds $s = 1, 2, \ldots$, until $V_{t,s} = \emptyset$.
    5. View $V_{t,s}$ as a hypergraph, whose vertices correspond to variables and whose hyper-edges correspond to bad-events. For each variable $i$, define the capacity $C_i = \lceil \frac{1}{Mq_i} \rceil$. Find a VCMEP $I_{t,s} \subseteq V_{t,s}$.
    6. For each $B \in I_{t,s}$ and each variable $X_i \in B$, we draw a resampling value $x_{B,i}$, drawn from $\Omega$. This represents that *if* we decide to resample $B$, then we will choose to set variable $X_i$ equal to $x_{B,i}$.
    7. For each $B \in I_{t,s}$ choose a random $\rho(B)$ independently from the real interval $[0, 1]$. We think of $\rho(B)$ as the priority of $B$; we will resample the bad-events in the order of increasing $\rho$. Construct the undirected graph $G_{t,s}$ whose vertices correspond to elements of $I_{t,s}$, and where there is an edge from $B_1$ to $B_2$ if $\rho(B_1) < \rho(B_2)$ and $B_1, B_2$ both share a variable $i$ and we have $x_{B_1,i} \neq a_i$.
    8. Find the lexicographically-first MIS (LFMIS) $I'_{t,s} \subseteq I_{t,s}$ of the graph $G_{t,s}$, with respect to the order $\rho$. (We will say more about this step later)
    9. For each variable $X_i$, if there is some $B \in I'_{t,s}$ with $x_{B,i} \neq a_i$, set $X_i = x_{B,i}$ (by the way that $G_{t,s}$ is constructed, there can be at most one such $B$ for each variable $i$); we say such that variable $i$ is *switched*; otherwise leave $X_i = a_i$.
    10. Update $V_{t,s+1}$ as $V_{t,s+1} = V_{t,s} - I_{t,s}-$ all bad events containing a switched variable

This algorithm is quite intricate to analyze. The most important aspect of this algorithm is that it simulates the sequential MT algorithm, with a particular choice of which bad-event to resample at each time. Because of this fact, one can build witness trees for each resampling and show that a Witness Tree Lemma still holds:

PROPOSITION 4.20. *Suppose that as we execute the Parallel MT algorithm, we build a sequential execution log as follows: within each round $t$ and sub-round $s$, we sort the elements of $I'_{t,s}$ in increasing order of $\rho$. We then list, in this order, elements of $I'_{t,s}$.*

*For this sequential execution log, one can still build witness trees as before. Then the Witness Tree Lemma, Lemma 2.8, still holds for such trees.*

PROOF. We can view the Parallel Algorithm as simulating the sequential MT algorithm as follows. Instead of resampling the bad-events in parallel, we suppose that as we proceed in increasing order $t, s, \rho$, we come across various bad-events $B \in I'_{t,s}$. We refer to the position of each resampling in the execution log as its "time". Each such bad-event is true at the time we encounter it; we then resample any variable $i$ it involves by setting $X_i = x_{B,i}$. As the sequential MT algorithm allows an arbitrary choice of which bad-event to resample, the parallel MT algorithm is thus a special case of the sequential MT algorithm.

One crucial detail of this algorithm is that when we make our resamplings $X_i = x_{B,i}$, the random variables $x_{B,i}$ are independent of any random variables we encountered in earlier times. This is why we choose the lexicographically-first MIS of $I_{t,s}$; in this way, one can see that the event "$B \in I'_{t,s}$" is independent of all events involving $B'$ with $\rho(B') > \rho(B)$ and the values of random variable $x_B$ are independent of all events involving $B'$ with $\rho(B') < \rho(B)$.

Thus, the parallel MT algorithm effectively becomes a stochastic process, with increasing $s, t, \rho$ playing the part of time. Hence Lemma 4.10 still holds. □

This Proposition 4.20 will serve to bound the number of rounds $t$ for our algorithm. The analysis of the run-time of each individual round is more complicated. We will bound this through the following series of Propositions.

PROPOSITION 4.21. *The LFMIS $I'_{t,s}$ can be found whp in time $O(\frac{\log n}{\log \log n})$.*

PROOF. In general, the problem of finding the LFMIS is P-complete [**29**], hence we do not expect a generic parallel algorithm for this. However, what saves us it that the ordering $\rho$ and the graph $G_{t,s}$ are constructed in a highly random fashion.

This allows us to use the following greedy algorithm to construct $I'_{s,t}$:

1. Let $H_1$ be the directed graph obtained by orienting all edges of $G_{s,t}$ in the direction of $\rho_{s,t}$. Repeat the following for $l = 1, 2, \ldots,$:
   2. If $H_l = \emptyset$ terminate.
   3. Find all source nodes of $H_l$. Add these to $I'_{s,t}$.
   4. Construct $H'_{l+1}$ by removing all source nodes and all successors of source nodes from $H'_l$.

The output of this algorithm is the LFMIS $I'_{s,t}$. Each step can be implemented in parallel time $O(1)$. The number of iterations of this algorithm is the length of the longest directed path in $G_{t,s}$. So it suffices it show that, whp, all directed paths in $G_{t,s}$ have small length.

Suppose we select $B_1, \ldots, B_l \in \mathcal{B}$ uniformly at random. Let us analyze how these could form a directed path in $G$.

Next, it must be that $B_2$ is a neighbor of $B_1$. Each variable $X_i \in B_1$ appears in at most $C_i - 1$ other bad-events. If $B_1$ and $B_2$ intersect in variable $i$, then that variable $i$ creates an edge between $B_1, B_2$ only if $x_{i,B_1} \neq a_i$, which occurs with probability $q_i$. Thus, for each $B_1$, the expected number of $B_2$ which are connected to that $B_1$ is at most

$$\sum_{X_i \in B_1} (C_i - 1) \leq \sum_{X_i \in B_1} 1/(Mq_i) \times q_i \leq 1$$

Continuing this way, that the expected number of $B_1, \ldots, B_l$ which are connected, is at most 1.

Finally, it must be the case that $\rho(B_1) < \rho(B_2) < \cdots < \rho(B_l)$. So far, none of the probabilistic statements have referred to $\rho$, so the probability this occurs conditional on all previous events is $1/l!$. Thus, for $l \geq \Omega(\frac{\log n}{\log \log n})$, this is $n^{-\Omega(1)}$ as desired. $\square$

PROPOSITION 4.22. *For $s = \Omega(M \log n)$, we have $V_{t,s} = \emptyset$ whp.*

PROOF. We will show that $V_{t,s}$ has an expected size which is decreasing exponentially in $s$.

First, we show the following fact: given any $B \in I_{t,s}$, we have $B \in I'_{t,s}$ with probability $\geq 1/2$. For, a sufficient condition for $B \in I'_{t,s}$ is that there is no variable $X_i \in B$ with $B' \in I_{t,s}, \rho(B') < \rho(B)$ and $x_{B',i} \neq a_i$. For each variable $i$, there are at most $C_i - 1$ candidate $B' \in I_{t,s}$, and each of them has probability $q_i$ of setting $x_{B',i} \neq a_i$, so the expected number of such $B'$ is at most $q_i \times 1/(Mq_i) \times 1/2 \leq \frac{1}{2M}$. Over all variables $X_i \in B$, this gives a total probability of $\leq 1/2$.

Now consider any $B \in V_{t,s}$. By maximality of $I_{t,s}$, either $B \in I_{t,s}$, or $B$ contains some variable which occurs $C_i$ times in $I_{t,s}$. In the former case, $B$ is necessarily removed from $V_{t,s+1}$.

Now, suppose variable $X_i \in B$ occurs exactly $C_i$ times in $I_{t,s}$. For each such occurrence $B'$, there is a probability of $\geq 1/2$ that $B' \in I'_{t,s}$. Note that the event that that $B' \in I'_{t,s}$ is independent of $x_{B',i}$, so each such $B'$ has a probability of $q_i/2$ that $B'$ is selected *and* $x_{B',i} \neq a_i$. Hence the total expected number of $B' \in I'_{t,s}$ with $x_{B',i} \neq a_i$, is at least $C_i \times q_i/2 \geq 1/(2M)$. Note that there are either zero or one elements $B' \in I'_{t,s}$ with this property, hence the probability that $X_i$ switches is at least $1/(2M)$. If this occurs, then we have $B \notin V_{t,s+1}$.

In either case, we have shown that a given $B \in V_{t,s}$ is removed with probability at least $1 - 1/(2M)$. Hence we have

$$\mathbf{E}[|V_{t,s+1}|] \leq (1 - 1/(2M))|V_{t,s}|$$

which implies that for $s \geq \Omega(M \log n)$ we have $V_{t,s} = \emptyset$ with high probability. $\square$

Finally, we show that the number of rounds is small.

PROPOSITION 4.23. *Suppose that $B$ is resampled in round $t$. Then the witness tree corresponding to this resampling has height $t$.*

PROOF. For each $t' \leq t$, let $\hat{\tau}_{(t')}$ denote the tree formed for the resampling of $B$ from round $t'$ onward (that is, we only add events in rounds $t', \ldots, t$ inclusive to the witness tree).

We will prove by induction the stronger claim: Suppose that $B$ is resampled in round $t$. Then for each $t' \leq t$, the tree $\hat{\tau}_{(t')}$ has height exactly $t - t' + 1$; furthermore, all the nodes at depth $t - t' + 1$ correspond to bad-events resampled at round $t'$. (Depth 1 corresponds to the root of the tree).

The base case of this induction is $t' = t$. In this case, note that all events resampled in round $t$ agree on all variables, and each bad-event $B \in \mathcal{B}$ is resampled at most once. Hence $\hat{\tau}_{(t)}$ consists consists of just a singleton node labeled by $B$.

We move on to the induction step. We begin with $\hat{\tau}_{(t')}$ and wish to extend it backward in time to round $t' - 1$. By induction hypothesis, $\hat{\tau}_{(t')}$ has height exactly $t - t' + 1$ and the nodes at depth $t - t' + 1$ correspond to bad-events resampled at round $t'$.

Note first that all bad-events encountered in round $t' - 1$ are true at the beginning of that round. So they agree on all variables, which implies that they cannot be children of each other. This implies that the only possible nodes at depth $t' - t + 2$ in $\hat{\tau}_{(t'-1)}$ correspond to bad-events resampled in round $t' - 1$ which have as their parent a node of depth $t' - t + 1$. Thus, the height of $\hat{\tau}_{(t'-1)}$ is either $t - t' + 2$ (as we want to show), or is $t - t' + 1$.

Next, we must show that $\hat{\tau}_{(t'-1)}$ has indeed height $t - t' + 2$. Suppose for contradiction that $\hat{\tau}_{(t'-1)}$ has height $t - t' + 1$.

By induction hypothesis, the tree $\tau_{(t')}$ contains some node $v$ labeled by $B'$ at height $t' - t + 1$ resampled in round $t'$.

First suppose $B'$ is true at the beginning of round $t' - 1$, so $B' \in \mathcal{V}_{t'-1,1}$. Then either $B'$ is resampled in round $t' - 1$, or $B'$ becomes false during round $t' - 1$. In the first case, $B'$ would be eligible to be placed as a child of $v$, so it is either placed there or at some other position at the same depth; either way, $\hat{\tau}_{(t'-1)}$ would have height $t - t' + 2$. In the second case, it must be that $B'$ contains a variable which switched in round $t' - 1$. This implies that $B'$ remains false at the end of round $t' - 1$, so $B' \notin \mathcal{V}_{t',1}$; but $B'$ was resampled in round $t'$ so this is a contradiction.

Second suppose $B'$ is false at the beginning of round $t' - 1$. It must have become true due to some variable $X_i$ switching in round $t' - 1$ due to resampling some $B''$. But then $B''$ disagrees with $B'$ on variable $X_i$, so $B'' \sim B'$. As $v$ is a leaf node in, this implies that $B''$ would be eligible to placed as a child of $v$. Again, such $B''$ will be placed either as a child of $v$ or at some position at the same depth, so that $\hat{\tau}_{(t'-1)}$ would have height $t - t' + 2$. □

PROPOSITION 4.24. *The parallel algorithm terminates after $O(\frac{\log W}{\epsilon})$ rounds whp.*

PROOF. In each round $t$, there is at least one resampling, which must correspond to some tree of height $t$. As shown in [**87**], due to the slack condition the total weight of all such trees rooted in a bad-event $B$ is $O(\mu(B)(1+\epsilon)^{-t})$. Summing over all such $B$, this implies that for $t = \Omega(\log(nW))$ this weight is $n^{-\Omega(1)}$. Hence whp there are no such trees. □

Putting this all together, we have the following:

THEOREM 4.25. *Suppose that each $B \in \mathcal{B}$ has size at most $M$. Suppose that we satisfy the condition*

$$\forall B \in \mathcal{B}, \mu(B) \geq P_\Omega(B)(1+\epsilon) \sum_{Y \text{ orderable to } B} \prod_{B' \in Y} \mu(B')$$

*Then whp the Parallel MT algorithm terminates in time* $\epsilon^{-1}M(\log W)(\log^{O(1)} n)(M+\log^{O(1)} m)$ *using* $(nm)^{O(1)}$ *processors.*

Note that the running time of the Simplified Parallel MT algorithm does not depend on $M$. In practice, when $M$ is large, then other aspects parallel aspects of the MT algorithm can become problematic; for example, we need non-trivial parallel algorithms to enumerate and check the events of $\mathcal{B}$. It remains an interesting open problem to find a parallel algorithm which works in the regime in which $M$ is large *and* the probabilities of the bad-events become close to 1.

## 4.3. Applications

**4.3.1. SAT with bounded variable occurrences.** Consider the following problem: we have a SAT instance, in which each clause contains at least $k$ variables. We are also guaranteed that each variable occurs in at most $L$ clauses, either positively or negatively. How can large can $L$ be so as to guarantee the existence of a solution to the SAT instance? This problem was first introduced by [**69**], which showed some bounds on $L$. Most recently, it was addressed by [**43**]; they showed that the criterion $L \leq \frac{2^{k+1}}{e(k+1)}$ suffices to guarantee that a solution exists (and can be found efficiently). This criterion is also shown to be asymptotically optimal (up to first-order terms). The main proof for this is to use the LLLL; they show that the worst-case behavior comes when each variable appears in a balanced way (half positive and half negative).

Although the criterion of [**43**] is asymptotically optimal, we can still improve its second-order terms. We show the following bound:

THEOREM 4.26. *If each variable appears at most*

$$L \leq \frac{2^{k+1}(1-1/k)^k}{k-1} - \frac{2}{k}$$

*times then the SAT instance is satisfiable, and the MT algorithm finds a satisfying occurrence in polynomial time.*

*Furthermore, suppose* $k = \log^{O(1)} n$ *and* $L \leq \frac{2^{k+1}(1-1/k)^k}{(k-1)(1+\epsilon)} - \frac{2}{k}$. *then whp the Parallel MT algorithm finds a satisfying occurrence in time* $\frac{k^{O(1)} \log^{O(1)} n}{\epsilon}$.

PROOF. We will only prove the sequential result; the parallel result is almost identical.

For each SAT clause, we have a bad-event $B$ that it is violated. We define $\mu(B) = \alpha$ for each bad-event, where $\alpha > 0$ is a constant to be chosen.

As described by [**43**], the key problem is to choose a good probability distribution for each variable. Suppose a variable $i$ occurs in $l_i$ clauses, of which it occurs $\delta_i l_i$ positively. In this case, we set variable $i$ to be T with probability $1/2 - x(\delta_i - 1/2)$, where $x \in [0, 1]$ is a parameter to be chosen. This is quite counter-intuitive. One would think that if a variable occurs positively in many clauses, then one should set the variable to be T with high probability; in fact we do the opposite.

We now wish to show that our MT criterion is satisfied. let $C$ be a clause, suppose wlg each variable appears in it negatively. Then the corresponding bad-event is that all such variables are true. This has probability $\prod_{i \in C}(1/2 - x(\delta_i - 1/2))$. Now, consider the assignable sets for the clause. We may either select the singleton $C$ itself, or for each of the $k$ variables we may select one or zero other clauses in which the corresponding variable appears *positively*. For each such variable $i$, the total number of such clauses is at most $\delta_i L$. Hence we have the criterion:

$$\alpha \geq \prod_{i \in C} \Big(1/2 - x(\delta_i - 1/2)\Big) \Big(\alpha + \prod_{i \in C} (1 + \delta_i L \alpha)\Big)$$

We bound the RHS as follows:

(6)

$$\prod_{i \in C} \Big(1/2 - x(\delta_i - 1/2)\Big) \Big(\alpha + \prod_{i \in C} (1 + \delta_i L \alpha)\Big) \leq \prod_{i \in C} \Big(1/2 - x(\delta_i - 1/2)\Big) \Big(1 + \delta_i L \alpha + \alpha/k\Big)$$

Now set $x = \frac{\alpha k L}{2\alpha + 2k + \alpha k L}$; clearly $x \in [0, 1]$. With this choice, verify that that the RHS of (6), viewed as a function of $\delta_i$, achieves its maximum value at $\delta_i = 1/2$. Thus

we have

$$\prod_{i\in C}\Big(1/2-x(\delta_i-1/2)\Big)\Big(\alpha+\prod_{i\in C}(1+\delta_i L\alpha)\Big) \le \prod_{i\in C}\frac{1}{2}(1+\alpha/k+\alpha L/2) = 2^{-k}(1+\alpha/k+\alpha L/2)^k$$

We thus need to find $\alpha \ge 0$ such that

$$\alpha - 2^{-k}(1+\alpha/k+\alpha L/2)^k \ge 0 \tag{7}$$

We differentiate with respect to $\alpha$ to make the LHS of (7) as large as possible. This yields our optimal choice of $\alpha$ namely:

$$\alpha = \frac{2k\left((\frac{2^{k+1}}{2+kL})^{\frac{1}{k-1}}-1\right)}{2+kL}$$

When $L \le \frac{2^{k+1}(1-1/k)^k}{k-1} - \frac{2}{k}$, note that $(\frac{2^{k+1}}{2+kL})^{\frac{1}{k-1}} \ge \frac{k}{k-1}$ and so $\alpha \ge 0$ as desired. Also, (7) is satisfied. □

**4.3.2. Hypergraph coloring.** Suppose we have a $k$-uniform hypergraph, in which each vertex appears in at most $L$ edges. We wish to $c$-color this hypergraph, while avoiding any monochromatic edges. There are many types of graphs and parameters for which better bounds are known, but the LLL gives very simple constructions and also provides the strongest bounds in some cases (particularly when $c, k$ are fixed small integers)[**83**].

Let us first examine how the conventional LLL analysis would work. Counterintuitively, when $c$ is large it is better to use the standard LLL (defining $\sim$ in terms of simple dependency) and when $k$ is large it is better to use the LLLL (defining $\sim$ in terms of lopsidependency.) In the first case, a bad-event is that an edge is monochromatic (of an unspecified color). Consider an edge $f$. The neighbors of $f$ would be other edges than intersect $f$. An independent set of neighbors of $f$ consists of either $f$ itself, or for each vertex $v \in f$ we may select one or zero edges (other than

$f$). Setting $\mu(B) = \alpha$ for all bad-events, this gives us the criterion

$$\alpha \geq c^{1-k}(\alpha + (1 + (L-1)\alpha)^k)$$

Routine calculations show that this can be satisfied if $L \leq \frac{c^{k-1}(1-1/k)^{k-1}}{k}$.

Alternatively, in the LLLL, a bad-event would be that an edge $f$ receives some color $j$. The neighbors of this event would be other edges receiving colors other than $j$; there are $k(L-1)(c-1)+c$ such neighbors. Using the symmetric LLL and some simplifications, one obtains the bound $L \leq \frac{c^k}{(c-1)ek}$.

Applying Pegden's criterion to this calculation could give some improvements, but the relevant calculations no longer have a simple closed form.

There seems to be a "basic" form of the bound $L \leq \frac{c^{k-1}}{ek}$; the standard LLL framework can improve on this using either Pegden's criterion (replacing $1/e$ by $(1 - 1/k)^{k-1}$) or by lopsidependency (replacing one factor of $c$ by $(c-1)$), but cannot do both simultaneously.

**Our new LLLL criterion** We now apply our new LLLL criterion to this problem. For each edge $f \in G$, we have $c$ bad-events, namely that $f$ is monochromatic of any given color. We assign $\mu(B) = \alpha$ to all bad-events, where $\alpha > 0$ is a constant to be determined. We color each vertex independently and uniformly.

Now consider a bad-event $B$, say wlg that edge $f$ receives color 1. It has probability $c^{-k}$. Now consider the orderable sets for $B$; we want to sum $\prod_{B' \in Y}$ over all such sets $Y$.

First, $Y$ may consist of $B$ itself; this contributes $\alpha$. Second, $Y$ may consist of, for each vertex $v \in f$, zero edges or one edge other than $f$ receiving one color $2, \ldots, c$. Finally, we may have one vertex select $f$ and some color for it; some set of other vertices selects other edges and other colors. Summing all these cases, we have the criterion for $B$:

(8)

$$\alpha \geq c^{-k}\Big[(1+\alpha(c-1)(L-1))^k + \alpha(c-1)((1+\alpha(c-1)(L-1))^k - (\alpha(c-1)(L-1))^k) + \alpha\Big]$$

This has no closed-form solution for general $L, k$. But, for fixed values of $L, k$ it is easily solvable. For example, when $c = 2$, we list the largest values of $L$ which are obtained by either our improved MT criterion or the original MT criterion (listed under $L'$):

| $k$ | $L$ | $L'$ | $k$ | $L$ | $L'$ |
|---|---|---|---|---|---|
| 4 | 2 | 2 | 8 | 13 | 12 |
| 5 | 3 | 3 | 9 | 23 | 21 |
| 6 | 5 | 4 | 10 | 40 | 38 |
| 7 | 8 | 7 | 11 | 72 | 69 |

We see that our new criterion indeed gives (slightly) stronger bounds. For the asymptotic case when $L$ is large, note that the RHS of (8) can be approximated:

$$\begin{aligned} \text{RHS} &\leq c^{-k}(\alpha + (1 + \alpha(c-1)(L-1))^k(1 + (c-1)\alpha)) \\ &\leq c^{-k}(1 + \alpha(c-1)L)^k \end{aligned}$$

Thus, setting $\alpha = \frac{\left(\frac{c^k}{(c-1)kL}\right)^{\frac{1}{k-1}} - 1}{(c-1)L}$, we satisfy the LLLL criterion if

$$L \leq \frac{c^k(1-1/k)^{k-1}}{(c-1)k}.$$

which is slightly better than the bounds from the conventional LLLL.

**4.3.3. Second Hamiltonian cycle.** Consider a $k$-regular graph $G$, with a Hamiltonian cycle $C$. Under what conditions is there a second Hamiltonian cycle $C'$ (that is, the cycle $C$ is not unique)? In [**112**], Thomassen showed that a necessary condition for the existence of the second cycle followed from the existence of a set of vertices $S \subseteq V$ satisfying the following properties:

(1) If $v$ and $w$ are adjacent on the cycle $C$, then $v$ and $w$ are not both in $S$.

(2) For any vertex $v \in G$, either $v$ is in $S$, or it is connected to a vertex $w \in S$ via some edge $e \notin C$.

In other words, $S$ is a dominating set for $G - C$, and an independent set for $C$.

Using the LLL, Thomassen then showed that this condition can always be satisfied as long as $k \geq 73$. This was based on a simple random process, in which each vertex was put into $S$ independently with probability $p$. Using the Lovász Local Lemma with a much more sophisticated random process, Haxell showed that this condition can be satisfied as long as $k \geq 23$ [**58**]. It was conjectured that this condition could be satisfied as long as $k \geq 5$.

Haxell's proof is quite involved, and our LLLL criterion would offer little benefit for it (as all the bad-events involve many vertices). In [**45**], there was a simple proof using the LLLL that this condition can be satisfied as long as $k \geq 48$. Our LLLL criterion can be used to give another very simple proof under the condition $k \geq 43$. While not as good as Haxell's construction, the proof is far simpler.

THEOREM 4.27. *If $G$ is a $k$-regular graph for $k \geq 43$ and $C$ is a Hamiltonian cycle of $G$, then there is a $G - C$-dominating, $C$-independent set $S \subseteq V$.*

PROOF. Each vertex enters into $S$ independently with probability $p$. There are two types of bad-events: for each edge of $C$, there is an event of type A, that the endpoints are both in $S$; for each vertex of $G - C$, there is an event of type B, that $v$ nor its $k - 2$ neighbors outside of $C$ are in $S$. We assign $\mu(B) = a$ for all events of the first type, and $\mu(B) = b$ for all events of the second type. Note that events of type A are lopsidependent only with events of type B, and vice versa.

Now consider an event of type A. It has probability $p^2$. There are two vertices in this edge, each of which participates in $k - 2$ events of type B. Similarly, an event of type B has probability $(1 - p)^{k-1}$, and each of the $k - 1$ vertices participates in two events of type A. Hence our LLLL criteria can be stated as

$$a \geq p^2(1 + (k - 2)b)^2, \quad b \geq (1 - p)^{k-1}(1 + 2a)^{k-1}$$

Routine calculations show that this is solvable for $k \geq 43$. □

**4.3.4. Independent transversals.** We have seen in Chapter 1 how the cluster-expansion criterion can show the existence of an independent transversal when $b \geq 4\Delta$. This can be easily translated, via Pegden's criterion, to give an algorithm which finds an independent transversal when $b \geq 4\Delta$. Although it is known that independent transversals exist under more stringent conditions, this remains, to the best of our knowledge, the strongest criterion known for any polynomial-time algorithm to find an independent transversal. Using our new LLLL criterion we slightly can improve this, obtaining the best constructive bound known so far:

PROPOSITION 4.28. *Suppose $b \geq 4\Delta - 1$. Then the MT algorithm finds an independent transversal in polynomial expected time. Furthermore, under these conditions, the Parallel MT algorithm runs in time*

$$\log^{O(1)} n \times \min(1, \frac{4(b-1)\Delta}{b^2 - 4(b-1)\Delta})$$

PROOF. We prove the first statement only; the second is similar.

Each edge corresponds to a bad-event; it has probability $1/b^2$. For an assignable set of neighbors to an edge $f = \langle u, v \rangle$, we may choose $f$, or we may choose $\langle u', x \rangle$ where $u' \neq u$ is in the class of $u$, or we may choose $\langle v', x \rangle$ where $v' \neq v$ is in the class of $v$; or we may choose both of the latter choices. This gives us the criterion

$$\alpha \geq b^{-2}(\alpha + (1 + (b-1)\Delta\alpha)^2)$$

which is satisfied by some $\alpha \geq 0$ whenever $b \geq 4\Delta - 1$. □

Note that the second condition gives an RNC algorithm either if $b \geq 4\Delta(1+\epsilon)$ for some constant $\epsilon > 0$, or if $b \geq 4\Delta - 1$ and $\Delta = \log^{O(1)} n$.

**4.3.5. Off-diagonal Ramsey numbers.** In this section, we consider the classical off-diagonal Ramsey problem on graphs. Suppose we wish to two-color – with colors red and blue – the edges of $K_n$, the complete graph on $n$ vertices. We wish to avoid any red $s$-cliques or blue $t$-cliques in the resulting graph. The largest value $n$ for

which it is possible to avoid such cliques is known as the *off-diagonal Ramsey number* $R(s,t)$. There are many aspects and generalizations studied for Ramsey numbers. One frequently studied scenario is when $s$ is held constant while $t \to \infty$.

It was shown in [**104**], using the LLL, that when $n \leq c(t/\log t)^{\frac{s+1}{2}}$ and $c$ is a constant (depending on $s$) that such a coloring is possible. In other words, $R(s,t) \geq \Omega_s\big((t/\log t)^{\frac{s+1}{2}}\big)$. For specific values of $s$, better bounds are known (e.g., $R(3,t) = \Theta(t^2/\log t)$), but this is the best bound known for general $s$. The algorithmic challenge is to efficiently find colorings of the edges of $K_n$ that avoid red $K_s$ and blue $K_t$. Such algorithms should operate when $n$ is as large as possible, ideally up to $R(s,t)$. Unfortunately, the LLL construction of [**104**] does not lead to efficient serial or parallel algorithms. The main roadblock is that there is a bad-event for each $t$-clique, so that finding a bad event requires exponential time. For specific values of $s$, again, there are known polynomial-time algorithms for finding good colorings. But in general there is no algorithm that corresponds to the best bounds.

In [**48**], an algorithm based on MT was proposed for finding such colorings. The basic idea of [**48**] is to find and resample red $K_s$, while ignoring blue $K_t$. One then shows that with high probability, none of the $K_t$ became blue, even though we did not explicitly check or resample them. The serial running time of this would be $\Omega_s(n^s)$, to search for the red $K_s$; no parallel algorithm was given.

These results depend on the MT-distribution. Though we did not show this explicitly in Section 4.1, an easy adaptation of Lemma 4.10 gives us the following:

THEOREM 4.29. *Suppose we have a set of bad-events $\mathcal{B}$ which satisfies our LLLL criterion which weights $\mu$. Suppose $E$ is any atomic event (which is not itself in $\mathcal{B}$). Then the probability that $E$ is true at the end of the MT algorithm is given by*

$$P(E \text{ is true at end of MT}) \leq P_\Omega(E) \sum_{Y \text{ orderable to } E} \prod_{B' \in Y} \mu(B')$$

PROOF. This is nearly identical to the proof of Theorem 2.16. □

Using this result,we will give a serial algorithm for off-diagonal Ramsey coloring with a much better run-time, and we will also give a parallel algorithm based on our Parallel MT algorithm.

THEOREM 4.30. *We want to find a red-blue coloring of the edges of $K_n$ avoiding red $K_s$ and blue $K_t$. Define*

$$c_s = \left(\frac{2}{s} - \frac{2}{s-1} + 1\right)^{\frac{s+1}{2}} \left(\frac{2(s-2)!}{s(s-1)^{\binom{s}{2}}}\right)^{\frac{1}{s-2}}.$$

(1) *Suppose $n \leq (\frac{t}{\log t})^{\frac{s+1}{2}}(c_s - o(1))$. There is a serial randomized algorithm which runs in time $n^{s/4+O(1)}$ and produces a correct solution when it halts, except with a failure probability of $n^{-\Omega(1)}$.*

(2) *Suppose $s$ is constant and $n \leq (\frac{t}{\log t})^{\frac{s+1}{2}}(c_s - o(1))$. There is a parallel randomized algorithm which runs in time $s^{O(1)} \log^{O(1)} n$ time using $n^{s/4+O(1)}$ processors, and produces a correct solution when it halts, except with a failure probability of $n^{-\Omega(1)}$.*

PROOF. The proofs of both parts are very similar; to simplify the discussion, we will focus mostly on the serial algorithm, noting any difference between that and the parallel algorithm.

The probability space $\Omega$ is defined by coloring each edge red with probability

$$p = \left(\frac{2(s-2)!}{(s-1)s}\right)^{\frac{2}{s^2-s-2}} n^{\frac{-2}{s+1}}$$

and blue otherwise. We ignore the blue $K_t$ and so our only bad-events are the red $K_s$. Each bad-event has probability $q = p^{\binom{s}{2}}$. Observe that each bad-event is *lopsidependent* with only a single bad-event, namely itself. So the LLLL criterion is satisfied, setting $\mu(B) = \frac{q}{1-q}$ for all bad-events $B$.

Now consider an arbitrary $K_t$, and let $E$ be the event that it is red at the end of the MT. We have $P_\Omega(E) = (1-p)^{\binom{t}{2}}$. The orderable sets for this event can be found as follows: for each of the $\binom{t}{2}$ edges, we may select zero or one blue $K_s$. Thus, by

Theorem 4.29, the probability that $E$ holds at the end of MT is given by

$$\begin{aligned}
P(K_t \text{ is blue}) &\leq \left((1-p)(1+\binom{n-2}{s-2}\mu)\right)^{\binom{t}{2}} \\
&\leq \left(1-\left(\frac{s(s-1)}{2(s-2)!}\right)^{\frac{2}{-s^2+s+2}}\left(1+\frac{2}{s-s^2}\right)n^{\frac{-2}{s+1}}\right)^{\binom{t}{2}} \\
&\leq \exp\left(-\binom{t}{2}c_s' n^{\frac{-2}{s+1}}\right) \\
&\text{where } c_s' = \left(\frac{2(s-2)!}{s(s-1)}\right)^{\frac{2}{s^2-s-2}}\left(1+\frac{2}{s-s^2}\right)
\end{aligned}$$

Hence, the expected number of blue $K_t$ is at most

$$\mathbf{E}[\text{Blue } K_t] \leq \frac{n^t}{t!}\exp(-t^2 c_s' n^{\frac{-2}{s+1}}/2)$$

In order to avoid all blue $K_t$ with high probability, say with probability $n^{-\phi}$ for $\phi > 0$ some arbitrary constant, we must take

$$\begin{aligned}
n &\leq \left(\frac{c_s' t^2}{(s+1)(\phi+t)\log\left(\frac{c_s' t^2 (t!)^{-\frac{2}{(s+1)(\phi+t)}}}{(s+1)(\phi+t)}\right)}\right)^{\frac{s+1}{2}} \\
&= \left(t/\log t\right)^{\frac{s+1}{2}}(c_s - o_s(1))
\end{aligned}$$

So far, we have shown that when we run the MT algorithm with the given parameters, then indeed we avoid $K_t$ with high probability. The next thing we must examine is how to run the MT algorithm. In this problem, in which the bad-events are defined to be red $K_s$, the MT algorithm is somewhat degenerate. The critical thing to note is that when we resample a bad-event, we can never create new bad-events. Thus, the most potentially time-consuming step of MT — repeatedly searching for any bad-events which are currently true — can be much simplified. At the beginning of the process, after we make the initial random color assignment but before we do any resamplings, we can enumerate all red $K_s$. For each such red $K_s$, we repeatedly sample the edges until the $K_s$ is no longer red. The process of finding the red $K_s$ can

be aided by the fact that we are searching for them in a *random* graph — namely, the edges are red independently with probability $p$.

The simplest way to search for such red $K_s$ seems to be through a branching process: we gradually build up red $K_l$, for $l \leq s$. In this branching process, the expected number of red $K_l$ is $p^{\binom{l}{2}}\binom{n}{l}$. Thus, the total time complexity of this branching process will be

$$\begin{aligned}\text{Time to find red } K_s &\leq n^{O(1)} \sum_{l=0}^{s} \binom{n}{l} p^{\binom{l}{2}} \\ &\leq n^{O(1)} \exp(\max_{l\in[0,s]} l\log n + \frac{(l-1)^2}{2}\log p - l\log l + l) \\ &\leq n^{s/4+O(1)} \qquad \text{(by routine calculus)}\end{aligned}$$

This concludes part (1) of the theorem. As the parallel MT algorithm simulates the serial algorithm, all the results about the MT distribution still remain true in the parallel setting; in particular we avoid the blue $K_t$ with high probability. Also, one can enumerate the red $K_s$ in parallel, using $n^{s/4+O(1)}$ processors and $s$ stages, via the same type of branching process.

One can see easily that we satisfy the parallel LLLL criterion for $n$ sufficiently large, setting $\mu(B) = 1$ for all $B$ and $\epsilon = 1/2$. All the bad events then use $\binom{s}{2}$ elements, and the total number of bad-events is $n^{O_s(1)}$, and we have $\log \sum_{B\in\mathcal{B}} \mu(B) = O_s(\log n)$. Hence by Theorem 4.25, the Parallel MT terminates in time $s^{O(1)} \log^{O(1)} n$ whp. □

### 4.4. MT can be more powerful than the Shearer criterion

In [**43**], the asymmetric LLL was applied to $k$-SAT satisfiability. We show that this type of analysis was close to best possible application of the LLL, in the sense that stronger forms of the LLL would not have been able to improve the bounds significantly. Suppose we are given a formula $\Phi$ in each variable appears at most $L$ times. Suppose each variable appears at most $L/2$ times positively and at most $L/2$ times negatively in $\Phi$. Suppose we define the natural probability space in which

each variable is set to true with probability 1/2 and false with probability 1/2, all independently of each other, and we define a separate bad-event for each clause in $\Phi$ becoming false. In this case, each bad-event has probability $p_i = p = 2^{-k}$.

Then, as shown in [**43**], for $L \leq L_0 = \frac{2^{k+1}}{e(k+1)}$, the LLLL can applied to this probability space to show that, with positive probability, none of the bad-events occur: i.e. none of the clauses are falsified, and so $\Phi$ is satisfiable. As we have shown in Section 4.3.1, for $L \leq L_1 = \frac{2^{k+1}(1-1/k)^k}{k-1} - \frac{2}{k}$, the Moser-Tardos "simulation" of this probability space terminates with a satisfying assignment.

In order to show that the MT algorithm, and our criterion for it, can be stronger than the Shearer criterion, we will create a formula $\Phi$, in which each variable appears at most $L/2$ times positively and $L/2$ times negatively, for some $L \in (L_0, L_1)$. However, for the natural probability space $\Omega$, the Shearer criterion is violated. That means that the LLL, applied to the natural probability space, cannot be used to show that this formula is satisfiable, even though the MT algorithm for it converges. In other words, information about the probability and dependency structure of the clauses is not sufficient to show that $\Phi$ is satisfiable: we must also take account of its decomposition into variables.

**4.4.1. Constructing the extremal formula $\Phi$.** We will construct a family of formulas $\Phi_T$, for $T \in \mathbf{N}$, as follows. Suppose $L$ is a fixed even integer. Initially, $\Phi_0$ contains no clauses. At stage $i$ of the process, we create $L-2$ clauses containing variable $i$; exactly $L/2-1$ clauses in which $i$ appears positively and exactly $L/2-1$ clauses in which $i$ appears negatively. All the other variables in these clauses are completely new, not appearing in any clause of $\Phi$; they all appear positively in the $L-2$ new clauses. When we form $\Phi_i$, each of the new variables (other than variable $i$) appears in exactly one new clause. We refer to the process of adding $L-2$ clauses containing variable $i$ as *expanding* variable $i$.

PROPOSITION 4.31. *For any $T \in \mathbf{N}$ and every $i \in [n]$, variable $i$ appears at most $L/2$ times positively and at most $L/2-1$ times negatively in $\Phi_T$*

PROOF. One positive occurrence of variable $i$ may occur in expanding $i'$ for $i' < 1$. Otherwise, the only occurrences of variable $i$ appear when expanding variable $i$; this adds $L/2 - 1$ positive occurrences of $i$ and $L/2 - 1$ negative occurrences of $i$. $\square$

There is a natural tree structure $\mathcal{T}$ on the variables of $\Phi_T$: we say that variable $i$ is a parent of variable $j$ if variable $j$ was first introduced into $\Phi_T$ during the expansion of variable $i$.

We define the lopsidependency graph corresponding to $\Phi_T$ in the usual way: for each clause of $\Phi_T$, we have a separate vertex and we have an edge between two vertices if the corresponding clauses *disagree* on a variable. We let $G_T$ denote the lopsidependency graph corresponding to the formula $\Phi_T$.

Although the graph $G_T$ is complicated, we will show that it contains a relatively simple and regular type of subgraph. We will actually show that Shearer's criterion is violated for this subgraph; as shown in [**102**], this suffices to show that Shearer's criterion is violated for the overall graph $G_T$.

The graph family $H_j$ will consist of many copies of $K_{L/2-1,L/2-1}$, the complete bipartite graph with $L/2 - 1$ vertices on each side. Each graph $H_j$ has a special copy of this $K_{L/2-1,L/2-1}$, which is labeled as the *root* of $H_j$. We define the graph family $H_j$ recursively. First, $H_0$ is the null graph (the graph on 0 vertices). To form $H_{j+1}$, we do the following: we take a new copy of $K_{L/2-1,L/2-1}$, which we will designate as the root of $H_{j+1}$. Then, for each vertex $v$ in this root, we add $k - 1$ separate new copies of $H_j$, along with an edge connecting $v$ to all the vertices in the right-half of the root of the corresponding $H_j$.

PROPOSITION 4.32. *Let $j > 0$ be any fixed integer. Then, there is some $T$ sufficiently large (which may depend on $j$) such that $G_T$ contains a copy of $H_j$.*

PROOF. Recall that $\mathcal{T}$ defines a tree structure on the variables of $\Phi_T$. For any variable $i$, let $\mathcal{T}[i]$ denote the subtree of $\mathcal{T}$ rooted at $i$.

We will prove by induction on $j$ a stronger claim: for any variable $i$, there is some $T = T(i, j)$ sufficiently large such that the subgraph of $G_T$ induced on the vertex set $\mathcal{T}[i]$ contains a copy of $H_j$, and the root of this copy of $H_j$ corresponds to the new clauses introduced during the expansion of $i$.

When $j = 0$ this is vacuously true. To show the inductive step, note that for $T$ sufficiently large, then variable $i$ is expanded in $\Phi_T$. Thus, in $\Phi_T$, variable $i$ has $(L-1)(k-1)$ children in $\mathcal{T}$ (the variables introduced during the expansion of $i$); we denote these by $C$ and we denote the clauses introduced during the expansion of $i$ by $A$. By inductive hypothesis, for $T$ sufficiently large, the subgraphs of $G_T$ induced on these children all contain copies of $H_j$. Because $\mathcal{T}$ is a tree structure, these subgraphs do not intersect in any vertices.

Now consider the clauses introduced during the expansion of $i$. All the clauses in which $i$ appears positively are lopsidependent with all of the clauses in which $i$ appears negatively. In addition, consider each variable $x \in C$. By inductive hypothesis, the root of the corresponding $H_j$ corresponds to the clauses introduced during the expansion of $x$. In the clauses corresponding to the expansion of variable $i$, this variable $x$ appears in exactly one clause; in this root node, it appears negatively in $L/2 - 1$ clauses. Hence, $G_T$ also contains $k - 1$ edges from each vertex in $A$ to all the vertices in the right-half of the new copy of $H_j$. □

**4.4.2. Computing the Shearer criterion for $H_j$.** We now discuss how to compute the Shearer criterion for the family of graphs $H_j$. We will show that, for $j$ sufficiently large, $S(H_j, \emptyset) < 0$.

We will make use of two computational tricks for independent set polynomials.

PROPOSITION 4.33. *Suppose $G$ has connected-components $G_1, G_2$. Then we have*

$$Q(G, \emptyset, p) = Q(G_1, \emptyset, p)Q(G_2, \emptyset, p)$$

PROPOSITION 4.34. *Suppose $X \subseteq V$ is any set of vertices. Then we have*

$$Q(G, \emptyset, p) = \sum_{\substack{U \subseteq X \\ U \text{ independent}}} Q(G - U - N(U), \emptyset, p) \prod_{i \in U} (-p_i)$$

*where $G - U - N(U)$ is the residual graph; we remove $U$ and all their neighbors from $G$.*

For the probability space corresponding to the formula $\Phi$, all the probabilities $p_i$ are equal to the common value $p = 2^{-k}$.

We will need to work also with slight variant of the graphs $H_j$. We define the graph family $H'_j$ as follows: the root node consists of a singleton vertex. To this we attach $k-1$ new copies of $H_{j-1}$, along with an edge from the singleton root to all the vertices in the right-half of the roots of $H_{j-1}$.

PROPOSITION 4.35. *Suppose we define*

$$s_j = Q(H_j, \emptyset, \vec{p}) \qquad r_j = Q(H'_j, \emptyset, \vec{p})$$

*where $\vec{p}$ is the probability distribution assigning probability $2^{-k}$ to each event. Then $s, r$ satisfy the mutual recurrence relations*

$$\begin{aligned} r_j &= s_{j-1}^{k-1} - p r_{j-1}^{(k-1)(L/2-1)} s_{j-2}^{(k-1)^2(L/2-1)} \\ s_j &= 2 r_j^{(L/2-1)} s_{j-1}^{(k-1)(L/2-1)} - s_{j-1}^{(k-1)(L-2)} \\ s_0 &= 1, r_0 = 1 \end{aligned}$$

PROOF. We will first deal with $H'_j$ as it is much simpler. We apply Proposition 4.34, taking $X$ as the singleton root node. In this case, $U$ is either the empty set, or $U$ is the root node. In the former case, the residual graph consists of $k-1$ independent copies of $H_{j-1}$.

In the latter case, suppose we remove the root node $v$ of $H'_j$ and its neighbors; let us consider one of the copies of $H_{j-1}$ to which $v$ was connected. In this copy of $H_{j-1}$,

all the vertices in the left half of the root are now disconnected and isolated, thus we have $L/2-1$ copies of $H'_{j-1}$. In addition, all the vertices in the right-half of the root are removed; each of those was connected to $k-1$ copies of $H_{j-2}$, which now become isolated copies. In total, when we remove $v$, then we are left with $(k-1)(L/2-1)$ isolated copies of $H'_{j-1}$ and $(k-1)^2(L/2-1)$ isolated copies of $H_{j-2}$.

Summing the contributions of these two terms according to Proposition 4.34 gives

$$r_j = Q(H'_j, \emptyset, \vec{p}) = s_{j-1}^{k-1} - pr_{j-1}^{(k-1)(L/2-1)} s_{j-2}^{(k-1)^2(L/2-1)}$$

We next deal with $H_j$. In any independent set $U$ of $H_j$, either $U$ contains zero vertices from the left half of the root of $H_j$, or zero vertices from the right-half of the root of $H_j$, or both. In the first two cases cases, when we remove the vertices in the left (respectively right) half of $H_j$, then we are left with $L/2-1$ copies of $H'_j$ and $(k-1)(L/2-1)$ copies of $H_{j-1}$. In the third case, we are left with $(k-1)(L-2)$ copies of $H_{j-1}$. We can sum the first two contributions and subtract the third, as it is double-counted: this gives

$$s_j = 2r_j^{(L/2-1)} s_{j-1}^{(k-1)(L/2-1)} - s_{j-1}^{(k-1)(L-2)}$$

□

PROPOSITION 4.36. *Define the function* $f : [0,1] \to \mathbf{R}$ *by*

$$f(a) = 1 - \frac{p}{(2 - a^{-(L/2-1)})^{k-1}}$$

*Suppose that the Shearer condition is satisfied for all* $G_T, T \geq 0$. *Then there is some* $a \in (2^{\frac{-2}{L-2}}, 1]$ *satisfying* $f(a) = a$.

PROOF. Define

$$a_j = \frac{r_j}{s_{j-1}^{k-1}}, b_j = \frac{s_j}{s_{j-1}^{(k-1)(L-2)}}$$

Then we have:

$$b_j = \frac{2r_j^{(L/2-1)} s_{j-1}^{(k-1)(L/2-1)} - s_{j-1}^{(k-1)(L-2)}}{s_{j-1}^{(k-1)(L-2)}} = \frac{2r_j^{(L/2-1)}}{s_{j-1}^{(k-1)(L/2-1)}} - 1 = 2a_j^{(L/2-1)} - 1$$

and we can obtain a pure first-order recurrence on the sequence $a_j$:

$$\begin{aligned} a_j &= \frac{s_{j-1}^{k-1} - pr_{j-1}^{(k-1)(L/2-1)} s_{j-2}^{(k-1)^2(L/2-1)}}{s_{j-1}^{k-1}} \\ &= 1 - p\frac{r_{j-1}^{(k-1)(L/2-1)}}{s_{j-2}^{(k-1)^2(L/2-1)}} \times \frac{s_{j-2}^{(k-1)^2(L-2)}}{s_{j-1}^{k-1}} \\ &= 1 - \frac{pa_{j-1}^{(k-1)(L/2-1)}}{b_{j-1}^{k-1}} \\ &= 1 - \frac{pa_{j-1}^{(k-1)(L/2-1)}}{(2a_{j-1}^{(L/2-1)} - 1)^{k-1}} = f(a_{j-1}) \end{aligned}$$

One may verify also that $a_0 = 1$.

Now suppose that for some $j \geq 1$ we have $a_j \leq 2^{\frac{-2}{L-2}}$. In this case, we have $b_j \leq 0$ and hence $\frac{s_j}{s_{j-1}^{(k-1)(L-2)}} \leq 0$. This implies that either $s_j \leq 0$ or $s_{j-1} \leq 0$, so the Shearer condition is violated for $H_j$ or $H_{j-1}$. This implies that the Shearer condition is violated for $G_T$ for $T$ sufficiently large.

Next, suppose that it holds that $f(a) < a$ for all $a \in (2^{-\frac{-2}{L-2}}, 1]$. This implies that the sequence $a_j$ is decreasing for $j \in \mathbf{N}$. As the sequence $a_j$ also satisfies $a_j \geq 2^{\frac{-2}{L-2}}$, it must converge to some limit point $a$. But, by continuity, this limit point must be a fixed point of the functional iteration, i.e. $f(a) = a$, which is a contradiction

So we know that $f(a) \geq a$ for some $a \in (2^{-\frac{-2}{L-2}}, 1]$. But also note that $f(1) = 1 - p < 1$. Hence, the function $f(a) - a$ changes sign on the interval $(2^{-\frac{-2}{L-2}}, 1]$. This implies there must be a fixed point $f(a) = a$ on this interval. □

PROPOSITION 4.37. *Suppose $L \geq \frac{2^{k+1}}{ek}(1 + \phi k^{-2})$, for some small constant $\phi > 0$. Then the Shearer condition is violated on some $G_T$, for $T$ sufficiently large.*

PROOF. Suppose the Shearer condition is satisfied for all $G_T$. By Proposition 4.36, the function $f$ has a fixed point $a \in (2^{-\frac{-2}{L-2}}, 1]$. Thus $L$ must satisfy the condition

$$L = 2 - \frac{2\ln\left(2 - 2^{\frac{k}{1-k}}(1-a)^{\frac{1}{1-k}}\right)}{\ln a}$$

Reparametrizing with $t = 2^{k/(1-k)}(1-a)^{1/(1-k)}$, we obtain the simpler expression $L = 2 - \frac{2\ln(2-t)}{\ln\left(1-2^{-k}t^{1-k}\right)}$, for some $t \in [0,2]$. Thus we have:

$$L \leq 2 + t^{k-1}2^{k+1}\ln(2-t) \tag{9}$$

Let $t_0 = 1 - (k+1/2)^{-1}$. A simple computation shows that the derivative of the RHS of (9) with respect to $t$ is negative for $t \geq t_0$. So the maximum value occurs at $t \leq t_0$, and for such $t$ we have

$$\ln(2-t) \leq \ln(2-t_0) - (t-t_0)\frac{1}{\frac{1}{k+\frac{1}{2}}+1}$$

Thus we have

$$\begin{aligned}
L &\leq 2 - \frac{2^{k+1}t^{k-1}\left(2k(t-1) - (2k+3)\log\left(\frac{1}{k+\frac{1}{2}}+1\right) + t + 1\right)}{2k+3} \\
&\leq 2 + \frac{2^{k+1}\left(\frac{k-1}{k}\right)^k (2k+1)^{1-k}\left(2k + (2k+3)\log\left(\frac{1}{k+\frac{1}{2}}+1\right) - 1\right)^k}{2k^2+k-3} + 2 \quad \text{by simple calculus} \\
&\leq \frac{2^{k+1}}{ek}(1 + O(k^{-2}))
\end{aligned}$$

□

We contrast this with our result in Section 4.3.1; there, we showed that the MT algorithm converges when $L \leq L_1$ and $L_1$ is on the order $\frac{2^{k+1}}{ek}(1+\Theta(k^{-1}))$. This means that there is a gap of size $\Theta(2^k/k^2)$ between the bounds provable from Shearer's criterion and the bounds provable from our new LLLL criterion.

CHAPTER 5

# The Moser-Tardos framework with partial resampling

In the standard MT algorithm, we resample all the variables which affect a true bad-event. In this chapter, we develop a new algorithm based on *partial resampling*, in which only a subset of the variables are resampled. We refer to this as the Partial Resampling Algorithm (PRA); the idea is to carefully choose a distribution $D_i$ over *subsets* of $\{j \in S_i : X_j\}$ for each $i$, and then, every time we need to resample, to first draw a subset from $D_i$, and then only resample the $X_j$'s that are contained in this subset. This partial-resampling approach leads to algorithmic results for many applications that are not captured by the LLL.

In order to motivate our applications, we start with two classical problems: scheduling on unrelated parallel machines [**74**], and low-congestion routing [**96**]. In the former, we have $n$ jobs and $K$ machines (we interchange the standard use of the indices $i$ and $j$ here, and use $K$ in place of the usual "$m$", in order to conform to the rest of our notation), and each job $i$ needs to be scheduled on any element of a given subset $M_i$ of the machines. If job $i$ is scheduled on machine $j$, then $j$ incurs a given load of $p_{i,j}$. The goal is to minimize the *makespan*, the maximum total load on any machine. The standard way to approach this is to introduce an auxiliary parameter $T$, and ask if we can schedule with makespan $T$ [**74, 106**]. Letting $[k]$ denote the set $\{1, 2, \ldots, k\}$, a moment's reflection leads to the following integer-programming

formulation:

$$\forall i \in [n],\ \sum_{j \in M_i} x_{i,j} = 1; \tag{10}$$

$$\forall j \in [K],\ \sum_{i} p_{i,j} x_{i,j} \leq T; \tag{11}$$

$$\forall (i,j),\ p_{i,j} > T \implies x_{i,j} = 0; \tag{12}$$

$$\forall (i,j),\ x_{i,j} \in \{0,1\}. \tag{13}$$

(Although (12) is redundant for the IP, it will be critical for the natural LP relaxation [**74**].) In low-congestion routing, we are given a collection of (source, destination) pairs $\{(s_i, t_i) : i \in [n]\}$ in a $V$-vertex, $K$-edge directed or undirected graph $G$ with edge-set $E$; each edge $f \in E$ has a capacity $c_f$, and we are also given a collection $P_i = \{P_{i,j}\}$ of possible routing paths for each $(s_i, t_i)$-pair, with each such path indexed by $i$ and an auxiliary index $j$. We aim to choose one path from $P_i$ for each $i$, in order to minimize the *relative congestion*: the minimal $T$ such that the maximum load on any edge $f$ is at most $T \cdot c_f$. We get a similar IP formulation:

$$\text{minimize } T \text{ subject to } \left[ \forall i,\ \sum_{j} x_{i,j} = 1;\ \forall f \in E,\ \sum_{(i,j):\ f \in P_{i,j}} x_{i,j} \leq T \cdot c_f;\ x_{i,j} \in \{0,1\}. \right]$$

**Our class of problems.** Given the above two examples, we are ready to define the class of problems that we will study. As above, we have $n$ variables $X_1, X_2, \ldots, X_n$; we need to choose one value for each variable, which is modeled by "assignment constraints" (10) on the underlying indicator variables $x_{i,j}$. In addition, we have a set $\mathcal{B}$ of (undesirable) events; these are all conjunctions of the elementary events $X_i = j$; we aim to choose the variables $X_i, \ldots, X_n$, and such that all the $B \in \mathcal{B}$ are falsified. It is easily seen that our two applications above, have the undesirable events are defined by *linear* threshold functions of the form "$\sum_{i,j} a_{k,i,j} x_{i,j} > b_k$"; we will also consider bad-events which are non-linear, some of which will be crucial in our packet-routing application. We develop a *partial resampling* approach to our basic

problem in Section 5.1; Theorem 5.6 presents some general conditions under which our algorithm quickly computes a feasible solution $X_1, \ldots, X_n$.

A key technical tool in our analysis is a new formula for counting the tree-structures which appear in the proof of the MT algorithm. This greatly simplifies the analysis of the Asymmetric LLL. It is critical to obtaining usable formulas for complicated applications of the Partial Resampling framework, but it is also very useful for analyzing the standard Moser-Tardos framework.

Let us next motivate our result by describing three families of applications.

**5.0.3. The case of non-negative linear threshold functions.** The scheduling and routing applications had each $B_k$ being a non-negative linear threshold function: our constraints (i.e., the complements of the $B_k$) were of the form

$$\forall k \in [K],\ \sum_{i,j} a_{k,i,j} x_{i,j} \leq b_k. \tag{14}$$

(The matrix $A$ of coefficients $a_{k,i,j}$ here, has $K$ rows indexed by $k$, and some $N$ columns that are indexed by pairs $(i,j)$.) Recall that all our problems will have the assignment constraints (10) as well. There are two broad types of approaches for such problems, both starting with the natural LP relaxation of the problem, wherein we allow each $x_{i,j}$ to lie in $[0,1]$. Suppose the LP relaxation has a solution $\{y_{i,j}\}$ such that for all $k$, "$\sum_{i,j} a_{k,i,j} y_{i,j} \leq b'_k$", where $b'_k < b_k$ for all $k$; by scaling, we will assume throughout that $a_{k,i,j} \in [0,1]$. The natural question is:

> "What conditions on the matrix $A$ and vectors $b'$ and $b$ ensure that there is an integer solution that satisfies (10) and (14), which, furthermore, can be found efficiently?"

The first of the two major approaches to this is polyhedral. Letting $D$ denote the maximum column sum of $A$, i.e., $D = \max_{i,j} \sum_k a_{k,i,j}$, the rounding theorem of [**63**] shows constructively that for all $k$,

$$b_k = b'_k + D \tag{15}$$

suffices. The reader is asked to verify that given a solution to the LP-relaxation of makespan minimization that satisfies (10), (11) and (12), bound (15) implies that we can find a schedule with makespan at most $2T$ efficiently. This 2-approximation is the currently best-known bound for this fundamental problem, and what we have seen here is known to be an alternative to the other polyhedral proofs of [**74, 106**].

The second approach to our problem is randomized rounding [**96**]: given an LP-solution $\{y_{i,j}\}$, choose exactly one $j$ independently for each $i$, with the probability of $y_{i,j}$ of setting variable $i$ equal to $j$. The standard "Chernoff bound followed by a union bound over all $K$ rows" approach [**96**] shows that Chernoff$(b'_k, b_k) \leq 1/(2K)$ suffices, for our goal to be achieved with probability at least $1/2$. That is, there is some constant $c_0 > 0$ such that

$$(16)\quad b_k \geq c_0 \cdot \frac{\log K}{\log(2\log K/b'_k)} \text{ if } b'_k \leq \log K;\ b_k \geq b'_k + c_0 \cdot \sqrt{b'_k \cdot \log K} \text{ if } b'_k > \log K$$

suffices. In particular, the low-congestion routing problem can be approximated to within $O(\log K/\log\log K)$ in the worst case, where $K$ denotes the number of edges.

Let us compare these known bounds (15) and (16). The former is good when all the $b'_k$ are "large" (say, much bigger than, or comparable to, $D$ – as in the 2-approximation above for scheduling); the latter is better when $D$ is too large, but unfortunately does not exploit the sparsity inherent in $D$ – also note that $K \geq D$ always since the entries $a_{k,i,j}$ of $A$ lie in $[0, 1]$. A natural question is whether we can interpolate between these two: especially consider the case (of which we will see an example shortly) where, say, all the values $b'_k$ are $\Theta(1)$. Here, (15) gives an $O(D)$-approximation, and (16) yields an $O(\log K/\log\log K)$-approximation. Can we do better? We answer this in the affirmative in Theorem 5.19 – we are able to essentially replace $K$ by $D$ in (16), by showing constructively that if we have $b'_k \leq R$ for all $k$, for some $R \geq 1$, then we achieve

$$b_k = C_0 \frac{\log(D+1)}{1 + \log \frac{\log(D+1)}{R}} \text{ if } R \leq \log(D+1);\ b_k = R + C_0\sqrt{R\log(D+1)} \text{ if } R \geq \log(D+1)$$

suffices.

*Application to multi-dimensional scheduling.* Consider the following $D$-dimensional generalization of scheduling to minimize makespan, studied by Azar & Epstein [**14**]. Their $(D+1)$-approximation here also holds for the following generalization, and again follows quickly from (15). Here, when job $i$ gets assigned to machine $j$, there are $D$ dimensions to the load on $j$ (say runtime, energy, heat consumption, etc.): in dimension $\ell$, this assignment leads to a load of $p_{i,j,\ell}$ on $j$ (instead of values such as $p_{i,j}$ in [**74**]), where the numbers $p_{i,j,\ell}$ are given. Analogously to (10), (11) and (12), we ask here: *given a vector $(T_1, T_2, \ldots, T_D)$, is there an assignment that has a makespan of at most $T_\ell$ in each dimension $\ell$?* The framework of [**14**] and (15) gives a $(D+1)$-approximation[1] while our bound (5.0.3) yields an $O(\log D / \log\log D)$-approximation; since $D \ll K$ typically in this application, this is also a significant improvement over the $O(\log K / \log\log K)$-approximation that follows from (16).

*Comparison with other known bounds.* As described above, our bound (5.0.3) improves over the two major approaches here. However, two related results deserve mention. First, a bound similar to (5.0.3) is shown in [**72**], but with $D^*$, the maximum number of nonzeroes in any column of $A$, playing the role of $D$. Note that $D^* \geq D$ always, and that $D^* \gg D$ is possible. Moreover, the bound of [**72**] primarily works when all the $b'_k$ are within an $O(1)$ factor of each other, and rapidly degrades when these values can be disparate. Finally, our bound is better when $R \gg \log(D+1)$; other works give the bound in this case of $b_k = R + C_0\sqrt{R \log R}$.

**5.0.4. Transversals with omitted subgraphs.** Given a partition of the vertices of an undirected graph $G = (V, E)$ into blocks (or *classes*), a *transversal* is a subset of the vertices, one chosen from each block. An *independent transversal*, or independent system of representatives, is a transversal that is also an independent set in $G$. The study of independent transversals was initiated by Bollobás, Erdős

[1]As usual in this setting, an "approximation" here is an algorithm that either proves that the answer to this question is negative (by showing that the LP is infeasible), or presents the desired approximation simultaneously for each $T_\ell$.

& Szemerédi [**18**], and has received a considerable amount of attention (see, e.g., [**2, 5, 7, 54, 59, 60, 75, 111, 116**]). Furthermore, such transversals serve as building blocks for other graph-theoretic parameters such as the linear arboricity and the strong chromatic number [**5, 7**]. We improve (algorithmically) a variety of known sufficient conditions for the existence of good transversals, in Section 5.3. In particular, Szabó & Tardos present a conjecture on how large the blocks should be, to guarantee the existence of transversals that avoid $K_s$ [**111**]; we show that this conjecture is true asymptotically for large $s$. We also study weighted transversals, as considered by Aharoni, Berger & Ziv [**2**], and show that near-optimal (low- or high-) weight transversals exist, and can be found efficiently.

**5.0.5. Packet routing with low latency.** A well-known packet-routing problem is as follows. We are given an undirected graph $G$ with $N$ packets, in which we need to route each packet $i$ from vertex $s_i$ to vertex $t_i$ along a *given simple path* $P_i$. The constraints are that each edge can carry only one packet at a time, and each edge traversal takes unit time for a packet; edges are allowed to queue packets. The goal is to conduct feasible routings along the paths $P_i$, in order to minimize the *makespan* $T$, the relative of the scheduling notion above that refers to the time by which all packets are delivered. Two natural lower-bounds on $T$ are the *congestion* $C$ (the maximum number of the $P_i$ that contain any given edge of $G$) and the *dilation* $D$ (the length of the longest $P_i$); thus, $(C + D)/2$ is a universal lower-bound, and there exist families of instances with $T \geq (1 + \Omega(1)) \cdot (C + D)$ [**97**]. A seminal result of [**73**] is that $T \leq O(C + D)$ for all input instances, using constant-sized queues at the edges; the big-Oh notation hides a rather large constant. Further work [**99, 93**] improved this constant term substantially, leading to the bound $23.4(C + D)$ given in [**93**]. We improve these further to a constructive $5.70(C + D)$ here.

**Informal discussion of the Partial Resampling Algorithm.** To understand the intuition behind our Partial Resampling Algorithm, consider the situation in which

we have bad events of the form $Z_1 + \cdots + Z_v \geq \mu + t$, where the expected value of $Z_1 + \cdots + Z_v$ is $\mu$. There are two basic ways to set this up for the standard LLL. The most straightforward way would be to construct a single bad-event for $Z_1 + \cdots + Z_v \geq \mu + t$. In this case, the single event would depend on $v$ variables, which might be very large. Alternatively, one could form $\binom{v}{\mu+t}$ separate bad-events, corresponding to every possible set of $\mu + t$ variables. Each of these bad-events individually would have a very low probability, and the overall dependency would also be low. The problem with this approach is that the collective probability of the bad-events has become very large. In effect, one is approximating the probability of the bad-event $Z_1 + \cdots + Z_v \geq \mu + t$ by a union-bound over all $\binom{v}{\mu+t}$ subsets. When $t$ is small, this union bound is very inaccurate.

In fact, both of these approaches are over-counting the dependence of the bad-event. In a sense, a variable $Z_i$ is causing the bad-event only if it is "the straw that breaks the camel's back," that is, if it is the key variable which brings the sum $Z_1 + \cdots + Z_v$ over the threshold $\mu + t$. Really, only about $t$ of the variables are "guilty" of causing the bad-event. The first $\mu$ variables were expected to happen anyway; after reaching a total $\mu + t$ variables, any remaining variables are redundant. Any individual variable $Z_i$ only has a small chance of being a guilty variable.

This is a situation in which there are many variables which have have some effect on the bad event, but this effect is typically very small. The standard LLL, which is based on a binary classification of whether a variable affects a bad-event, cannot see this. We will give a new criterion which quantifies how likely a variable is to be responsible for the bad-event. This will in turn greatly lower the dependency between bad events, while still keeping an essentially accurate bound on the probability. Further specific comparisons with the standard LLL are made in Sections 5.3 and 5.4.

In general, the Partial Resampling Algorithm tends to work well when there are common configurations, which are not actually forbidden, but are nonetheless "bad" in the sense that they are leading to a forbidden configuration. So, in the case of a

sum of random variables, if a large group of these variables is simultaneously one, then this is bad but, by itself, still legal. We will see other examples of more complicated types of bad-but-legal configurations.

**Organization of the chapter.** The Partial Resampling Algorithm is discussed in detail in Section 5.1. We give a criterion, similar to the asymmetric LLL, for showing that this algorithm terminates in expected polynomial time.

Section 1.6 describes an alternative, more succinct formulation of the "weighting function" necessary to analyze the PRA. This method is very useful for the standard MT algorithm as well, but it is particularly important for the PRA, because the criterion we develop in Section 5.1 has a huge number of parameters and is difficult to work with directly.

A related but non-constructive framework, which is able to apply to similar types of combinatorial configurations with qualitatively similar result, is presented in the appendix in Section 5.C.

Applications are discussed in the following three sections: transversals with omitted subgraphs, improved integrality gaps for column-sparse packing problems, and packet routing in Sections 5.3, 5.4, and 5.5 respectively. The appendix contains numerical results and a useful probabilistic lemma.

## 5.1. The Partial Resampling Algorithm

**5.1.1. Notation.** We begin by discussing some basic definitions that will apply throughout.

We have $n$ variables; each variable has a set of possible assignments, which may be countably infinite or finite, and which we identify with (a subset of) the integers. We define the probability space $\Omega$, in which the variables are assigned independently: for each variable $i$ we set $X_i = j$ with probability $p_{i,j}$, where where $j$ ranges over the set of valid assignment to variable $i$. Henceforth we will not be explicit about the set of possible assignments, so we write simply $\sum_j p_{i,j} = 1$.

We refer to any ordered pair $(i, j)$ where $j$ is an assignment of variable $i$, as an *element*. We let $\mathcal{X}$ denote the set of all element. Given any vector $\vec{\lambda} = (\lambda_{i,j})$ indexed by elements $(i, j) \in \mathcal{X}$, we define, for any set $Y \subseteq \mathcal{X}$,

$$\lambda^Y = \prod_{(i,j)\in Y} \lambda_{i,j}. \tag{17}$$

**Atomic events as sets of elements.** We are also given a collection $\mathcal{B}$ of bad-events. For our purposes, we may suppose that all the events in $\mathcal{B}$ are *atomic events*; that is, each bad-event is defined by

$$B \equiv X_{i_1} = j_1 \wedge \cdots \wedge X_{i_l} = j_l$$

We slightly abuse notation so this atomic event would be represented by the set $\{(i_1, j_1), \ldots, (i_l, j_l)\}$. Thus, when we write $(i, j) \in B$, we mean that a necessary condition for bad-event $B$ to be true is that $X_i = j$.

In many applications of the LLL, the bad-events may be more complex. However, we can always write a complex bad event as a union of a (possible large) number of atomic bad-events. For example, if a bad-event is that $Z_1 + \cdots + Z_v \geq \mu + t$, then this can be represented as $\binom{v}{\mu+t}$ separate atomic events.

**Labeling bad-events.** For technical reasons, which we will discuss more in Section 5.1.5, we may sometimes need to attach labels in the range $\{1, \ldots, K\}$ to each atomic bad-event. Thus, we suppose there is a la belling map $\mathcal{L} : \mathcal{B} \to [K]$. We will use the notation

$$\mathcal{B}_k = \{B \in \mathcal{B} \mid \mathcal{L}(B) = k\}$$

To provide a very brief explanation of the role played by labels: in many applications of the LLL, there are multiple complex bad-events. We often want to analyze each complex bad-event separately. By adding labels to each complex bad-event, we will be able to limit their interactions with each other. **In reading the following sections, the reader is advised to keep in mind the case when $K = 1$ (so**

**that essentially the la belling map does not play a part). Almost all of the intuition and results apply in that setting, and many of our applications will use it as well. We refer to the case when $K = 1$ as the *trivial la belling.***

**5.1.2. Fractional hitting-sets.** In order to use our algorithm, we will need to specify an additional parameter. For each $k = 1, \ldots, K$ we must specify a *fractional hitting-set* $Q_k$, which essentially tells us how to resample the variables in that bad event.

DEFINITION 5.1. *Suppose $C : 2^{\mathcal{X}} \to [0, 1]$ is some weight function on the subsets of $\mathcal{X}$.*

*Let $B$ be an atomic bad-event, which we view as a set of elements $B = \{(i_1, j_1), \ldots, (i_r, j_r)\}$. We say that $C$ is a* fractional hitting set for $B$ *if we have that*

$$\sum_{Y \subseteq B} C(Y) \geq 1 \tag{18}$$

*Suppose $\mathcal{B}$ is a set of atomic bad-events. We say that $C$ is a* fractional hitting set for $\mathcal{B}$ *if $C$ is a fractional hitting set for all $B \in \mathcal{B}$.*

**Remark regarding Definition 5.1.** We may assume, without loss of generality, that if $Y$ contains more than one value for a variable (that is, if it contains $(i, j)$ and $(i, j'))$ then $C(Y) = 0$.

If there are $K$ possible labels, we will select $K$ fractional hitting sets $Q_1, \ldots, Q_K$; for each $k = 1, \ldots, K$ we require that $Q_k$ is a fractional hitting set for $\mathcal{B}_k$.

One possible choice for the fractional hitting set is the bad-event itself, which is easily verified to be a valid fractional hitting-set:

DEFINITION 5.2. *Let $\mathcal{B}$ be a set of bad-events. We define the* trivial hitting-set *for $\mathcal{B}$ by*

$$C(Y) = \begin{cases} 1 & \text{if } Y \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases}$$

If the trivial hitting-set is used, then our analysis essentially reduces to the ordinary MT algorithm (but we will still show improvements in that case). We will discuss later how to construct such fractional hitting-sets, but for now we suppose that we are provided some specified $Q_1, \ldots, Q_K$.

DEFINITION 5.3 (A supported event). *Given any $Y \subseteq X$ and $k \in [K]$, we say that $(Y, k)$ is* supported *if $Q_k(Y) > 0$.*

**5.1.3. Partial Resampling Algorithm and the Main Theorem.** We present our partial resampling algorithm, and main theorem related to it.

Consider the following relative of the MT algorithm. We refer to this as the *Partial Resampling Algorithm* or abbreviate as *PRA*.

1. Draw $X_1, \ldots, X_n \sim \Omega$
2. Repeat the following, as long as there is some true bad-event $B \in \mathcal{B}$:
    3. Select, arbitrarily, some true bad-event $B \in \mathcal{B}$. We refer to this set $B$ as the *violated set*.
    4. Select exactly one subset $Y \subseteq A$. The probability of selecting a given $Y$ is given by

$$P(\text{select } Y) = \frac{Q_k(Y)}{\sum_{Y' \subseteq B} Q_k(Y')}$$

    where $k = \mathcal{L}(B)$. We refer to $Y$ as the *resampled set*.
    5. Resample all the variables involved in $Y$ independently from $\Omega$.

This differs from the MT algorithm in that if there is a bad event that is currently true, then MT would resample *all* the variables which it depends upon. Here, we only resample a (carefully-chosen, random) subset of these variables.

We will need to keep track of the dependency graph corresponding to our fractional hitting set. This is more complicated than the usual Moser-Tardos setting, because we will need to distinguish two ways that subsets of elements $Y, Y'$ could affect each other: they could share a variable, *or* they could both be potential resampling targets for some bad-event. In the usual Moser-Tardos analysis, we only need to keep track of the first type of dependency. The following symmetric relation $\approx$ (and its two supporting relations $\sim$ and $\bowtie$) will account for this:

DEFINITION 5.4. ***(Symmetric relations $\sim$, $\bowtie_k$, and $\approx$)*** *Let $Y, Y' \subseteq \mathcal{X}$.*

*We say $Y \sim Y'$ iff there exists a triple $(i, j, j')$ such that $(i, j) \in Y$ and $(i, j') \in Y'$: i.e., iff $Y$ and $Y'$ overlap in a variable. We also write $i \sim Y$ (or $Y \sim i$) to mean that $Y$ involves variable $i$ (i.e., $(i, j) \in Y$ for some $j$.)*

*For each $k$, we say $Y \bowtie_k Y'$ iff $Y \not\sim Y'$ and there is some event $B \in B_k$ with $Y, Y' \subseteq B$.*

*Relation $\approx$ is defined between pairs $(Y, k)$: We define $(Y, k) \approx (Y', k')$ iff $Y \sim Y'$ or $Y \bowtie_k Y'$. Note that, by definition, it is impossible for both to occur simultaneously.*

*We say that the $\bowtie$ relations are* null *if for all $Y, Y'$ with $Q_k(Y) > 0, Q_k(Y') > 0$ we have $Y \not\bowtie_k Y'$.*

We give three conditions for the PRA to terminate. These conditions are analogous to, respectively, the cluster-expansion criterion, the asymmetric LLL, and the symmetric LLL. In order to state the conditions, we introduce some definitions:

DEFINITION 5.5. ***(A neighbor-set for $(Y, k)$)*** *Given any $Y \subseteq X$ and $k \in [K]$ and a set $\mathcal{T} \subseteq 2^{\mathcal{X}} \times [K]$, we say that $\mathcal{T}$ is a* neighbor-set *for $(Y, k)$ if the following conditions hold:*

(1) *For all $(Z, l) \in \mathcal{T}$ we have $Z \approx Y$.*

(2) *There do not exist $(Z, l), (Z', l') \in \mathcal{T}$ with $Z \sim Z'$*

(3) *There is* at most *one element $(Z, k) \in \mathcal{T}$ satisfying $Z \bowtie_k Y$*

THEOREM 5.6. ***(Main PRA Theorem)*** *Suppose we are given fractional hitting sets $Q_1, \ldots, Q_K$ for $\mathcal{B}_1, \ldots, \mathcal{B}_K$ respectively.*

*In each of the following three cases, the PRA terminates in a feasible configuration avoiding all bad events with probability one.*

*(a) Suppose there exists $\mu : 2^{\mathcal{X}} \times [K] \to [0, \infty)$ which satisfies, for all $Y \subseteq \mathcal{X}$ and all $k \in [K]$,*

$$\mu(Y,k) \geq p^Y Q_k(Y) \sum_{\mathcal{T} \text{ a neighbor-set for } Y} \prod_{(Y,k') \in T} \mu(Y',k')$$

*Then, the expected number of resamplings is at most $\sum_{(Y,k)} \mu(Y,k)$.*

*(b) Suppose there exists $\mu : 2^{\mathcal{X}} \times [K] \to [0, \infty)$ which satisfies, for all $Y \subseteq \mathcal{X}$ and all $k \in [K]$,*

$$\mu(Y,k) \geq p^Y Q_k(Y) \Big( \prod_{Y' \sim Y} (1 + \sum_{k' \in [K]} \mu(Y',k')) \Big) \Big( 1 + \sum_{Y'' \bowtie_k Y} \mu(Y'',k) \Big)$$

*Then, the expected number of resamplings is at most $\sum_{(Y,k)} \mu(Y,k)$.*

*(c) Suppose that for all $Y \in 2^{\mathcal{X}}, k \in [K]$ we have $p^Y Q_k(Y) \leq P$; and suppose that for all supported $(Y,k)$, there are* at most *$D$ supported $(Y',k')$ with $(Y',k') \approx (Y,k)$ (note that we allow here $(Y',k') = (Y,k)$.) And suppose finally that we satisfy the criterion*

$$ePD \leq 1$$

*Then, the expected number of resamplings is at most $e \sum_{(Y,k)} p^Y Q_k(Y)$.*

**5.1.4. Proof ingredient: Witness Trees.** A key component of our proofs will be the notion of witness trees as we have introduced in Chapter 2, but with a small but critical difference. Suppose we run the PRA and encounter bad-events $B_1, \ldots, B_t$; for each of these we resample the set $Y_1, \ldots, Y_t$. We define the *execution log* of this algorithm to be the listing $(Y_1, \mathcal{L}(B_1)), \ldots, (Y_t, \mathcal{L}(B_t))$. It is crucial to note that we do *not* list the violated sets $B$ themselves (only their labels). Not listing the violated

sets themselves, is one of our critical ideas, and helps prune the space of possible witness trees substantially.

So the execution log is a series of pairs $(Y_1, k_1), \ldots, (Y_t, k_t)$. We form the witness tree in in a manner similar to that given in Chapter 2, driven by the relation $\approx$, but there is a key difference. Each node $(Y, k)$ is only allowed to have a *single* child due to the relation $\bowtie_k$, and has *no children* due to the relation $\bowtie_{k'}$ for $k' \neq k$. As we will see, a single $\bowtie$-child provides all the information needed. A much simpler proof for the critical Witness Tree Lemma could be obtained if we allow for multiple $\bowtie_k$-children, but having this stronger form of the Witness Tree Lemma pays off later with simpler and stronger results.

Here is how we build a witness tree for an entry in execution log $(Y^*, k^*)$ (for example, the final resampling). The event $(Y^*, k^*)$ goes at the root of the tree. This, and all nodes of the tree, will be labeled by the corresponding pair $(Y, k)$. Stepping backward in time, suppose the current event being processed is labeled $(Y, k)$. Suppose that either there is a node $(Y', k')$ in the current tree with $Y' \sim Y$, *or* there is an node $(Y', k)$ and $Y \bowtie_k Y'$ *and the node* $(Y', k)$ *does not currently have a child* $(Y'', k)$ *with* $Y'' \bowtie_k Y$. In this case, we find the node $v$ of lowest level (i.e. highest depth) which satisfies either of these conditions, and make the new node labeled $(Y, k)$ a child of the node $v$. If there are no nodes satisfying either condition, we skip $(Y, k)$. Continue this process going backward in time to complete the construction of the witness tree.

Because this will come up repeatedly in our discussion, given a node $v$ labeled by $(Y, k)$, we refer to a child $(Y', k)$ with $Y \bowtie_k Y'$ as a $\bowtie$-*child.* If a node $v$ has a $\bowtie$-child $v'$ we say it $v$ *is saturated by* $v'$ otherwise we say $v$ is *unsaturated.* Evidently in this construction each node may have one or zero $\bowtie$-children.

As in [**94**], note that all nodes in any level of the witness tree must form an independent set under $\sim$; this will be useful in our analysis.

We next introduce our main Lemma 5.7, connecting the witness trees to the execution logs. However, as pointed out after the proof of the lemma, its proof is much more involved than Lemma 2.8.

LEMMA 5.7 (Witness Tree Lemma). *Let $\tau$ be any tree-structure, with nodes labeled by $(Y_1, k_1), (Y_2, k_2) \ldots, (Y_t, k_t)$. Then the probability that the witness tree $\tau$ appears is at most*

$$P(\tau \text{ appears}) \leq \prod_{s=1}^{t} p^{Y_s} Q_{k_s}(Y_s).$$

**Remark.** Recall the notation (17) in parsing the value "$p^{Y_s}$" above.

PROOF. We will construct a necessary conditions for the tree $\tau$ to appear. These conditions all have the following form: they contain some conditions on the past history of the PRA; as soon as these conditions are triggered, we demand that the random variable which is about to be sampled takes on a specific value. We refer to the preconditions of each of these checks as *triggers* and the demanded event to be the *target*. The target will be either that some variable $i$ we are about to resample takes on a specific value $j$; or that when we detect a bad-event $B$, we select some specified $Y \subseteq B$ to resample. For the tree $\tau$ to appear, it must be the case that each trigger is detected exactly once in the execution log and no two triggers occur simultaneously. In this case, Lemma 5.32 from Appendix 5.B can be used to show that the probability of this event is at most the product of the individual target probabilities, namely $p_{i,j}$ and $Q_{\mathcal{L}(B)}(Y)$ respectively.

Recall that we do an initial sampling of all categories (which we can consider the zeroth resampling), followed by a sequence of resamplings. Now, consider a node $s$ of $\tau$ labeled by $(Y, k)$. Suppose $(i, j) \in Y$. Because any node – labeled $(Y', k')$, say – in which $i$ is sampled would satisfy $(Y, k) \sim (Y', k')$, it must be that any earlier resamplings of the variable $i$ must occur at a lower level of the witness tree. So if we let $r$ denote the number of times that variable $i$ has appeared in lower levels of $\tau$, then we are demanding that the $r^{\text{th}}$ resampling of variable $i$ selects $j$. Such a condition

has the form we mentioned earlier; the trigger is that we have come to resample $i$ for the $r^{th}$ time, and the target is that we select value $j$. The probability of this is $p_{i,j}$.

We next consider the probability of selecting set $Y$. Consider all the nodes $(Y_1, k_1), \ldots, (Y_l, k_l) \approx (Y, k)$ of the witness tree which are at a lower level than $Y$. To simplify the discussion, we will suppose that $Y_i$ are distinct; if not, we would simply have to count events with their appropriate multiplicities.

We divide these nodes into four mutually exclusive categories:

(1) $Y_i \sim Y$

(2) $k_i = k$ and $Y_i \bowtie_k Y$ and $(Y_i, k)$ is unsaturated

(3) $k_i = k$ and $Y_i \bowtie_k Y$ and $(Y_i, k)$ is saturated by $Y_i'$ and $(Y_i, k)$ is the $\bowtie_k$-child of $(Y, k)$.

(4) $k_i = k$ and $Y_i \bowtie_k Y$ and $(Y_i, k)$ is saturated by $Y_i'$ and $(Y_i, k)$ is not the $\bowtie_k$-child of $(Y, k)$.

Define a *potential time* for $(Y, k)$ as some time in the execution of the PRA such that:

- For all $Y_i$ in cases (1), (2), (3), then $Y_i$ has been resampled.
- For all $Y_i$ in case (4), then either $Y_i'$ has not been resampled or $Y_i$ and $Y_i'$ have been resampled.

Note that the potential times are not necessarily contiguous in time — these conditions can flip multiple times between satisfied and unsatisfied.

Given a bad-event $B$, we say that $Y$ is *eligible* for $B$ if $\mathcal{L}(B) = k$ and $Y \subseteq B$. If we are a state in the PRA at which we have selected some bad-event $B \supseteq Y$, but not yet chosen the resampled set, then we say that $Y$ is *eligible* (omitting the implied dependency on $B$).

We begin by claiming that $Y$ must have been selected during a potential time. For, suppose that there is a node $Y_i$ of type (1), but $Y$ occurred before $Y_i$. Then, forming the witness tree in the backward manner, by the time $Y$ is added to the witness tree, $Y_i$ is already present. So $Y$ would be placed below $Y_i$, contradicting the

way we have defined the enumeration. A similar argument is seen to apply to nodes of type (2). For nodes of type (3), the reasoning is simpler: the $\bowtie_k$ child always occurs earlier in time than its parent. For a node of type (4), we suppose for contradiction that we encounter $Y$ after $Y_i'$ but before $Y_i$. So the sequence of resamplings must be $Y_i', Y, Y_i$. So when $Y$ is added to the tree, node $Y_i$ is present but lacks its $\bowtie_k$-child. So $Y$ would be placed beneath $Y_i$ (or lower in the tree).

We now claim that in the first situation during a potential time in which $Y$ is eligible we must resample $Y$. For, suppose not. In order for $\tau$ to appear we must eventually resample $Y$; so there must be multiple situations during the execution of the PRA which are potential times in which $Y$ is eligible, and in one of those times (but not the first) we must resample $Y$. Suppose that we resample the set $Y$ during the $r$th such situation; let $Y''$ be the set that was resampled in the $r-1$st such situation (this is well-defined as $r \neq 1$). As $Y$ is eligible at this time, $Y''$ must occur with label $k$.

We will show that $(Y'', k'')$ must be placed below $(Y, k)$ in the witness tree. This is obvious if $Y'' \sim Y$ so suppose $Y'' \bowtie_k Y$. First suppose that $(Y, k)$ has no $\bowtie_k$ child. In this case, $(Y'', k)$ could be placed as a $\bowtie_k$ child of $Y$ (or deeper in the tree).

Next, suppose that $Y$ has a $\bowtie_k$-child $Y'$ in $\tau$. So $Y'$ is a node of type (3) for $Y$. As $Y''$ is resampled during a potential time, it must be that $Y'$ is resampled before $Y''$. So the sequence of resamplings must be $Y', Y'', Y$. In that case, going backward to form the witness tree, at the time $Y$ is created its $\bowtie_k$-child $Y'$ has not been added. So again $(Y'', k)$ could be placed in the tree under $(Y, k)$.

In any of these cases, the node $(Y'', k)$ appears below $Y$ in the witness tree. Also, $(Y'', k) \approx (Y, k)$. So $Y''$ appears in the listing $(Y_1, k_1), \ldots, (Y_l, k_l)$; i.e. $Y'' = Y_s$ for some $s \in [l]$.

Suppose that $Y'' \sim Y$. Then node $(Y'', k)$ is of type (1) in the above listing. This implies that in any potential time for $Y$, it must be that $Y''$ has already been

resampled. But node $Y''$ is resampled during a potential time, i.e. node $Y''$ is selected *after* node $Y''$ is resampled which is a contradiction.

This same argument applies for type (2) or type (3) in the above listing. The only complicated case is type (4): that is, $Y'' \bowtie_k Y$ and $Y''$ is saturated by some by $Y'''$ and $Y''$ is not the $\bowtie_k$ child of $(Y, k)$. We are assuming that $Y''$ is resampled during a potential time for $Y$. By definition, this implies that either $Y'''$ has not been resampled *or* both $Y'', Y'''$ have been resampled. These are both contradictions.

In summary, we have shown that the first potential time in which $Y$ is eligible for some $B \in \mathcal{B}_k$, then we in fact resample $Y$. Critically, the target of this condition has a probability of at most $Q_k(Y)$ due to (18), for any bad-event $B$.

It is clear that each condition of the first kind, refers to a distinct target. The second type of event must also — for if $(Y, k)$ and $(Y', k')$ are selected in the same event, then $k = k'$ and $Y = Y'$ and one of these nodes would have placed above the other in the witness tree. Observe that if two identical nodes appear in the witness tree, then our conditions state that one must be selected before the other.

By Lemma 5.32, taking the product over all such conditions, the total probability is at most $\prod p^{Y_s} Q_{k_s}(Y_s)$. The key point in this proof is that, based on the execution of the PRA and the information in the tree $\tau$, we can determine exactly when each $Y_s, k_s$ should have been selected and what these should be resampled to. $\square$

The PRA and its proof are very similar to the MT algorithm, and it is tempting to view this as a special case of that algorithm. However, the proof Lemma 2.8, which is the analog of Lemma 5.7 for the MT algorithm uses a quite different argument based on coupling. The tree-structure $\tau$ imposes necessary conditions on these samplings, whose probability distribution can be computed easily. While this coupling argument works for the value selected in each variable, it does not appear to work for bounding the probability of selecting a given $Y$. For this, we appear to need the "Nostradamus Lemma": Lemma 5.32 from Appendix 5.B.

For any tree $\tau$, we define its weight

$$w(\tau) = \prod_s p^{Y_s} Q_{k_s}(Y_s).$$

In order to show that the PRA terminates, we bound the total weight of all tree-structures. This is very similar to the calculation in Moser & Tardos, so we give a sketch here:

PROPOSITION 5.8. *Suppose we are given a weighting function satisfying Theorem 5.6(a). Then the total weight of all tree-structures with root labeled $(Y, k)$ is at most $\mu(Y, k)$.*

PROOF. For any $Y \subseteq X, k \in [K]$ let $T_h(Y, k)$ be the total weight of all tree-structures with root node labeled $(Y, k)$ of height at most $h$. One can show by induction on $h$ that $T_h(Y, k) \leq \mu(Y, k)$. To show this, note that the children of the root node must receive distinct labels $(Z_1, l_1), \ldots, (Z_r, l_r)$, and that $\{(Z_1, l_1), \ldots, (Z_r, l_r)\}$ must be a neighbor-set for $(Y, k)$. □

Next, note that for any distinct times $t < t'$, the witness trees $\hat{\tau}^t, \hat{\tau}^{t'}$ are distinct. Thus, the expected number of resamplings is at most the expected number of witness trees which appear, which is at most the sum of the weights of all witness trees, which is at most $\sum_{(Y,k)} \mu(Y, k)$. This immediately shows Theorem 5.6(a).

We can derive Theorem 5.6(b), by using the following method to enumerate neighbor-sets $\mathcal{T}$. First, we put into $\mathcal{T}$ either one element $(Y'', k)$ with $Y'' \bowtie_k Y$, or no such elements; this contributes a factor $\left(1 + \sum_{Y'' \bowtie_k Y} \mu(Y'', k)\right)$. Next, for any $Y' \sim Y$, we place $(Y', k')$ into $\mathcal{T}$, for at most one choice of $k$. (Observe that $\mathcal{T}$ cannot contain both $(Y', k'_1)$ and $(Y', k'_2)$ by definition of a neighbor-set). For any $Y' \sim Y$, this contributes the term $1 + \sum_{k' \in [K]} \mu(Y', k')$. This produces every neighbor-set $\mathcal{T}$, but some of the sets produced in this way are not neighbor-sets; thus this is an

over-estimate. So we have

$$\sum_{\mathcal{T} \text{ a neighbor-set for } Y} \prod_{(Y,k')\in T} \mu(Y',k') \leq \Big(\prod_{Y'\sim Y}(1+\sum_{k'\in[K]}\mu(Y',k'))\Big)\Big(1+\sum_{Y''\bowtie_k Y}\mu(Y'',k)\Big)$$

Finally, we can derive Theorem 5.6(c) by setting $\mu(Y,k) = ep^Y Q_k(Y)$ for all $Y, k$.

**5.1.5. Complex bad-events and the role of labels.** Now that we have developed our PRA framework, we can clarify the role played by the labeling function $\mathcal{L} : \mathcal{B} \to [K]$.

Suppose, as is often the case in applying the LLL, that we have multiple *complex* bad-events. We say a bad-event is complex if it is not an atomic event (a conjunction of elements). For example, a bad-event might be defined in terms of a linear threshold $Z_1 + \cdots + Z_v \geq \mu + t$. Any complex bad-event can be expressed as a disjoint union (possibly exponentially sized) of atomic bad-events. For example, the linear threshold is definable as $\binom{v}{\mu+t}$ separate atomic bad-events. This decomposition is not necessarily unique, but we suppose we have fixed one particular choice for this decomposition.

So, we may write our family of bad events $\mathcal{B}$ as $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \cdots \cup \mathcal{B}_K$; here, each $\mathcal{B}_k$ is a complex bad-event, which is a set of atomic bad-events.

At this point, we might want to analyze each $\mathcal{B}_k$ separately, deriving an appropriate fractional hitting set $Q_k$ and computing some measure of "badness" which we can aggregate over all $\mathcal{B}$. When we are applying Theorem 5.6, the linkages due to $\sim$ are relatively easy to handle in this way — it is very similar to the situation for the usual asymmetric LLL. But the linkages due to $\bowtie$ may become hard to handle. In particular, we may have some $B \in \mathcal{B}_k$ with $Y, Y' \subseteq B$; in this case, the sets $Y, Y'$ become dependent with each other in a complicated, non-linear way.

As is suggested by the notation, we can decouple the events $\mathcal{B}_k$ by defining an appropriate labeling:

$$\forall B \in \mathcal{B} \quad \mathcal{L}(B) = \min_{B\in\mathcal{B}_k} k$$

We can now state a version of the Theorem 5.6(a) which involves no non-linear interactions between the separate bad-events. These results are almost identical to Theorem 5.6(a); the only difference is than in Theorem 5.6, we assume that the sets $\mathcal{B}_1, \ldots, B_K$ are disjoint while here they may overlap in arbitrary ways. However, in fact the overlap between bad-events $\mathcal{B}_1, \ldots, B_K$ can only help us.

THEOREM 5.9. *Suppose we are given complex bad-events $\mathcal{B}'_1, \ldots, B'_K$ (which are not necessarily disjoint), along with fractional hitting sets $Q_1, \ldots, Q_K$ for $\mathcal{B}'_1, \ldots, \mathcal{B}'_K$ respectively. Suppose there exists $\mu : 2^{\mathcal{X}} \times [K] \to [0, \infty)$ which satisfies, for all $Y \subseteq \mathcal{X}$ and all $k \in [K]$,*

$$\mu(Y,k) \geq p^Y Q_k(Y) \sum_{\mathcal{T} \text{ a neighbor-set for } Y} \prod_{(Y,k') \in T} \mu(Y',k')$$

*Then, the expected number of resamplings is at most $\sum_{(Y,k)} \mu(Y,k)$.*

PROOF. It is easy to verify that $\mathcal{B}_k \subseteq \mathcal{B}'_k$ for each $k \in [K]$. Thus, $\mu$ still satisfies Theorem 5.6(a). □

There is one further complication, which is really a quite minor technical point. Suppose that $\mathcal{B}_{k_1}, \mathcal{B}_{k_2}$ both contain some atomic event $B$. To apply Theorem 5.9, we must assign the label of $B$ to $\min(k_1, k_2)$. This means that when we encounter bad-event $B$, we must use the fractional hitting set $Q_{\min(k_1,k_2)}$. This can actually present some computational difficulties. For example, suppose that during the execution of the PRA, we know that $B \in \mathcal{B}_k$ is currently true. We would like to resample $Y \subseteq B$ according to the fractional hitting set $Q_k$; but it may in fact be the case that $B \in \mathcal{B}_{k'}$ for some $k' < k$. Furthermore, if $K$ is large, it may take exponential time to decide if $B \in B_{k'}$ for $k' < k$.

In fact, everything works if when we encounter a bad event $B \in \mathcal{B}_1 \cup \cdots \cup B_K$, we select some *arbitrary* $k$ with $B \in \mathcal{B}_k$ and resample $B$ according to $Q_k$. To prove this, we must generalize the labeling function so that, for all bad-events $B \in \mathcal{B}$, we

have that $L(B)$ is a *non-empty subset of* $[K]$. But this substantially complicates the notation and proofs.

### 5.2. Enumeration of witness trees by variables

For many applications of the PRA, in which the fractional hitting-sets are relatively simple, Theorem 5.6, in one of its three forms, is sufficient. However, it can be awkward to use because it requires us to analyze the very large space $2^{\mathcal{X}} \times [K]$, and check a constraint for every $Y, k$. We contrast this with the LLL (in its symmetric and asymmetric forms) which requires us only to check a constraint for each *bad-event*. For these reasons, the variable-based accounting method developed in Section 1.6 can be particularly useful for the PRA. However, there are a few slight changes that must be made to take into account the $\bowtie$ relations.

Suppose that are given an assignment of non-negative real numbers $\lambda_{i,j}$ to each element $(i,j) \in \mathcal{X}$. For any variable $i$, define $\lambda_i = \sum_j \lambda_{i,j}$. We will derive a probability distribution $p$ and a weighting function $\mu$ from this vector, and show corresponding conditions of the vector $\lambda$ to ensure that the PRA terminates.

DEFINITION 5.10. ***(Values $G_{i,j}, G_i, G$ that depend on a function $Q$ and a vector $\lambda$)*** *Suppose we are given an assignment of non-negative real numbers $\lambda = \lambda_{i,j}$ to each element $(i,j) \in \mathcal{X}$. For $Q : 2^{\mathcal{X}} \to [0,1]$, recalling the notation (17), we define*

$$G_{i,j}(Q, \lambda) = \sum_{Y \ni (i,j)} Q(Y) \lambda^Y$$

*along with "summation notations"*

$$G_i(Q,\ \lambda) = \sum_j G_{i,j}(Q, \lambda) \text{ and } G(Q, \lambda) = \sum_{i,j} G_{i,j}(Q, \lambda).$$

Roughly speaking, if $Q$ is a fractional hitting set for $\mathcal{B}$, then $G_{i,j}(Q, p)$ is the probability that variable $i$ takes on value $j$, and causes some bad-event in $\mathcal{B}$ to occur, and is selected for resampling.

In many settings, the linkages due to $\bowtie$ are relatively insignificant compared to the linkages due to $\sim$. One possible reason for this is that the $\bowtie$ linkage becomes null; this always occurs in the usual MT algorithm (without partial resampling). One of the main assumptions we make in this section is that the $\bowtie$ relations are relatively negligible. Thus, instead of carefully tracking them, we use a relatively crude estimate as follows:

DEFINITION 5.11. ***(Value $S_k$ that depends on a vector $\lambda$)*** *Suppose that we are given an assignment of non-negative real numbers $\lambda_{i,j}$ to each element $(i,j)$. For each $k$ and $Y$, suppose*

$$\sum_{Y' \bowtie_k Y} Q_k(Y')\lambda^{Y'} < 1$$

*We define, for each $k$, the parameter $S_k(\lambda) > 0$ to be*

$$S_k(\lambda) = \max_{Y:\ Q_k(Y)>0} \frac{1}{1 - \sum_{Y' \bowtie_k Y} Q_k(Y')\lambda^{Y'}} \tag{19}$$

Our main theorem is now as follows:

THEOREM 5.12. ***(Main PRA Theorem in terms of $\lambda$)*** *Suppose we are given fractional hitting sets $Q_1, \ldots, Q_K$ for $\mathcal{B}_1, \ldots, \mathcal{B}_K$ respectively and a vector $\lambda$. Suppose that we set the probability distribution for the variables by $p_{ij} = \frac{\lambda_{ij}}{\lambda_i}$.*

*(a) Suppose that*

$$\forall i,\ \lambda_i \geq 1 + \sum_k S_k(\lambda) G_i(Q_k, \lambda)$$

*Then the PRA terminates, and the expected number of resamplings is most $\sum_i \lambda_i$.*

*(b) Suppose that for all $k \in [K]$, $G(Q_k, \lambda) < 1$; suppose further that*

$$\forall i,\ \lambda_i \geq 1 + \sum_k \frac{G_i(Q_k, \lambda)}{1 - G(Q_k, \lambda)}.$$

*Then the PRA terminates, and the expected number of resamplings is most $\sum_i \lambda_i$.*

*(c) Suppose the $\bowtie_k$ relations are null (i.e. for supported $Y_1, Y_2$ we have $Y_1 \not\bowtie_k Y_2$). Suppose further that*

$$\forall i,\ \lambda_i \geq 1 + \sum_k G_i(Q_k, \lambda)$$

*Then the PRA terminates, and the expected number of resamplings is at most $\sum_i \lambda_i$.*

PROOF. We prove only (a). To show part (b), observe that we have $S_k(\lambda) \leq \frac{1}{1-G(Q_k,\lambda)}$. To show part (c), observe that if $\bowtie_k$ is null then we have $S_k(\lambda) = 1$.

We define the probability vector by $p_{ij} = \frac{\lambda_{ij}}{\lambda_i}$. We define the weighting function $\mu$ by

$$\mu(Y, k) = \lambda^Y Q_k(Y) S_k(\lambda)$$

We now wish to show that this satisfies Theorem 5.6(a). Consider some $Y = \{(i_1, j_1), \ldots, (i_r, j_r)\}$ and some $k \in [K]$; we may assume that $i_1, \ldots, i_r$ are distinct (as otherwise $Q_k(Y) = 0$.) We will show an upper bound on the expression

$$p^Y Q_k(Y) \sum_{\mathcal{T} \text{ a neighbor-set for } Y} \prod_{(Y,k') \in T} \mu(Y', k') \tag{20}$$

First, $\mathcal{T}$ may contain at most one choice $(Y', k)$ with $Y' \bowtie_k Y$; this contributes at most $1 + \sum_{Y' \bowtie_k Y} \mu(Y', k)$ which we estimate as:

$$\begin{aligned}
1 + \sum_{Y' \bowtie_k Y} \mu(Y', k) &= 1 + \sum_{Y' \bowtie_k Y} \lambda^{Y'} Q_k(Y') S_k(\lambda) \\
&= 1 + S_k(\lambda) \sum_{Y' \bowtie_k Y} \lambda^{Y'} Q_k(Y') \\
&= 1 + S_k(\lambda)(1 - 1/v) \qquad \text{where } v = \frac{1}{1 - \sum_{Y' \bowtie_k Y} \lambda^{Y'} Q_k(Y')} \\
&\leq 1 + S_k(\lambda)(1 - 1/S_k(\lambda)) \\
&= S_k(\lambda)
\end{aligned}$$

Next, for each $(i, j) \in Y$, then $\mathcal{T}$ may contain at most one choice $(Y', k')$ with $(i, j') \in Y'$. This is because if there are multiple such sets $(Y', k'), (Y'', k'')$ then we

would have $Y' \sim Y''$ which violates the definition of a neighbor-set. For any fixed $(i,j)$ in $Y'$, this contributes at most $1+\sum_{Y' \ni (i,j')} \sum_{k'} \mu(Y',k')$ which we may estimate as:

$$\begin{aligned} 1 + \sum_{Y' \ni (i,j')} \sum_{k'} \mu(Y',k') &= 1 + \sum_{Y' \ni (i,j'),k'} \lambda^{Y'} Q_k(Y') S_{k'}(\lambda) \\ &= 1 + \sum_{k'} G_i(Q_{k'},\lambda) S_{k'}(\lambda) \\ &\leq \lambda_i \qquad \text{by the hypothesis of Theorem 5.12(a).} \end{aligned}$$

Putting these two estimates together, we may estimate (20) by:

$$\begin{aligned} p^Y Q_k(Y) \sum_{\mathcal{T} \text{ a neighbor-set for } Y} \prod_{(Y,k') \in T} \mu(Y',k') &\leq p^Y Q_k(Y) S_k(\lambda) \prod_{(i,j) \in Y} \lambda_i \\ &= Q_k(Y) S_k(\lambda) \prod_{(i,j) \in Y} p_{ij} \lambda_i \\ &= Q_k(Y) S_k(\lambda) \prod_{(i,j) \in Y} \frac{\lambda_{ij}}{\lambda_i} \lambda_i \\ &= Q_k(Y) S_k(\lambda) \lambda^Y \\ &= \mu(Y,k) \end{aligned}$$

Thus, Theorem 5.6(a) holds. The expected number of resamplings is

$$\begin{aligned} \mathbf{E}[\#\text{ resamplings}] &\leq \sum_{Y,k} \mu(Y,k) \\ &\leq \sum_{i \in [n]} \sum_{j} \sum_{Y \ni (i,j),k} \lambda^Y Q_k(Y) S_k(\lambda) \\ &= \sum_{i \in [n]} \sum_{k} G_i(Q_k,\lambda) S_k(\lambda) \\ &\leq \sum_{i \in [n]} (\lambda_i - 1) \leq \sum_{i \in [n]} \lambda_i \end{aligned}$$

□

It is instructive to compare these formulas to those conditions of Theorem 5.34.

### 5.3. Transversals with omitted subgraphs

Suppose we are given a graph $G = (V, E)$ with a partition of its vertices into sets $V = V_1 \sqcup V_2 \sqcup \cdots \sqcup V_l$, each of size $b$. We refer to these sets as *blocks* or *classes*. We wish to select exactly one vertex from each block. Such a set of vertices $A \subseteq V$ is known as a *transversal*. There is a large literature on selecting transversals such that the graph induced on $A$ omits certain subgraphs. (This problem was introduced in a slightly varying form by [**18**]; more recently it has been analyzed in [**111, 59, 116, 60, 57**]). For example, when $A$ is an independent set of $G$ (omits the 2-clique $K_2$), this is referred to as an *independent transversal*.

It is well-known that a graph $G$ with $n$ vertices and average degree $d$ has an independent set of size at least $n/(d+1)$. For an independent transversal, a similar criterion exists. Alon gives a short LLL-based proof that a sufficient condition for such an independent transversal to exist is to require $b \geq 2e\Delta$ [**5**], where $\Delta$ is the maximum degree of any vertex in the graph. Haxell provides an elegant topological proof that a sufficient condition is $b \geq 2\Delta$ [**54**]. The condition of [**54**] is existentially optimal, in the sense that $b \geq 2\Delta - 1$ is not always admissible [**60, 116, 111**]. The work of [**59**] gives a similar criterion of $b \geq \Delta + \lfloor \Delta/r \rfloor$ for the existence of a transversal which induces no connected component of size $> r$. (Here $r = 1$ corresponds to independent transversals.) Finally, the work of [**75**] gives a criterion of $b \geq \Delta$ for the existence of a transversal omitting $K_3$; this is the optimal constant but the result is non-constructive.

These bounds are all given in terms of the *maximum degree* $\Delta$, which can be a crude statistic. The proof of [**54**] adds vertices one-by-one to partial transversals, which depends very heavily on bounding the maximum degree of any vertex. It is also highly non-constructive. Suppose we let $d$ denote the maximum *average* degree of any class $V_i$ (that is, we take the average of the degree (in $G$) of all vertices in

$V_i$, and then maximize this over all $i$). This is a more flexible statistic than $\Delta$. We present our first result here in parts (R1), (R2), and (R3) of Theorem 5.13. As shown in [**60, 116, 111**], the result of (R1) cannot be improved to $b \geq 2\Delta - 1$ (and hence, in particular, to $b \geq 2d-1$). As shown in [**75**], the result of (R3) cannot be improved to $b \geq cd$ for any constant $c < 1$. The result (R1) for independent transversals can also be obtained using the LLL variant of [**94**], but (R2) and (R3) appear new to our knowledge.

THEOREM 5.13. *Suppose we have a graph $G$ whose vertex set is partitioned into blocks of size at least $b$. Suppose that the average degree of the vertices in each block is at most $d$. Then:*

**(R1):** *If $b \geq 4d$, then $G$ has an independent transversal;*

**(R2):** *If $b \geq 2d$, then $G$ has a transversal which induces no connected component of size $> 2$;*

**(R3):** *If $b \geq (4/3)d$, then $G$ has a transversal which induces no 3-clique $K_3$.*

*Furthermore, these transversals can be constructed in expected polynomial time.*

PROOF. We define $l$ separate variables $X_1, \ldots, X_l$; variable $X_l$ selects which of the $b$ vertices in block $l$ go into the transversal. For each forbidden subgraph which appears in $G$, we associate an atomic bad event. (Note that the atomic bad events for (R2) are all the path of length two in $G$; for (R3) they are the triangles of $G$.)

We use the trivial labeling $K = 1$. Each forbidden subgraph corresponds to an atomic bad-event. We define a fractional hitting set $Q$ as follows. For each edge $f = \langle u, v \rangle \in E$ we assign weight $B'(\{u, v\}) = 1/r$, where $r$ is a parameter depending on the structure we are avoiding. For case (R1), we assign $r = 1$. For case (R2), we assign $r = 2$; for case (R3) we assign $r = 3$. ($B'$ is zero everywhere else.) Now, note that in any of the three cases, the atomic bad events all involve exactly $r$ edges, so the fractional hitting set is valid. Furthermore, any pair of such edges overlap in at least one vertex, so the $\bowtie$ relation is null in this case.

Note that the precondition of Theorem 5.12(c) holds here. We apply Theorem 5.12(c) with all entries of $\lambda$ being $\alpha$, for some scalar $\alpha$ to be determined. Let $d_v$ denote the degree of vertex $v$. Then, in order to prove $\lambda_i \geq 1 + \sum_k G_i(Q_k, \lambda)$, we need to show

$$b\alpha - \sum_{v \in V_i} d_v \alpha^2 / r \geq 1, \text{ i.e., } b\alpha - bd\alpha^2/r \geq 1 \text{ suffices.}$$

This has a solution $\alpha > 0$ iff $b \geq \frac{4d}{r}$, which gives us the three claimed results. □

**5.3.1. Avoiding large cliques.** For avoiding cliques of size $s > 3$, the above approach based on the maximum average degree $d$ no longer works; we instead give a bound in terms of the maximum degree $\Delta$. We will be interested in the case when both $s$ and $\Delta$ is large. That is, we will seek to show a bound of the form $b \geq \gamma_s \Delta + o(\Delta)$, where $\gamma_s$ is a term depending on $s$ and $s$ is large. Clearly we must have $\gamma_s \geq 1/(s-1)$; e.g., for the graph $G = K_s$, we need $b \geq 1 = \Delta/(s-1)$. An argument of [**111**] shows the slightly stronger lower bound $\gamma_s \geq \frac{s}{(s-1)^2}$; intriguingly, this is conjectured in [**111**] to be exactly tight. On the other hand, a construction in [**75**] shows that $\gamma_s \leq 2/(s-1)$. This is non-constructive, even for fixed $s$; this is the best upper-bound on $\gamma_s$ previously known.

We show that the lower-bound of [**111**] gives the correct *asymptotic* rate of growth, up to lower-order terms; i.e., we show in Theorem 5.14 that $\gamma_s \leq 1/s + o(1/s)$. In fact, we will show that when $b \geq \Delta/s + o(\Delta)$, we can find a transversal which avoids any $s$-star; that is, all vertices have degree $< s$. This implies that the transversal avoids $K_s$. Furthermore, such transversals can be found in polynomial time.

**Comparison with the standard LLL:** Before we give our construction based on the PRA, we discuss how one might approach this problem using the standard LLL, and why this approach falls short. As in [**5**], we make the natural random choice for a transversal: choose a vertex randomly and independently from each $V_i$. Suppose we define, for each $s$-clique $H$ of the graph, a separate bad event. Each bad event has

probability $(1/b)^s$. We calculate the dependency of an $s$-clique as follows: for each vertex $v \in H$, we choose another $v'$ in the block of $v$, and $v'$ may be involved in up to $\Delta^{s-1}/(s-1)!$ other $s$-cliques. This gives us the LLL criterion

$$e \times (1/b)^s \times sb\Delta^{s-1}/(s-1)! \leq 1$$

which gives us the criterion

$$b/\Delta \geq \left(\frac{es}{(s-1)!}\right)^{\frac{1}{s-1}} = e/s + o(1/s)$$

In implementing the PRA, it is simpler to enforce the stronger condition that the traversal omits $s$-stars. (That is, in the traversal $T$, no vertex may have induced degree $\geq s$). We will resample the central vertex as well as $r \leq s$ of the exterior vertices in an $s$-star, chosen uniformly. We thus obtain our result:

THEOREM 5.14. *There is a constant $c > 0$ such that, whenever*

$$b \geq \Delta/s + cs^{-3/2} \log s$$

*then there is a transversal which omits any $s$-stars. Furthermore, such a transversal can be found in polynomial time.*

PROOF. We will use Theorem 5.12(c), assigning the constant vector $\vec{\lambda} = \alpha$ where $\alpha$ is a constant to be chosen. We use the trivial labeling. We use the following fractional hitting set: for each $H$ consisting of a single vertex and $r$ neighbors, we assign weight $\binom{s}{r}^{-1}$; for all other sets we assign weight zero. In this case, $\bowtie$ is null: for any two $r$-stars $H, H'$ which both correspond to the same $s$-star, will overlap in their central vertex.

Then the condition of Theorem 5.12(c) becomes

$$b\alpha - b\left(\binom{\Delta}{r} + \Delta\binom{\Delta-1}{r-1}\right)\binom{s}{r}^{-1}\alpha^{r+1} \geq 1$$

Routine algebra shows that this is satisfied when $b \geq \Delta/s + cs^{-3/2}\log s$, for some sufficiently large constant $c$.

To implement a step of the PRA, one must search the graph for any $s$-star in the current candidate transversal; this can be done easily in polynomial time. □

We note that this result improves on [**75**] in three distinct ways: it gives a better asymptotic bound; it is fully constructive; it finds a transversal omitting not only $s$-cliques but also $s$-stars.

### 5.4. Sums of random variables, and column-sparse packing

Different types of bad events call for different hitting-sets, and the best choice may depend on "global" information about the variables it contains, in addition to local parameters. However, there is a natural and powerful option for upper-tail bad events of the form $\sum_\ell Z_\ell \geq k$, which is what we discuss next. In the discussion below, elements will often be referred to as $x, x_r$ etc.; note that an element is always some pair of the form $(i, j)$.

THEOREM 5.15. *Suppose we are given an assignment of non-negative real numbers $\lambda_{i,j}$ to each element $(i,j) \in \mathcal{X}$. Let $Y \subseteq \mathcal{X}$ be a set of elements. Define $\mu = \sum_t \lambda_{x_t}$, and for each variable $i$ let*

$$\mu_i = \sum_{j:(i,j)\in Y} \lambda_{i,j}$$

*Suppose that there is a bad event $\mathcal{B}$ defined by*

$$\mathcal{B} \equiv \sum_{x\in Y} Z_x \geq \mu(1+\delta),$$

*where $\delta > 0$ and $\mu(1+\delta)$ is an integer. Let $d \leq \mu(1+\delta)$ be a positive integer. Then, recalling Definition 5.10, there is a fractional hitting-set $Q$ with the property*

$$G(Q,\lambda) \leq \frac{\mu^d}{d!\binom{(1+\delta)\mu}{d}};\ G_i(Q,\lambda) \leq (\mu_i/\mu)\cdot d\cdot(1-(\mu_i/\mu))^{d-1}\cdot\frac{\mu^d}{d!\binom{(1+\delta)\mu}{d}}.$$

*Also, we refer to the parameter $d$ as the* width *of this hitting-set.*

PROOF. Assign the following fractional hitting-set: for each subset $Y' = \{x_{r_1}, \ldots, x_{r_d}\} \subseteq Y$ of cardinality $d$ in which all the elements $x_{r_1}, \ldots, x_{r_d}$ come from distinct variables, assign weight $Q(Y') = \frac{1}{\binom{(1+\delta)\mu}{d}}$. One can easily see that $Q$ is a valid fractional hitting-set.

We now have

$$\begin{aligned} G(Q,\lambda) &= \frac{\sum_{\substack{x_{r_1}<\cdots<x_{r_d} \\ \text{from distinct categories}}} \lambda_{r_1}\ldots\lambda_{r_d}}{\binom{(1+\delta)\mu}{d}} \\ &= \frac{\sum_{1\le i_1<\cdots<i_d\le n} \mu_{i_1}\cdots\mu_{i_d}}{\binom{(1+\delta)\mu}{d}} \\ &\le \frac{\binom{n}{d}(\mu/n)^d}{\binom{(1+\delta)\mu}{d}} \le \frac{\mu^d}{d!\binom{(1+\delta)\mu}{d}}. \end{aligned}$$

For $i \in [n]$, we have

$$\begin{aligned} G_i(Q,\lambda) &\le \frac{\sum_{i_1,\ldots,i_{d-1}\neq i} \mu_{i_1}\cdots\mu_{i_{d-1}}\mu_i}{\binom{(1+\delta)\mu}{d}} \\ &\le \frac{\mu_i\binom{n-1}{d-1}(\frac{\mu-\mu_i}{n-1})^{d-1}}{\binom{(1+\delta)\mu}{d}} \\ &\le \frac{\mu_i(\mu-\mu_i)^{d-1}}{(d-1)!\binom{(1+\delta)\mu}{d}} \\ &\le (\mu_i/\mu)d(1-(\mu_i/\mu))^{d-1}\frac{\mu^d}{d!\binom{(1+\delta)\mu}{d}}. \end{aligned}$$

□

Note that by setting $d = \lceil \mu\delta \rceil$, one can achieve the Chernoff bounds [**100**]:

$$\frac{\mu^d}{d!\binom{(1+\delta)\mu}{d}} \le \Big(\frac{e^\delta}{(1+\delta)^{1+\delta}}\Big)^\mu.$$

**5.4.1. LP rounding for column-sparse packing problems: background.** In light of Theorem 5.15, consider the family of CSPs where we have a series of linear packing constraints of the form "$\sum_x a_{k,x} Z_x \le b_k$", with non-negative coefficients $a, b$

and where $Z \in \{0,1\}^N$. (Here and in what follows, $x$ will often refer to some element $(i,j)$.) In addition, there is the usual assignment constraint, namely a series of disjoint blocks $\mathcal{X}_1, \ldots, \mathcal{X}_n$ with the constraint that $\sum_{j \in X_l} Z_{i,j} = 1$. When does such an integer linear program have a feasible solution?

Suppose we wish to solve this via LP relaxation. One technique is to solve the simpler linear program where the integrality constraints on $Z \in \{0,1\}^N$ are relaxed to $Z' \in [0,1]^N$, and in addition the packing constraints are tightened to $\sum_x a_{k,x} y_x \leq b'_k$ for some $b'_k \leq b_k$. We assume that each $a_{k,x} \in [0,1]$ and that for each $x \in \mathcal{X}$ we have $\sum_k a_{k,x} \leq D$.

We note that there are constructions using the standard LLL and standard MT algorithm that can yield results qualitatively similar to the ones in this section. The simplest case for these discrepancy results is when all the packing coefficients $b'_k$ have similar magnitudes; so suppose that for all $k$ we have $b_k \leq R$; we compare our result with those of [**52**] and [**72**] in this regime.

The analysis of [**52**] shows that one may select $b_k = R + O(\sqrt{R \log(RD)})$, but it applies only to situations in which the fractional solution is given by $Z'_{i,1} = Z'_{i,2} = 1/2$. This is useful for problems based on hypergraph discrepancy and its generalizations, and in the regime in which $R \ll D$ it can lead to stronger results than we are able to show. However, it does not apply to the general case, and in particular it is not useful if the fractional solution $Z'$ comes from some problem-specific LP relaxation. Our analysis is more general than [**51**]; in addition, in the regime $R \gg D$, we are able to show the stronger bound $b_k = R + O(\sqrt{R \log D})$.

The analysis of [**72**] is phrased in terms of the total *number* of constraints each variable participates in (as opposed to the $\ell_1$ sum of the corresponding coefficients). It is quite difficult for the standard LLL to deal with fractional coefficients in the constraint matrix; the reason is that variable $y_{i,j}$ affects constraint $k$ if $a_{k,i,j} > 0$, and it is possible that every variable affects every constraint.

The case in which $b'_k$ have differing magnitudes appears to be beyond the scope of either of these analyses. In addition, these papers are quite difficult technically; in [**52**], there is a quantization argument, in which one must handle separately coefficients of different magnitudes. In [**72**], there is an iterated application of the LLL, in which one must track quite carefully the sizes of the relevant parameters as they are reduced from the original system. The PRA provides however, a simple and comprehensive framework to obtain an integral solution. A single application of the PRA directly produces our desired solution, and fractional coefficients are handled almost automatically.

Our condition on the separation between $b_k$ and $b'_k$ is based on the Chernoff bound. To state this theorem in the simplest and broadest form, we will need the following form of the standard Chernoff upper-tail bound:

DEFINITION 5.16. ***(The Chernoff upper-tail separation function)*** *For $0 < \mu \leq t$, letting $\delta = \delta(\mu, t) = t/\mu - 1 \geq 0$, we define*

$$\textit{Chernoff-}U(\mu, t) = \Big(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\Big)^{\mu};$$

*i.e., Chernoff-$U(\mu, t)$ is the Chernoff bound that a sum of $[0,1]$-bounded and independent random variables with mean $\mu$ will exceed $t$. If $t < \mu$ we define Chernoff-$U(\mu, t) = 1$.*

We state two simple propositions about the behavior of this function:

PROPOSITION 5.17. *For any $t \geq \mu \geq 0$, and any real number $z \geq 0$, we have*

$$\textit{Chernoff-}U(\mu + z, t + z) \geq \textit{Chernoff-}U(\mu, t)$$

PROOF. We prove this in Proposition 6.31. □

PROPOSITION 5.18. *For any $v \geq 0$, and any real numbers $0 \leq z_1 \leq z_2$ we have*

$$\textit{Chernoff-}U(z_1, z_1 + v\sqrt{z_1}) \geq \textit{Chernoff-}U(z_2, z_2 + v\sqrt{z_2}) \geq e^{-v^2/2}$$

PROOF. Differentiating $\ln \text{Chernoff-U}(z, z + v\sqrt{z})$ with respect to $z$ yields

$$\frac{2v - (v + 2\sqrt{z})\log(1 + v/\sqrt{z})}{2\sqrt{z}}$$

Suppose $v \geq \sqrt{z}$. Then the numerator is at most $2v - (v + 2v)\log(1 + 1) \leq -0.7v \leq 0$.

Suppose $v \leq \sqrt{z}$. Then, taking a power series of the logarithm function around $v = 0$, the denominator is $\leq -v + v^3/(2z) - 2\sqrt{z}$. Simple analysis shows that for $v \leq \sqrt{z}$ we have $-v + v^3/(2z) \leq 0$.

In either case, the derivative is negative and so Chernoff-U$(z, z + v\sqrt{z})$ is a decreasing function of $z$. Simple calculus shows that as $v \to \infty$ it approaches the limit $e^{-v^2/2}$.

□

**5.4.2. LP rounding for column-sparse packing problems: the PRA.** We can now state our main theorem for column-sparse packing problems:

THEOREM 5.19. *There is some universal constant $C > 0$ with the following property. Suppose that we have an LP parametrized by $a_{k,x}, b'_k$, where $D = \max_x \sum_{k,x} a_{k,x} \geq 1$.*

*Now let $\epsilon \in [0, 1]$, $b_k, \phi \in [0, 1]$ be given such that:*

**(C1):** *For all $k$ we have $(\frac{b_k+1}{b'_k(1+\epsilon)} - 1)$ Chernoff-U$(b'_k(1 + \epsilon), b_k) \leq \frac{C\epsilon}{D}$*

**(C2):** *For all $k$ we have $b_k \geq 1$*

**(C3):** *For all $k$ we have Chernoff-U$(b'_k(1 + \epsilon), b_k) \leq 1/2$*

*Then if the linear program*

$$\sum_{j \in X_l} Z_{i,j} \geq 1, \qquad \sum_x a_{k,x} y_x \leq b'_k, \qquad y_x \in [0, 1]$$

*is satisfiable, then so is the **integer** program*

$$\sum_{j \in X_l} Z_{i,j} \geq 1, \qquad \sum_x a_{k,x} y_x \leq b_k, \qquad y_x \in \{0, 1\}.$$

*Furthermore, suppose we have a* separation oracle *for the LP and the IP. (That is, given a variable assignment, we can either find a violated linear constraint, or determine that all constraints are satisfied). Then such a satisfying assignment can be found in time which is polynomial in the number $n$ of blocks.*

PROOF. Using the separation oracle, solve the LP in polynomial time. Suppose we have a feasible solution $Z$ to the relaxed linear program. Then we set $\lambda = (1+\epsilon)Z$. We associate a distinct label to each packing constraint being violated. We will use the framework of Theorem 5.9, in which each packing constraint will correspond to its complex bad-event.

Let us analyze a constraint $k$. Define $\mu = b'_k(1+\epsilon)$, define $t = b_k$, and define $\delta = t/\mu - 1$. Then the $k$th packing constraint corresponds to the complex bad-event $\sum_x a_{k,x} y_x > b_k$. We use the fractional hitting set $Q_k$ which assigns, to each set of $d$ elements $x_1, \ldots, x_d$ from distinct categories, the weight

$$Q_k(\{x_1, \ldots, x_l\}) = \frac{a_{k,x_1} \ldots a_{k,x_l}}{\binom{b_k}{l}}$$

where $d = \lceil \mu\delta \rceil$.

We first claim that this is a valid hitting set. For, suppose we have a set of $d'$ elements such that $a_{k,x_1} + \cdots + a_{k,x_{d'}} > b_k$. As $a \in [0,1]$ we must have $d' > b_k \geq d$. Furthermore, summing over all $d$-subsets of $\{x_1, \ldots, x_{d'}\}$ we have

$$\sum_{s_1 < s_2 < \cdots < s_l} Q_k(\{x_{s_1}, \ldots, x_{s_d}\}) = \sum_{s_1 < s_2 < \cdots < s_d} \frac{a_{k,x_{s_1}} \ldots a_{k,x_{s_d}}}{\binom{b_k}{d}}$$

This can be regarded as a polynomial in the weights $a$. Subject to the constraint $a_{k,x_1} + \cdots + a_{k,x_d} \geq b_k$ and $a \in [0,1]$, the numerator achieves its minimum value when there are are $\lfloor b_k \rfloor$ elements of weight $a = 1$ and one further element of weight $a = b_k - \lfloor b_k \rfloor$. In particular, the numerator is at least $\binom{b_k}{d}$, and this sum is at least 1 as desired.

We will next bound the contribution of each bad event. For a constraint $k$, we have

$$\begin{aligned} G(Q_k, \lambda) &= \sum_{x_1 < \cdots < x_d} \lambda_{x_1} \ldots \lambda_{x_d} B'_k\{x_1, \ldots, x_d\} \\ &= \frac{\sum_{x_1 < \cdots < x_d} \lambda_{x_1} a_{k,x_1} \ldots \lambda_{x_d} a_{k,x_l}}{\binom{b_k}{d}} \end{aligned}$$

Now, note that $\sum_x \lambda_x a_{k,x} = \sum_t a_{k,x} y_{k,x}(1+\epsilon) \leq b'_k(1+\epsilon)$. By concavity, the numerator is maximized when all $|\mathcal{X}|$ terms $\lambda_t a_{k,t}$ are equal to $b'_k(1+\epsilon)/|\mathcal{X}|$. (Recall that $X$ is the set of all possible ordered pairs $(i,j)$.) Using a similar argument to Theorem 5.15, we have

$$G(Q_k, \lambda) \leq \frac{\binom{|\mathcal{X}|}{d}\left(\frac{b'_k(1+\epsilon)}{|\mathcal{X}|}\right)^d}{\binom{b_k}{l}} \leq \frac{\mu^d}{d!\binom{\mu(1+\delta)}{d}}$$

For $d = \lceil \mu\delta \rceil$, this expression is bounded by the Chernoff bound

$$\begin{aligned} G(Q_k, \lambda) &\leq \text{Chernoff-U}(b'_k(1+\epsilon), b_k) \\ &\leq 1/2 \quad \text{by (C3)} \end{aligned}$$

Similarly, along the lines of Theorem 5.15, for variable $i$, set $\mu_i = \sum_j a_{k,i,j}\lambda_{i,j}$ so that we have

$$\begin{aligned} G_i(Q_k, \lambda) &\leq \frac{\mu_i d}{\mu} \frac{\mu^d}{d!\binom{(1+\delta)\mu}{d}} \\ &\leq \mu_i(\delta + 1/\mu)\text{Chernoff-U}(\mu, t) \\ &\leq \mu_i\Big(\frac{b_k+1}{b'_k(1+\epsilon)} - 1\Big)\text{Chernoff-U}(\mu, t) \\ &= O(C\mu_i\frac{\epsilon}{D}) \quad \text{by (C1)} \end{aligned}$$

Now, by Theorem 5.12(b), we sum over all $j$ obtaining

$$\begin{aligned}
\sum_j \lambda_{i,j} - \sum_k \frac{G_i(Q_k,\lambda)}{1-G(Q_k,\lambda)} &\geq (1+\epsilon) - \sum_k \frac{\mu_{k,i} O(C\epsilon/D)}{1/2} \\
&= (1+\epsilon) - \sum_j \sum_k O(a_{k,i,j}\lambda_{i,j} C\frac{\epsilon}{D}) \\
&\geq (1+\epsilon) - C\epsilon O(\sum_j \lambda_{i,j}) \\
&\geq 1+\epsilon - O(C\epsilon(1+\epsilon)) \\
&\geq 1+\epsilon - O(C\epsilon) \qquad \text{as } \epsilon \in [0,1]
\end{aligned}$$

for $C$ sufficiently small, this is $\geq 1$ as desired.

Furthermore, we have $\sum_j \lambda_{i,j} \leq 1+\epsilon \leq 2$, so the expected number of iterations before the PRA terminates is $O(n)$. Although the number of constraints may be exponential, it is not hard to see that one can efficiently implement a single step of the PRA using the separation oracle. So this gives a polynomial-time algorithm.

□

Theorem 5.19 is given in a very generic setting, which is intended to handle a wide range of sizes for the parameters $b_k, b'_k, D$. One can obtain simplify it substantially when $b'_k$ is constant.

PROPOSITION 5.20. *Suppose we are given an LP satisfying the requirement of Theorem 5.19 and suppose that $b'_k \leq R$ for some $R \geq 1$.*

*Then, for $\psi$ some sufficiently large constant, setting*

$$b_k = \begin{cases} \psi \frac{\ln(D+1)}{1+\ln(\frac{\ln(D+1)}{R})} & \text{if } R \leq \ln(D+1) \\ R + \psi\sqrt{R(1+\ln D)} & \text{if } R > \ln(D+1) \end{cases}$$

*suffices to satisfy Theorem 5.19.*

PROOF. First, note that we may assume $b'_k = R$ exactly; for if $b'_k \leq R$, then we simply set $b'_k = R$, and this results in a relaxed linear program.

**Case I:** $R \geq \ln(D+1)$

Let $v = \sqrt{R\ln(D+1)}$ and set $\epsilon = v/R$. Note that $v \leq R$ and so $\epsilon \leq 1$. Then to show (C3):

$$\begin{aligned}
\text{Chernoff-U}(b'_k(1+\epsilon), b_k) &= \text{Chernoff-U}(R+v, R+\psi v) \\
&\leq \text{Chernoff-U}(2R, 2R+(\psi-1)v) \qquad \text{by Proposition 5.17} \\
&\leq \text{Chernoff-U}(2R, 2R+\frac{\psi-1}{\sqrt{2}}\sqrt{2R}) \\
&\leq \text{Chernoff-U}(2, 2+(\psi-1)/\sqrt{2}) \text{by Proposition 5.18} \\
&\leq 1/2 \quad \text{for } \psi \text{ sufficiently large}
\end{aligned}$$

Now for (C1), we have

$$\Big(\frac{b_k+1}{b'_k(1+\epsilon)} - 1\Big)\text{Chernoff-U}(b'_k(1+\epsilon), b_k) = O(\frac{\psi v}{R}\text{Chernoff-U}(R+v, R+\psi v))$$

We want to show that this is $\leq \frac{C\epsilon}{D} = \frac{Cv}{DR}$. Thus, it suffices to show that

$$D\psi\text{Chernoff-U}(R+v, R+\psi v) \leq C'$$

where $C'$ is a sufficiently small constant.

To show this:

$$\begin{aligned}
&D\psi\text{Chernoff-U}(R+v, R+\psi v) \\
&\quad = D\psi\text{Chernoff-U}(R+v, (R+v)+(\psi-1)v) \\
&\quad \leq D\psi\text{Chernoff-U}(2R, 2R+\frac{(\psi-1)\sqrt{\ln(D+1)}}{\sqrt{2}}\sqrt{2R}) \qquad \text{by Proposition 5.17} \\
&\quad \leq D\psi\text{Chernoff-U}(2\ln(D+1), 2\ln(D+1) \\
&\qquad + \frac{(\psi-1)\sqrt{\ln(D+1)}}{\sqrt{2}}\sqrt{2\ln(D+1)} \qquad \text{by Proposition 5.18, as } R \geq \ln(D+1) \\
&\quad \leq (D+1)\psi\text{Chernoff-U}(2\ln(D+1), 2\ln(D+1)) + (\psi-1)\ln(D+1))
\end{aligned}$$

$$= \psi(D+1)^{\psi+\psi\ln 2-(\psi+1)\ln(\psi+1)+\ln 2}$$

Simple analysis shows that this is $\leq C'$ for $D \geq 1$ and $\psi$ sufficiently large.

**Case II:** $R \leq \ln(D+1)$

Let $v = \frac{\ln(D+1)}{1+\ln(\frac{\ln(D+1)}{R})}$ and set $\epsilon = 1$. Note that $v \geq R$. Then we have Chernoff-U$(b'_k(1+\epsilon), b_k)$ = Chernoff-U$(2R, \psi v) \leq$ Chernoff-U$(2v, \psi v) \leq$ Chernoff-U$(2, \psi)$. For $\psi$ sufficiently large this is $\leq 1/2$, so (C3) is satisfied.

Now for (C1), we have

$$(\frac{b_k+1}{b'_k(1+\epsilon)} - 1)\text{Chernoff-U}(b'_k(1+\epsilon), b_k) = O(\psi v \text{Chernoff-U}(2R, \psi v))$$

We want to show that this is $\leq \frac{C\epsilon}{D} = \frac{C}{D}$. As $v \leq \ln(D+1) \leq D$, it suffices to show

$$\psi D^2 \text{Chernoff-U}(2R, \psi v) \leq C' \tag{21}$$

To show this, we have:

$$\begin{aligned}
\text{Chernoff-U}(2R, \psi v) &= (2e)^{\psi v} e^{-2R} (\psi v/R)^{-\psi v} \\
&\leq (2e)^{\psi v} (\psi v/R)^{-\psi v} \\
&\leq (ev/R)^{-\psi v} \quad \text{for } \psi \text{ sufficiently large} \\
&= \exp\Big[\psi \ln(D+1) \left( -1 + \frac{\ln\big(1 + \ln \frac{\ln(D+1)}{R}\big)}{1 + \ln \frac{\ln(D+1)}{R}} \right)\Big]
\end{aligned}$$

Now, differentiating this expression with respect to $R$, one sees that it achieves a maximum value at $R = e^{1-e}\ln(D+1)$. Thus, plugging this value in, we have the bound

$$D^2 \psi \text{Chernoff-U}(2T, \psi v) \leq \psi(D+1)^{2-0.632\psi}$$

Simple analysis shows that this is $\leq C'$ for $D \geq 1$ and $\psi$ sufficiently large.

□

The multi-dimensional scheduling application from the introduction follows as an easy application of Proposition 5.20. First, given $(T_1, T_2, \ldots, T_D)$, we can, motivated by (12), set $x_{i,j} := 0$ if there exists some $\ell$ for which $p_{i,j,\ell} > T_\ell$. After this filtering, we solve the LP relaxation. If it gives a feasible solution, we scale the LP so that all the $b'_k$ values equal 1; our filtering ensures that the coefficient matrix has entries in $[0, 1]$ now, as required. By Proposition 5.20, we can now set $b_k = O\Big(\frac{\log(D+1)}{1+\log\log(D+1)}\Big)$. This result is not new to this thesis, but it is obtained in a particularly straightforward way.

Theorem 5.19 can also apply when the RHS have different magnitudes.

PROPOSITION 5.21. *Suppose we are given an LP satisfying the requirement of Theorem 5.19; let $c > 0$ be any desired constant. Then we can set $b_k$ as follows so as to satisfy Theorem 5.19:*

(1) *For each $k$ with $b'_k \leq \ln(D+1)$, we set $b_k = O\Big(\frac{\log(D+1)}{1+\log\frac{\log(D+1)}{b'_k}}\Big)$;*
(2) *For each $k$ with $b'_k \geq \ln(D+1))$, there is some constant $c' > 0$ such that we may set $b_k = b'_k(1 + D^{-c}) + c'\sqrt{b'_k \log(D+1)}$.*

PROOF. Set $\epsilon = D^{-c}$. The remainder of the proof is similar to Proposition 5.20. □

## 5.5. Packet routing

**5.5.1. Review of background and known approaches.** We begin by reviewing the basic strategy of [**73**], and its improvements by [**99**] and [**93**]. We recommend consulting [**99**], which is a very readable presentation of this problem as well as many more details and variants than we cover here. We note that [**93**] studied a more general version of the packet-routing problem, so their choice of parameters was not (and could not be) optimized.

We are given a graph $G$ with $N$ packets. Each packet has a simple path, of length at most $D$, to reach its endpoint vertex (we refer to $D$ as the *dilation*). In any timestep, a packet may wait at its current position, or move along the next edge on

its path. Our goal is to find a schedule of smallest makespan in which, in any given timestep, an edge carries at most a single packet.

We define the *congestion* $C$ to be the maximum, over all edges, of the number of packets scheduled to traverse that edge. It is clear that $D$ and $C$ are both lower bounds for the makespan, and [**73**] has shown that in fact a schedule of makespan $O(C+D)$ is possible. [**99**] provided an explicit constant bound of $39(C+D)$, as well as describing an algorithm to find such a schedule. This was improved to $23.4(C+D)$ in [**93**] as will be described below.

While the final schedule only allows one packet to cross an edge at a time, we will relax this constraint during our construction. We consider "infeasible" schedules, in which arbitrarily many packets pass through each edge at each timestep. We define an *interval* to be a consecutive set of times in our schedule, and the *congestion* of an edge in a given interval to be the number of packets crossing that edge. If we are referring to intervals of length $i$, then we define a *frame* to be an interval which starts at an integer multiple of $i$.

From our original graph, one can easily form an (infeasible) schedule with delay $D$ and overall congestion $C$. Initially, this congestion may "bunch up" in time, that is, certain edges may have very high congestion in some timesteps and very low congestion in others. So the congestion is not bounded on any smaller interval than the trivial interval of length $D$. During our construction, we will "even out" the schedule, bounding the congestion on successively smaller intervals.

Ideally, one would eventually finish by showing that on each each individual timestep (i.e. interval of length 1), the congestion is roughly $C/D$. In this case, one could turn such an infeasible schedule into a feasible schedule, by simply expanding each timestep into $C/D$ separate timesteps.

As [**93**] showed, it suffices to control the congestion on intervals of length 2. Given our infeasible schedule, we can view each interval of length 2 as defining a new subproblem. In this subproblem, our packets start at a given vertex and have

paths of length 2. The congestion of this subproblem is exactly the congestion of the schedule. Hence, if we can schedule problems of length 2, then we can also schedule the 2-intervals of our expanded schedule.

We quote the following result from [**93**].

PROPOSITION 5.22 ([**93**]). *Suppose there is an instance with delay $D = 2$ and congestion $C$. Then there is a schedule of makespan $C + 1$, which can be found in polynomial time.*

This was used by [**93**] to improve the bound on the overall makespan to $23.4(C + D)$. [**93**] speculated that by examining the scheduling for longer, but still small, delays, one could further improve the general packet routing. Unfortunately, we are not able to show a general result for small delays such as $D = 3$. However, as we will see, the schedules that are produced in the larger construction of [**99**] are far from generic, but instead have relatively balanced congestion across time. We will see how to take advantage of this balanced structure to improve the scheduling.

We begin by first reviewing the general construction of [**99**].

**5.5.2. Using the LLL to find a schedule.** The general strategy for this construction is to add random delays to each packet, and then allowing the packet to move through each of its edges in turn without hesitation. This effectively homogenizes the congestion across time. We have the following lemma:

LEMMA 5.23. *Let $i' < i$, let $m, C'$ be non-negative integers. Suppose there is a schedule $S$ of length $L$ such that every interval of length $i$ has congestion at most $C$. Suppose that we have*

$$e \times P(Binomial(C, \frac{i'}{i - i'}) > C') \times (Cmi^2 + 1) < 1$$

*Then there is a schedule $S'$ of length $L' = L(1 + 1/m) + i$, in which every interval of length $i'$ has congestion $\leq C'$. Furthermore, this schedule can be constructed in expected polynomial time.*

PROOF. We break the schedule $S$ into frames of length $F = mi$, and refine each separately. Within each frame, we add a random delay of length $i - i'$ to each packet separately.

Let us fix an $F$-frame for the moment. Associate a bad event to each edge $f$ and $i'$-interval $I$, that the congestion in that interval and edge exceeds $C'$.

For each $I, e$, there are at most $C$ possible packets that could cross, and each does so with probability $p = \frac{i'}{i-i'}$. Hence the probability of the bad event is at most the probability that a Binomial random variable with $C$ trials and probability $p$ exceeds $C'$.

Next consider the dependency. Given an edge $f$ and $i'$-interval $I$, there are at most $C$ packets crossing it, each of which may pass through up to $mi$ other edges in the frame. We refer to the combination of a specific packet passing through a specific edge as a *transit*. Now, for each transit, there are (depending on the delay assigned to that packet) at most $i$ other $i'$-intervals in which this transit could have been scheduled. Hence the dependency is at most $Cmi^2$.

By the LLL, the condition in the hypothesis guarantees that there is a positive probability that the delays avoid all bad events. In this case, we refine each frame of $S$ to obtain a new schedule $S'$ as desired. We can use the algorithmic LLL to actually find such schedules $S'$ in polynomial time.

So far, this ensures that *within each frame*, the congestion within any interval of length $i'$ is at most $C'$. In the refined schedule $S'$ there may be intervals that cross frames. To ensure that these do not pose any problems, we insert a delay of length $i'$ between successive frames, during which no packets move at all. This step means that the schedule $S'$ may have length up to $L(1 + 1/m) + i$. □

Using this Lemma 5.23, we can transform the original problem instance (in which $C, D$ may be unbounded), into one in which $C, D$ are small finite values. In order to carry out this analysis properly, one would need to develop a series of separate bounds depending on the sizes of $C, D$. To simplify the exposition, we will assume that $C, D$

are very large, in which case certain rounding effects can be disregarded. When $C, D$ are smaller, we can show stronger bounds but doing this completely requires extensive case analysis of the parameters.

LEMMA 5.24. *Assume $C + D \geq 2^{896}$. There is a schedule of length at most $1.004(C + D)$ and in which the congestion on any interval of length $2^{24}$ is at most 17040600. Furthermore, this schedule can be produced in polynomial time.*

PROOF. Define the sequence $a_k$ recursively as follows.

$$a_0 = 256 \qquad a_{k+1} = 2^{a_k}$$

There is a unique $k$ such that $a_k^{3.5} \leq (C + D) < a_{k+1}^{3.5}$. By a slight variant on Lemma 5.23, one can add delays to obtain a schedule of length $C + D$, in which the congestion on any interval of length $i' = a_k^3$ is at most $C' = i'(1 + 4/a_k)$.

At this point, we use Lemma 5.23 repeatedly to ensure to control the congestion intervals of length $a_j^3$, for $j = k - 1, \ldots 0$. At each step, this increases the length of the resulting schedule from $L_j$ to $L_j(1 + 1/a_{j+1}) + a_j$, and increases the congestion on the relevant interval from $i(1 + 4/a_k)$ to

$$i(1 + 4/a_k) \prod_{j=0}^{k-1} (1 + 4/a_j)(\frac{1}{1 - (a_j/a_{j+1})^3})$$

(We use the Chernoff bound to estimate the binomial tail in Lemma 5.23.)

For $C + D \geq a_k^{3.5}$, it is a simple calculation to see that the increase in length is from $C + D$ (after the original refinement) to at most $1.004(C + D)$. In the final step of this analysis, we are bounding the congestion of intervals of length $a_0^3 = 2^{24}$, and the congestion on such an interval is at most 17040600.

Furthermore, since all of these steps use the LLL, one can form all such schedules in polynomial time.

See [**99**] for a much more thorough explanation of this process. □

Now that we have reduced to constant-sized intervals, we are no longer interested in asymptotic arguments, and come down to specific numbers.

LEMMA 5.25. *There is a feasible schedule of length at most* $10.92(C + D)$*, which can be constructed in polynomial time.*

PROOF. For simplicity, we assume $C + D \geq 2^{896}$.

By Lemma 5.24, we form a schedule $S_1$, of length $L_1 \leq 1.004(C + D)$, in which each interval of length $2^{24}$ has congestion at most 17040600.

Now apply Lemma 5.23 to $S_1$, with $m = 64, i' = 1024, C' = 1385$ to obtain a schedule $S_2$, of length $L_2 \leq 1.0157L_1 + 2^{24}$, in which each interval of length 1024 has congestion at most 1385.

Now apply Lemma 5.23 to $S_2$ with $m = 64, i' = 2, C' = 20$, to obtain a schedule $S_3$ of length $L_3 \leq 1.0157L_2 + 1024$, in which each frame of length 2 has congestion at most 20.

Now apply Proposition 5.22 to $S_3$, expanding each 2-frame to a *feasible* schedule of length 21. The total length of the resulting schedule is at most $\frac{21}{2}L_3 \leq 10.92(C + D)$. □

**5.5.3. The PRA applied to packet routing.** So far, all of the improvements we have made to the packet routing problem used nothing more than the conventional LLL. We now show how to modify this construction to use the PRA in the appropriate places.

Let us examine more closely the process used to refine a schedule in which each interval of length $C$ has congestion at most $i$. We break the schedule $S$ into frames of length $F = mi$, and refine each separately. Within each frame, we add a random delay of length $b = i - i'$ to each packet separately. Let us fix an $F$-frame for the moment.

This is an assignment problem, in which we must assign a delay to each packet. Our bad events here correspond to an edge receiving an excessive congestion in some time interval.

We can modify Lemma 5.24 and 5.25 to use Theorem 5.12 instead of the LLL.

PROPOSITION 5.26. *Let $i' < i$, let $m, C', k$ be non-negative integers. Suppose there is a schedule $S$ of length $L$ such that every interval of length $i$ has congestion at most $C$.*

*For a given choice of $d \leq C'$, $\lambda \in [0,1]$ define*

$$\mu = Ci'\lambda$$

*and*

$$p = \frac{\mu^d}{d!\binom{C'+1}{d}}$$

*Suppose we have $p < 1$ and*

$$(i - i')\lambda - mi^2 i'\lambda(d/\mu)\frac{p}{1-p} \geq 1$$

*Then there is a schedule $S'$ of length $L' = L(1 + 1/m) + i$, in which every interval of length $i'$ has congestion $\leq C'$. Furthermore, such a schedule can be found in polynomial time.*

PROOF. Suppose we add delays in the range $b = i - i'$ uniformly to each packet within each frame of length $F = mi$. In this case, the categories correspond to each packet $x$, and for each delay $t$ we assign $\lambda_{x,t} = \lambda$. For each edge $f$ and $i'$-interval $I$, we introduce a bad event $B_{f,I}$ that the congestion in the interval exceeds $C'$. For this bad event we use a fractional hitting-set of width $d$ as described in Theorem 5.15.

Fix a bad event $B_{f,I}$. This is an upper-tail event. There are at most $C$ packets which could be scheduled to pass through the given edge, and there are $i'$ possible delays which would contribute to the congestion of the given edge-interval. So, in all,

the mean number of packets passing through the edge-interval is $\mu = Ci'\lambda$. The bad event is that this exceeds $C'$, so we have here $\delta = \frac{C'+1}{Ci'\lambda} - 1$. This gives $G^{f,I} \leq \frac{\mu^d}{d!\binom{C'+1}{d}}$

Now consider a fixed packet $x$. This packet $x$ may pass through up to $mi$ edges in the $F$-frame, and each of these can affect at most $i$ intervals. Hence the packet $x$ affects at most $mi^2$ of these bad events. For each such bad event $f, I$, there are at most $i'$ possible delays that could be assigned to the given packet $x$ to contribute to congestion of $f, I$. Hence, for each bad event $B_{f,I}$, we have $\mu_x = i'\lambda$ and hence $G_x^{f,I} \leq i'\lambda d/\mu(\frac{\mu^d}{d!\binom{C'+1}{d}})$.

By Theorem 5.12, this suffices to show a good configuration exists. □

We can use this to improve various steps in the construction.

PROPOSITION 5.27. *Suppose $C + D \geq 2^{896}$. Then there is a schedule of length $\leq 1.0157(C+D)$, in which every interval of length 1024 has congestion at most 1312.*

PROOF. By Lemma 5.24, we form a schedule $S_1$, of length $L_1 \leq 1.004(C+D)$, in which each interval of length $2^{24}$ has congestion at most 17040600. Apply Proposition 5.26 with $\lambda = 5.985 \times 10^{-8}, C' = 1312, d = 247, m = 64$ to obtain a schedule $S_2$ of length $L_2 \leq 1.0157L_1 + 2^{24}$, in which each interval of length 1024 has congestion at most 1312. □

THEOREM 5.28. *Suppose $C+D \geq 2^{896}$. Then there is a schedule of length at most $8.77(C + D)$ which can be constructed in polynomial time.*

PROOF. By Proposition 5.27, there is a schedule $S_1$ of length at most $1.0157(C + D)$ in which each interval of length 1024 has congestion at most 1312.

Now apply Lemma 5.24 with $i = 1024, C = 1312, i' = 2, m = 64, C' = 16, d = 12, \lambda = 0.0011306$ to obtain a schedule $S_2$ of length $L_2 \leq 1.0157L_1 + 1024$, in which each interval of length 2 has congestion at most 16.

Now apply Proposition 5.22 to $S_2$, expanding each 2-frame to a *feasible* schedule of length 17. The total length of the resulting schedule is at most $\frac{17}{2}L_2 \leq 8.77(C + D)$. □

**5.5.4. Better scheduling of the final 2-frame.** Let us examine the last stage in the construction more closely. In this phase, we are dividing the schedule into intervals of length 2, and we want to control the congestion of each edge in each 2-frame.

For a given edge $f$ and time $t$, we let $c_t(f)$ denote the number of packets scheduled to cross that edge in the four time steps of the original (infeasible) schedule.

Suppose we have two consecutive 2-frames starting at time $t$. The reason for the high value of $C'$ in the final step of the above construction is that it is quite likely that $c_t + c_{t+1}$ or $c_{t+2} + c_{t+3}$ are much larger than their mean. However, it would be quite rare for *both* these bad events to happen simultaneously. We will construct a schedule in which we insert an "overflow" time between the 2-frames. This overflow handles cases in which either $c_t + c_{t+1}$ is too large *or* $c_{t+2} + c_{t+3}$ is too large.

Our goal will be to modify either of the 2-frames so as to ensure that the congestion is at most $T$. In order to describe our modification strategy, we first fix, for every packet and frame, a "first edge" and "second edge" in this frame. Some packets may only transit a single edge, which we will arbitrarily label as first or second. As we modify the schedule, some packets that initially had two transits scheduled will be left with only one; in this case, we retain the label for that edge. So, we may assume that every edge is marked as first or second and this label does not change.

We do this by shifting transits into the overflow time. For each 2-frame, there are two overflow times, respectively earlier and later. If we want to shift an edge to the later overflow time, we choose any packet that uses that edge as a second edge (if any), and reschedule the second transit to the later overflow time; similarly if we shift an edge to the earlier overflow time. See Figure 5.5.4.

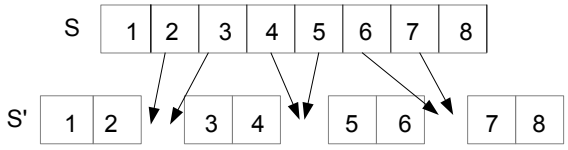


FIGURE 1. The packets in the original schedule $S$ are shifted into overflow times in the schedule $S'$.

Note that in the analysis of [**93**], the only thing that matters is the *total* congestion of an edge in each 2-frame. In deciding how to shift packets into the overflow times, we need to be careful to account for how often the edge appears as the first or second transit. If an edge appears exclusively as a "first edge", we will only be able to shift it into the earlier overflow, and similarly if an edge appears exclusively as a "second edge".

Keeping this constraint in mind, our goal is to equalize as far as possible the distribution of edges into earlier and later overflows. We do this by the following scheme:

1. For each edge $f$ and every odd integer $t = 1, 3, 5, \ldots, L$, repeat while $c_t(f) + c_{t+1}(f) > T$:
   2. If $c_t(f) = 0, c_{t+1}(f) > T$, then shift one packet into the later overflow time.
   3. Else if $c_t(f) > T, c_{t+1}(f) = 0$, then shift one packet into the earlier overflow time.
   4 Else if $c_t(f) + c_{t+1}(f) > T, c_t(f) > 0, c_{t+1}(f) > 0, c_t(f) + c_{t+1}(f) = \text{odd}$, then shift one packet into the earlier overflow time.
   5. Else if $c_t(f) + c_{t+1}(f) > T, c_t(f) > 0, c_{t+1}(f) > 0, c_t(f) + c_{t+1}(f) = \text{even}$, then shift one packet into the later overflow time.

Suppose we fix $t$ to be some odd integer. If we let $c'$ denote the congestions at the end of this overflow-shifting process, then we have $c'_t(f) + c'_{t+1}(f) \leq T$, and the number of packets shifted into the earlier (respectively later) overflow time can be viewed as a function of the original values of the congestions $c_t, c_{t+1}$. We denote these "overflow" functions by $\text{OF}^-(c_t, c_{t+1}; T)$ and $\text{OF}^+(c_t, c_{t+1}; T)$ respectively.

Specifically we get the following condition:

PROPOSITION 5.29. *Suppose that we have a schedule of even length $L$, and let $c_t(f)$ for $t = 1, \ldots, L$ denote the number of times $f$ is scheduled as the $t^{th}$ edge of a packet. Suppose that for all edges $f \in E$ and all $t = 1, 3, 5, \ldots$ we satisfy the constraint*

$$OF^{+}(c_t(f), c_{t+1}(f); T) + OF^{-}(c_{t+2}(f), c_{t+3}(f); T) \leq T'$$

*as well as the boundary constraints*

$$OF^{-}(c_1(f), c_2(f); T) \leq T' \qquad OF^{+}(c_{L-1}(f), c_L(f); T) \leq T'$$

*Then there is a schedule of makespan $L \times \frac{T+T'+2}{2} + T'$, which can be constructed in polynomial time.*

PROOF. After the modification, each 2-frame has congestion at most $T$, while each overflow time has congestion at $T'$. Each overflow time has delay at most 2, since for any packet $x$, there may be at most two edges scheduled into that overflow time, namely the edge that had been originally marked as the second edge of the earlier 2-frame, and the edge that had been originally marked as the first edge of the latter 2-frame. Hence each 2-frame can be scheduled in time $T + 1$ and each overflow can be scheduled in time $T' + 1$. As there are $L/2$ 2-frames in the original schedule, there are $L/2 + 1$ overflow periods. Hence the total cost is at most $L\frac{T+T'+2}{2} + T'$. □

Note that the conditions required by this Proposition 5.29 are local, in the sense that any violation is any event which affects an individual edge and a 4-interval which starts at an odd time $t$. We refer to such an interval for simplicity as an *aligned 4-interval*. We refer to the conditions required by this Proposition as the *4-conditions*; these conditions can be viewed as either pertaining to an entire schedule, or to an individual aligned 4-interval. We also note that the 4-conditions are *monotone*, in the sense that if a configuration violates them, then it will continue to do so if the congestion of any edge at any time is increased.

We will show how to use the PRA to find a schedule satisfying the conditions of Proposition 5.29.

PROPOSITION 5.30. *Let $m = 64, C = 1312, i = 1024, T = 5, T' = 4$ be given. Suppose there is a schedule $S$ of length $L$ such that every interval of length $i$ has congestion at most $C$. There is a schedule of length $L' \leq (1+1/m)L+i$, which satisfies the 4-conditions with respect to $T, T'$. This schedule can be produced in polynomial time.*

PROOF. For each edge $f$, and each aligned 4-interval $I$ starting at time $t$, we introduce a complex bad event $\mathcal{B}_{f,I}$ that

$$\mathrm{OF}^{+}(c_t, c_{t+1}; T) + \mathrm{OF}^{-}(c_{t+2}, c_{t+3}; T) > T'$$

For this edge $f, I$, and any packet $x$ with delay $t$, we say that $\langle x, t\rangle$ has *type* $j$, if that packet-delay assignment would cause the given packet $x$ to land at position $t + j$ within the bad event, for $j = 0, ..3$. If that assignment $x, t$ does not contribute to $\mathcal{B}_{f,I}$, then $\langle x, t\rangle$ has no type. For each bad event, there are at most $C$ variables of each type.

For a bad event $\mathcal{B}_{f,I}$ and a fractional hitting-set $Q$, we define the score $s_j(f, I)$, for $j = 0, 1, 2, 3$ to be the maximum over all delays $x, t$ of type $j$, of the quantity

$$\sum_{Y \subseteq B \in \mathcal{B}, \langle x,t\rangle \in Y} Q(Y)\lambda^Y$$

Similarly, we define the overall-score to be $s(f, I) = g^{f,I} = \sum_{Y \subseteq B} B'(Y)\lambda^Y$. For a collection of all bad events $\mathcal{B}_{f,I}$, we define the score $s_j$ to be the maximum $s_j(f, I)$ over all $f, I$.

By Theorem 5.12, if we satisfy the condition

$$(i-4)\lambda - \frac{mi}{2}\frac{s_0 + s_1 + s_2 + s_3}{1-s} \geq 1$$

then there is a schedule of length $L' \leq (1+1/m)L+i$, which satisfies the 4-conditions with $T, T'$.

Hence it suffices to produce a series of hitting-sets, for each of the bad events $\mathcal{B}_{f,I}$, with a sufficiently small score.

Now let us fix a bad event $f, I$, and suppose that we have fixed $\lambda = 1.23 \times 10^{-3}$. We will describe how to produce a good hitting-set. Although we have stated the proposition for a particular choice of parameters, we will walk through the algorithm we use to construct and find such a hitting-set.

The bad event depends solely on the number of assigned variables of each type, of which there are at most $C$. To simplify the notation, we suppose there are exactly $C$. Our hitting-set assigns weights to any subset of the $4C$ variables involved in the bad event. We will also select a symmetric hitting-set, in the sense that the weight assigned to any $Y \subseteq [C] \times [4]$ depends solely on the number of variables of each type in $Y$. So, for any $y_0, y_1, y_2, y_3 \leq C$, we will assign $B'(Y) = b(y_0, y_1, y_2, y_3)$ for any $Y \subseteq [C] \times [4]$ which has $|Y \cap [C] \times \{j\}| = y_j$, that is, for any subset $Y$ which has exactly $y_j$ variables of each type. In this case, we will have

$$s_0 = \sum_{y_0,y_1,y_2,y_3} \binom{C-1}{y_0-1}\binom{C}{y_1}\binom{C}{y_2}\binom{C}{y_3} b(y_0, y_1, y_2, y_3)\lambda^{y_0+y_1+y_2+y_3}$$

and similarly for $s_1, s_2, s_3, s$.

In order to be valid, we must have $\sum_{Y \subseteq A} Q(Y) \geq 1$ for any atomic bad event $A \in \mathcal{B}_{f,I}$. By symmetry, this means that if we have $k_0, k_1, k_2, k_3$ minimal such that

$$\mathrm{OF}^+(k_0, k_1; T) + \mathrm{OF}^-(k_2, k_3; T) > T',$$

then we require

$$\sum_{y_0,y_1,y_2,y_3} \binom{k_0}{y_0}\binom{k_1}{y_1}\binom{k_2}{y_2}\binom{k_3}{y_3} b(y_0, y_1, y_2, y_3) \geq 1$$

We are trying to satisfy $(s_0+s_1+s_2+s_3)/(1-s) \leq t$, where $t$ is some target value; here $t = 2.803\times10^{-6}$. For a fixed value of $t$, this is equivalently to minimizing $s_0+s_1+s_2+s_3+ts$. If we view the collection of all values $b(y_0, y_1, y_2, y_3)$ as linear unknowns, then we can view both the objective function and the constraints as linear. Hence this defines a linear program, which we can solve using standard linear programming algorithms.

For any $y_0, y_1, y_2, y_3$, we will set $b(y_0, y_1, y_2, y_3) = 0$ unless there is some such minimal bad $k_0, k_1, k_2, k_3 \geq y_0, y_1, y_2, y_3$. This greatly reduces the number of variables we need to consider, to something which is very large but tractable. For $T = 5, T' = 4$, for instance, the linear program has 5990 variables and 156 constraints. This is too large to write explicitly, but we wrote computer code which generates this system and solves it.

The resulting hitting-set achieves a bound of

$$\frac{s_0 + s_1 + s_2 + s_3}{1 - s} \leq 2.81 \times 10^{-6}$$

which satisfies the conditions of Theorem 5.12(b). We have listed this hitting-set in full in Appendix 5.A. Note that this hitting-set gives a compact witness for Proposition 5.30; using it, one could verify the proposition directly without any need to use the algorithm we have just described. □

We now apply this construction to replace the two final steps in the construction of Section 5.5.4.

THEOREM 5.31. *There is a feasible schedule of makespan at most* $5.70(C + D)$*, which can be constructed in expected polynomial time.*

PROOF. For simplicity, we assume $C + D \geq 2^{896}$. By Proposition 5.27, we obtain a schedule $S_1$ of length $L_1 \leq 1.0157(C + D)$, in which each interval of length 1024 has congestion at most 1312.

Apply Proposition 5.30. This gives a schedule $S_2$ of length $L_2 \leq 1.0158L_1 + 1024$ satisfying the 4-conditions with $T = 5, T' = 4$. By Proposition 5.29, this yields a schedule whose makespan is $5.5L_2 + 5 \leq 5.70(C + D)$. □

## 5.A. The fractional hitting-set for packet routing

The following table lists the fractional hitting-set used in Proposition 5.30. All sets which do not appear in this list have $b(y_1, y_2, y_3, y_4) = 0$.

| $y_0, y_1, y_2, y_3$ | $b(y_0, y_1, y_2, y_3)$ | $y_0, y_1, y_2, y_3$ | $b(y_0, y_1, y_2, y_3)$ | $y_0, y_1, y_2, y_3$ | $b(y_0, y_1, y_2, y_3)$ |
|---|---|---|---|---|---|
| 0 0 4 7 | 5.099206e-03 | 0 0 5 6 | 9.778912e-04 | 0 0 6 5 | 3.996599e-03 |
| 0 0 7 4 | 2.579365e-03 | 0 0 8 0 | 2.222222e-02 | 0 5 2 5 | 4.385269e-04 |
| 0 5 3 4 | 1.359760e-04 | 0 5 4 3 | 3.833186e-04 | 0 5 5 2 | 9.740495e-05 |
| 0 5 6 0 | 1.539957e-03 | 0 6 3 2 | 4.658320e-05 | 0 6 3 3 | 9.241610e-07 |
| 0 6 5 0 | 1.893794e-04 | 0 7 0 4 | 4.585073e-04 | 0 7 1 3 | 1.992969e-03 |
| 0 7 2 2 | 1.199898e-03 | 0 7 3 0 | 2.093047e-05 | 0 7 3 1 | 1.698355e-03 |
| 0 7 4 0 | 9.174968e-04 | 0 8 0 0 | 2.222222e-02 | 1 3 7 0 | 3.316187e-04 |
| 2 2 7 0 | 3.883201e-04 | 2 3 2 6 | 1.818870e-04 | 2 3 3 5 | 6.708826e-05 |
| 2 3 4 4 | 1.277478e-04 | 2 3 5 3 | 1.369389e-04 | 2 3 7 0 | 8.113415e-04 |
| 2 4 2 5 | 5.158908e-05 | 2 4 3 4 | 8.499152e-05 | 2 4 4 3 | 4.975736e-06 |
| 2 4 5 2 | 1.133274e-04 | 3 1 7 0 | 2.672980e-04 | 3 2 2 6 | 4.410280e-05 |
| 3 2 3 5 | 7.970540e-05 | 3 2 4 4 | 4.896451e-05 | 3 2 5 3 | 7.392384e-05 |
| 3 2 6 2 | 5.816254e-06 | 3 2 7 0 | 1.742220e-04 | 3 3 2 5 | 6.943956e-06 |
| 3 3 5 2 | 3.800255e-05 | 3 5 1 4 | 1.369777e-04 | 3 5 2 3 | 9.535644e-05 |
| 3 5 3 2 | 1.282475e-04 | 3 5 4 0 | 8.297563e-05 | 3 5 4 1 | 5.296674e-05 |
| 3 6 2 2 | 9.002473e-06 | 4 1 2 6 | 1.767087e-04 | 4 1 3 5 | 5.212549e-05 |
| 4 1 4 4 | 1.093016e-04 | 4 1 5 3 | 9.464293e-05 | 4 1 6 2 | 3.789534e-05 |
| 4 1 7 0 | 1.125084e-03 | 4 2 2 5 | 7.353027e-05 | 4 2 3 4 | 1.587323e-05 |
| 4 2 4 3 | 1.237762e-05 | 4 2 5 2 | 8.670195e-05 | 4 3 2 4 | 5.232234e-05 |
| 4 3 3 3 | 8.595063e-05 | 4 3 4 2 | 2.209769e-05 | 4 3 5 0 | 1.041270e-04 |
| 4 4 1 4 | 1.052873e-04 | 4 4 2 3 | 1.437882e-05 | 4 4 3 2 | 4.431716e-05 |
| 4 4 4 0 | 7.896302e-05 | 4 4 4 1 | 4.184396e-05 | 4 5 1 3 | 5.225282e-05 |
| 4 5 2 2 | 5.006599e-05 | 4 5 3 1 | 3.748908e-05 | 5 1 2 5 | 2.980766e-05 |
| 5 1 3 4 | 8.802145e-05 | 5 1 4 3 | 4.768986e-05 | 5 1 5 2 | 7.014272e-05 |
| 5 3 1 4 | 6.990688e-05 | 5 3 2 3 | 1.155525e-05 | 5 3 3 2 | 3.894932e-05 |
| 5 3 4 1 | 3.980330e-05 | 5 4 1 3 | 3.277361e-05 | 5 4 2 2 | 7.791839e-05 |
| 5 4 3 1 | 2.503639e-05 | 5 6 0 0 | 1.549509e-03 | 6 2 1 4 | 1.448414e-04 |
| 6 2 2 3 | 6.887231e-05 | 6 2 3 2 | 1.145665e-04 | 6 2 4 0 | 1.365900e-04 |
| 6 2 4 1 | 4.324878e-05 | 6 2 5 0 | 4.360905e-05 | 6 3 1 3 | 7.156338e-05 |
| 6 3 2 2 | 3.895600e-05 | 6 3 3 1 | 5.915442e-05 | 6 6 0 0 | 2.267574e-04 |
| 7 2 1 3 | 3.562315e-05 | 7 2 2 2 | 7.950298e-05 | 7 2 3 1 | 1.473170e-05 |
| 7 5 0 0 | 1.289683e-03 | 8 4 0 0 | 3.756614e-03 | | |

## 5.B. The Nostradamus Lemma

In the proof of Lemma 5.7, we use the following general "Nostradamus Lemma." To justify this somewhat cryptic name, consider the following analogy. In a dusty book of prophecy, one reads that "sometime in the future, you will meet a man named John Doe. The first time you meet such a man, he will flip a coin a hundred times, all of which come up heads. Also, you will eventually meet a woman Jane Doe; the first such woman whom you meet will also flip a hundred coins, all of which come up heads." Now, you do not know when these meetings will come or which will come first, if ever, but you can confidently say that the probability that this prophecy comes true is at most $2^{-200}$.

This probabilistic principle seems fairly intuitive, but we note that there are two ways it can go wrong. Suppose that we predict that John Doe will flip a hundred heads and also a red-haired man will flip a hundred heads. This probability could be just $2^{-100}$, because the two men may be the same person. Another possibility: suppose we predict that "sometime in the future you will meet a man named John Doe who flips a hundred heads." The probability of this event could be one, if we encounter an infinite series of men with the same name.

LEMMA 5.32 (Nostradamus Lemma). *Suppose one has a stochastic process indexed* $\langle X_t \mid t \in \mathbf{N} \rangle$. *Let* $S$ *denote the countably infinite set of possible histories for this process; for notational simplicity we suppose that the states* $X_t$ *themselves lie in* $S$ *(i.e., that the state also includes the history thus far). The process begins in state* $s_0 \in S$, *which represents the null history. There are* $\ell$ *Boolean functions* $A_i, B_i : S \to \{0,1\}$, *for some finite* $\ell$. *Suppose that, for all* $t \in \mathbf{N}$ *and all* $s \in S$, *we have that*

$$P(B_i(X_{t+1}) = 1 \mid X_t = s_t) \leq p_i.$$

*Now define the event* $E$ *that the following conditions are all satisfied:*

(1) *For each* $i \in [l]$, *there is exactly one time* $t_i$ *such that* $A_i(X_{t_i}) = 1$;

(2) *For all* $i \neq i'$ *we have* $t_i \neq t_{i'}$*;*

(3) *For all* $i \in [l]$ *we have* $B_i(X_{t_i+1}) = 1$.

*Then the event* $E$ *is measurable, and its probability is at most* $P(E) \leq p_1 p_2 \dots p_\ell$.

PROOF. Define the event $E_T$, which is that event $E$ holds and also that $t_1, \dots, t_\ell \leq T$. We will prove that $P(E_T) \leq p_1 \dots p_\ell$ by induction on $T$. For a given value of $T$, the induction will apply across all possible stochastic systems and all possible values of $\ell$.

First, suppose $T = 0$. If $\ell > 0$, then the event $E_T$ is impossible; if $\ell = 0$, then event $E_T$ is certain. Either way the inequality holds.

Next, suppose $T > 0$. Count how many values of $i$ are there such that $A_i(s_0) = 1$. If there is more than one such value, then event $E_T$ is impossible, and the statement holds.

Suppose first that for all $i$ we have $A_i(s_0) = 0$. Then the event $E_T$ is equivalent to the event $E'_{T-1}$, where $E'$ is an event similar to $E$ except that it is defined on the stochastic process which starts at state $X_1$. By induction hypothesis, the event $E'_{T-1}$ has probability at most $p_1 \dots p_\ell$ for any $X_1$. Integrating over $X_1$, we have the $E_T$ has at most this probability as well.

Finally, suppose that there is exactly one value of $i$ such that $A_i(s_0) = 1$; without loss of generality say it is $i = \ell$. Then the event $E_T$ is equivalent to the event that $B_\ell(X_1) = 1$ *and* that the event $E'_{T-1}$ occurs, where $E'$ is an event similar to $E$ except that it is defined on the stochastic process which starts at state $X_1$ and only includes Boolean functions $A_1, B_1, \dots, A_{\ell-1}, B_{\ell-1}$. The probability of $B_\ell(X_1) = 1$ is at most $p_\ell$. By induction hypothesis, for any such $X_1$, the probability of the event $E'$ is at most $p_1 \dots p_{\ell-1}$. Hence, integrating over all $X_1$, the probability of this event is at most $p_1 \dots p_\ell$.

So we have shown that $P(E_T) \leq p_1 \dots p_\ell$ for all $T \in \mathbf{N}$. Now note that $E_0 \subseteq E_1 \subseteq E_2 \subseteq \dots$ and $E = \cup_{T \in \mathbf{N}} E_T$. Each set $E_T$ is cylindrical (it is determined by

the first $T$ coordinates), hence measurable. This implies that $E$ is measurable as well with $P(E) \leq p_1 \dots p_\ell$. $\square$

## 5.C. A probabilistic LLL variant

The standard LLL has two faces: the efficient MT algorithm, which finds a valid configuration by resampling bad-events, and the original probabilistic formulation discussed in Chapter 1. In the latter formulation, one selects the variables according to the indicated distribution, without any resamplings. One then has a positive, exponentially small, probability of avoiding all bad-events. This typically gives an existence proof without a corresponding polynomial-time algorithm.

These two interpretations of the LLL lend themselves to different generalizations and there are many useful interconnections between them. Historically, many improvements and specializations of the LLL were first developed in the probabilistic framework (including the original LLL itself), and then were translated into the Moser-Tardos framework. The Partial Resampling Algorithm we have given follows the opposite path: it is a generalization of the Moser-Tardos framework, without any probabilistic interpretation.

In this section, we will describe a probabilistic variant of the LLL which closely parallels the PRA. This process does not exactly match the bounds of the PRA; it is sometimes stronger but usually weaker. We will discuss the connections between the interpretations in Section 5.C.3.

We begin by setting and recalling some notations. We let $[n]$ index the variables, and let $Z_{i,j}$ be the indicator for selecting $X_i = j$. The set of bad-events $\mathcal{B}$ is, as before, a union of atomic bad-events. As before, we refer to an ordered pair $(i, j)$ as an *element*. Recall that we let $\mathcal{X}$ denote the set of all elements; we let $|\mathcal{X}| = N$. (This differs from the PRA, in which we never explicitly required that $\mathcal{X}$ was finite). We will *not* make use of a labeling function here.

**5.C.1. The assignment LLL.** Unlike in the usual LLL, we cannot simply define a probability distribution and compute the probability of the bad event occurring. We will only have partial control over this probability distribution, and the following definition will be important:

DEFINITION 5.33. ***(UNC)*** *Given a probability distribution $\Omega$ on the underlying indicator variables $Z$, we say that $\Omega$ satisfies* upper negative correlation with respect to probability vector $p$ *or simply "UNC(p)" if all entries of $p$ lie in $[0,1]$ and if for all elements $x_1, \ldots, x_k$, we have*

$$P_\Omega(Z_{x_1} = \cdots = Z_{x_k} = 1) \leq p_{x_1} \ldots p_{x_k}.$$

*For an event $E$ and a probability vector $p$, we define $P_p^*(E)$ to be the minimum, over all $\Omega$ satisfying UNC(p), of $P_\Omega(E)$. (As the number of variables is finite, this minimum is achieved.)*

Essentially, when computing $P^*(E)$, we are not allowing the random variables $Z$ to be positively correlated. For some types of events, such as large-deviation events, this allows us to control the probability very strongly; for other events, such as a union of many events, this is no better than the union bound.

Our main theorem is:

THEOREM 5.34 (Assignment LLL). *Suppose we are given a CSP for which there exists $\lambda \in [0,1]^N$ such that when we sample all the $Z_x$ independently with $\Pr[Z_x = 1] = \lambda_x$, we have*

$$\forall i \in [n],\ \sum_j \lambda_{i,j} \cdot P_\lambda^*[\bigcap_{B \in \mathcal{B}_{i,j}} \overline{B} \ \big|\ Z_{i,j} = 1] > 1.$$

*Then, if no bad-event $B \in \mathcal{B}$ is a tautology, the CSP is feasible.*

To prove the theorem, we will study the following probabilistic process. We are given a vector $p \in [0,1]^N$ of probabilities, one for each indicator $Z_x$. Each $Z_x$ is

drawn *independently* as Bernoulli-$p$, i.e. $P(Z_x = 1) = p_x$. (If for some event $x$ we have $p_x > 1$, then by an abuse of notation, we take this to mean that $Z_x = 1$ with certainty). Our goal is to satisfy all the assignment constraints and avoid all the events in $\mathcal{B}$. If $\mathcal{C} \subseteq \mathcal{B}$, we use the notation $\exists\mathcal{C}$ to denote the event that some $B \in \mathcal{C}$ occurs. So in this case, we want to *avoid* the event $\exists\mathcal{B}$. For an element $x$ we define $\mathcal{B}_x$ to index the set of all bad events $B \in \mathcal{B}$ which are (explicitly) affected by $Z_x$.

We recall a basic lemma concerning increasing and decreasing events, which follows from the FKG inequality [**42**]:

LEMMA 5.35. *Let $\mathcal{X}_0, \mathcal{X}_1 \subseteq \mathcal{X}$ be two disjoint subsets of the elements. Let $E_1$ be some event depending solely on variables in $\mathcal{X}_1$. Let $E^-$ be a decreasing event. Then,*

$$P(\forall x \in \mathcal{X}_0 \quad Z_x = 1 \mid \mathcal{E}_1, E^-) \leq p^{\mathcal{X}_0}$$

*Similarly, if $E^+$ is increasing, then $P(\forall x \in \mathcal{X}_0 \quad Z_x = 1 \mid E_1, E^+) \geq \prod_{x\in\mathcal{X}_0} p^{\mathcal{X}_0}$.*

*Recall that we are using the power notation so that $p^{\mathcal{X}_0}$ means simply $\prod_{x\in\mathcal{X}_0} p_x$.*

PROOF. We will only prove the first part of this lemma; the second is analogous.

We average over all assignments to the variables $Z_x$, for $x \in \mathcal{X}_1$. For any such assignment-vector $\vec{z}$, the event $\bigwedge_{x\in\mathcal{X}_0} Z_x = 1$ is an increasing function, while $E^-$ is a decreasing function in the remaining variables. Hence, by FKG, the probability of this event conditional on $(Z_{\mathcal{X}_1} = \vec{z} \wedge E^-)$ is at most its value conditional on $Z_{\mathcal{X}_1} = \vec{z}$ alone. But, the events $\bigwedge_{x\in\mathcal{X}_0} Z_{\mathcal{X}_0} = 1$ and $Z_{\mathcal{X}_1} = \vec{z}$ involve disjoint sets of variables, so they are independent. Hence this probability is at most the unconditional probability of $\bigwedge_{x\in\mathcal{X}_0} Z_x = \vec{1}$, namely $p^{\mathcal{X}_0}$. □

If $A \subseteq [n]$ is any subset of the variables, we define the event Assigned($A$) to be the event that, for all $i \in A$, there is *at least one* value of $j$ for which $Z_{i,j} = 1$. In the PRA, this is automatic because of the way we are defining the probability space. Here, however, we are thinking of the variables $Z_x$ as independent Bernoulli, and so the number of assignments to each variable may be zero or may be more than one.

Our goal is to satisfy the constraint Assigned($[n]$). If $i \in [n]$ we write Assigned($i$) as short-hand for Assigned($\{i\}$). Because all the bad events are atomic, if we can find a configuration in which each block has at least one value assigned, we can easily alter it to a feasible configuration in which each block has *exactly one* value assigned.

We are now ready to state the first lemma concerning this probabilistic process. We want to show that the there is a positive probability of satisfying all the assignment constraints and avoiding all bad events. We will show the following by induction, with stochastic domination playing a key role:

LEMMA 5.36. *Let $\epsilon < 1$. Suppose $p \in [0, \epsilon]^N$ is a probability vector such that for all blocks $i$,*

$$\sum_j p_{i,j}(P^*_{p/\epsilon}(\neg\exists \mathcal{B}_{i,j} \mid Z_{i,j} = 1)) - \sum_{j,j'} p_{i,j}p_{i,j'} \geq \epsilon.$$

*Then for any block $i$, any $\mathcal{C} \subseteq \mathcal{B}$ a set of bad events, and any $A \subseteq [n]$, we have*

$$P(\mathit{Assigned}(i) \mid \neg\exists B', \mathit{Assigned}(A)) \geq \epsilon.$$

PROOF. We show this by induction on $|\mathcal{C}| + |A|$. We may assume that $i \notin A$, as otherwise this is vacuous. First, suppose $|\mathcal{C}| = 0$. Then, $P(\text{Assigned}(i) \mid \neg\exists\mathcal{C}, \text{Assigned}(A))$ equals $P(\text{Assigned}(i))$ as these events are independent. By Inclusion-Exclusion, the latter probability is at least

$$P(\text{Assigned}(i)) \geq \sum_j p_{i,j} - \sum_{j,j'} p_{i,j}p_{i,j'}$$

and it is easy to see that the lemma's hypothesized constraint implies that this is at least $\epsilon$.

Next suppose $|\mathcal{C}| > 0$. We use Inclusion-Exclusion to estimate $P(\text{Assigned}(i) \mid \neg\exists\mathcal{C}, \text{Assigned}(A))$. First, consider the probability that a distinct pair $j, j'$ in block $i$ are jointly chosen, conditional on all these events. For this, by Lemma 5.35 we have

$$P(Z_{i,j} = Z_{i,j'} = 1 \mid \text{Assigned}(A), \neg\exists\mathcal{C}) \leq p_{i,j}p_{i,j'} \tag{22}$$

as $i \notin A$ and $\neg\exists\mathcal{C}$ is decreasing.

Let us fix $j$. We next need to show a lower bound on $P(Z_{i,j} = 1 \mid \text{Assigned}(A), \neg\exists\mathcal{C})$. This is easily seen to equal $P(Z_{i,j} = 1)$ if $\mathcal{B}_{i,j} \cap \mathcal{C} = \emptyset$, so we can assume $\mathcal{B}_{i,j} \cap \mathcal{C} \neq \emptyset$. We see by Bayes' Theorem that $P(Z_{i,j} = 1 \mid \text{Assigned}(A), \neg\exists\mathcal{C})$ equals

$$\frac{P(\neg\exists(\mathcal{C} \cap \mathcal{B}_{i,j}) \mid Z_{i,j} = 1, \text{Assigned}(A), \neg\exists(\mathcal{C} - \mathcal{B}_{i,j}))}{P(\neg\exists(\mathcal{C} \cap \mathcal{B}_{i,j}) \mid \text{Assigned}(A), \neg\exists(\mathcal{C} - \mathcal{B}_{i,j}))} \times P(Z_{i,j} = 1 \mid \text{Assigned}(A), \neg\exists(\mathcal{C} - \mathcal{B}_{i,j}))$$

since the denominator is a probability, Lemma 5.35 yields

$$P(Z_{i,j} = 1 \mid \text{Assigned}(A), \neg\exists\mathcal{C}) \geq p_{i,j} \cdot P(\neg\exists\mathcal{B}_{i,j} \mid Z_{i,j} = 1, \text{Assigned}(A), \neg\exists(\mathcal{C} - \mathcal{B}_{i,j})).$$

(This approach to handling a conditioning was inspired by [**19**].)

Consider the random variables $Z$ **conditioned on** the events $\text{Assigned}(A), \neg\exists(\mathcal{B} - \mathcal{B}_{i,j}), Z_{i,j} = 1$. Our *key claim* now is that these conditional random variables $Z$ (apart from $Z_{i,j}$ itself) satisfy UNC($p/\epsilon$): note that $p/\epsilon$ is a valid probability vector since $p \in [0, \epsilon]^N$. To show this, we need to upper-bound $P(\mathcal{E}_1 \mid \mathcal{E}_2)$, where

$$\mathcal{E}_1 \equiv (Z_{i'_1,j'_1} = \cdots = Z_{i'_k,j'_k} = 1) \text{ and}$$
$$\mathcal{E}_2 \equiv (\text{Assigned}(A), \neg\exists(Y - \mathcal{B}_{i,j}), Z_{i,j} = 1),$$

and where $k$ and $i'_1, j'_1, \ldots, i'_k, j'_k$ are arbitrary. Letting $I' = \{i'_1, \ldots, i'_k\}$, we also define

$$\mathcal{E}_3 \equiv (Z_{i,j} = 1, \text{Assigned}(A - I'), \neg\exists(\mathcal{B} - \mathcal{B}_{i,j})).$$

By simple manipulations, we see that $P(\mathcal{E}_1 \mid \mathcal{E}_2)$ is at most

$$P(\mathcal{E}_1 \mid \mathcal{E}_3)/P(\text{Assigned}(i'_1, \ldots, i'_k) \mid \mathcal{E}_3). \tag{23}$$

Note that $\mathcal{E}_1$ does not share any variables with $(Z_{i,j} = 1, \text{Assigned}(A - i'_1 - \cdots - i'_k))$, and that $\neg\exists(\mathcal{B} - \mathcal{B}_{i,j})$ is a decreasing event. Hence by Lemma 5.35 the numerator is at most $p_{i'_1,j'_1} \ldots p_{i'_k,j'_k}$. Now let us examine the denominator. The variable $Z_{i,j}$ does not affect any of the events mentioned in the denominator, so we may remove it from

the conditioning:

$$P(\text{Assigned}(I') \mid \mathcal{E}_3) = P(\text{Assigned}(I') \mid \text{Assigned}(A - I'), \neg\exists(\mathcal{B} - \mathcal{B}_{i,j})),$$

which in turn is at least $\epsilon^{|I'|}$, by iterated application of the induction hypothesis (recall that $\mathcal{B}_{i,j} \cap \mathcal{C} \neq \emptyset$).

Putting this all together, we have that the probability of the event $Z_{i'_1} = \cdots = Z_{i'_k}$ is at most $p^k/\epsilon^{|I'|} \leq (p/\epsilon)^k$. So the random variables $Z$ satisfy $\text{UNC}(p/\epsilon)$ and we have

$$P(\neg\exists\mathcal{B}_{i,j} \mid Z_{i,j} = 1, \text{Assigned}(A), \neg\exists(\mathcal{C} - \mathcal{B}_{i,j})) \geq P^*_{p/\epsilon}(\neg\exists\mathcal{B}_{i,j} \mid Z_{i,j} = 1)$$

The right-hand side is substantially simpler, as there is *no conditioning to link the variables*. Substituting this into (22) and (5.C.1), we get

$$P(\text{Assigned}(i) \mid \text{Assigned}(A), \neg\exists\mathcal{C}) \geq \sum_j p_{i,j} P^*_{p/\epsilon}(\neg\exists\mathcal{B}_{i,j} \mid Z_{i,j} = 1) - \sum_{j,j'} p_{i,j} p_{i,j'}$$

and by our hypothesis the right-hand side is at least $\epsilon$. □

We can now allow all entries of $p$ to tend to 0 at the same rate, which simplifies our formulae:

THEOREM 5.34 (Assignment LLL – restated). *For any element $x \in \mathcal{X}$ and any vector of probability $\lambda \in [0,1]^N$ define*

$$h_x(\lambda) = P^*_\lambda(\neg\exists\mathcal{B}_x \mid Z_x = 1)$$

*For any block $i$ define*

$$H_i(\lambda) = \sum_j \lambda_{i,j} h_{i,j}(\lambda)$$

*Suppose that for all blocks $i \in [n]$ we satisfy the constraint $H_i(\lambda) > 1$. Then the corresponding CSP has a feasible solution.*

PROOF. Let $p = \alpha\lambda$ and let $\epsilon = \alpha$ for some $\alpha > 0$. In order to use Lemma 5.36, it suffices to satisfy the constraint for all $i$

$$\sum_j p_{i,j} h_{i,j}(p/\epsilon) - \sum_{j,j'} p_{i,j} p_{i,j'} \geq \epsilon. \tag{24}$$

Let us fix a block $i$. Suppose we allow $\alpha \to 0$. In this case, (24) will be satisfied for some $\alpha > 0$ sufficiently small if we have

$$\sum_j \lambda_{i,j} \cdot P^*_\lambda(\neg\exists \mathcal{B}_{i,j} \mid Z_{i,j} = 1) > 1$$

As there are only finitely many blocks, there is $\alpha > 0$ sufficiently small which satisfies all constraints simultaneously.

In this case, we claim that there is a positive probability of satisfying Assigned$(i), \overline{B}$ for all blocks $i$ and all bad events $B \in \mathcal{B}$, when we assign variables $Z$ independently Bernoulli-$p$. First, $P(\neg\exists\mathcal{B}) \geq \prod_{x\in\mathcal{X}} P(Z_x = 0)$, since no $B \in \mathcal{B}$ is a tautology; the latter product is clearly positive for small-enough $\alpha$. Next, by Lemma 5.36 and Bayes' Theorem,

$$P(\text{Assigned}(1) \wedge \cdots \wedge \text{Assigned}(n) \mid \neg\exists Y) \geq \prod_{i=1}^{n} \epsilon > 0.$$

In particular, there is a configuration of the $Z_x$ which satisfies all the constraints simultaneously. □

**5.C.2. Computing $P^*$.** In the usual LLL, one can fully specify the underlying random process, so one can compute the probability of a bad event fairly readily. In the assignment LLL, we know that the random variables must satisfy their UNC constraints, but we do not know the full distribution of these variables. This can make it much harder to bound the probability of a bad event.

Roughly speaking, the UNC constraints force the underlying variables to be negatively correlated (or independent). For some types of bad events, this is enough to give strong bounds:

LEMMA 5.38. *For random variables $Z_{x_1}, \ldots, Z_{x_k}$, let $\mu = \lambda_{x_1} + \cdots + \lambda_{x_k}$. Then*

$$P^*_\lambda(Z_{x_1} + \cdots + Z_{x_k} \leq \mu(1+\delta)) \geq 1 - \Big(\frac{e^\delta}{(1+\delta)^{1+\delta}}\Big)^\mu$$

PROOF. The Chernoff upper-tail bound applies to negatively correlated random variables [**92**]. □

Suppose we have a set of bad-events $\mathcal{B}$ which depends on $Z_{x_1}, \ldots, Z_{x_k}$. We are given $\lambda \in [0,1]^N$. Note that $\Omega$ is a probability distribution on $Z_1, \ldots, Z_N$, but we abuse notation to view it as a distribution on $Z_{x_1}, \ldots, Z_{x_k}$ as well. We describe a generic algorithm to compute $P^*_\lambda(\overline{\mathcal{B}})$ (we sometimes just denote $P^*_\lambda(\cdot)$ as $P^*(\cdot)$).

By definition, $\mathcal{B}$ can be written $\mathcal{B} = \{A_1, \ldots, A_m\}$ where each $A_i$ is an atomic event. We may suppose that we are given a minimal such description, that is, we cannot replace any $A_i$ with $A'_i$ where $A'_i \subsetneq A_i$. We assume $\emptyset \notin B$, as otherwise $B$ is a tautology and $P^*(B) = 1$.

We say that a probability distribution $\Omega$ on the variables $Z_{x_1}, \ldots, Z_{x_k}$ is *worst-case* if $P^*_\lambda(B) = P_\Omega(B)$. By finiteness, such $\Omega$ exists. The basic idea of our algorithm is to view each $P_\Omega(\omega)$ as an unknown quantity, where $\omega \in \Omega$ is an atomic event. We write $q_\omega = P_\Omega(\omega)$ for simplicity. In this case, $P_\Omega(B)$ is the sum

$$P_\Omega(B) = \sum_{\omega \in B} q_\omega$$

Furthermore, the UNC constraints can also be viewed as linear constraints in the variable $q_\omega$. For each $x'_1, \ldots, x'_{k'}$, we have the constraint

$$\sum_{\omega_{x'_1} = \cdots = \omega_{x'_{k'}} = 1} q_\omega \leq \lambda_{x'_1} \ldots \lambda_{x'_{k'}}$$

This defines a linear program, in which we maximize the objective function $P^*(B) = \sum_{\omega \in B} q_\omega$ subject to the UNC constraints. The size of this linear program may be enormous, potentially including $2^k$ variables and $2^k$ constraints. However, in many

applications $k$ is a parameter of the problem which can be regarded as constant, so such a program may still be tractable.

In general, we can reduce the number of variables and constraints of this linear program with the following observations:

PROPOSITION 5.39. *There is a distribution $\Omega$ such that $P_\Omega(B) = P^*(B)$ and such that $\Omega$ is only supported on the atomic events $\{0, A_1, \ldots, A_m\}$.*

PROOF. Suppose $\Omega$ satisfies the UNC constraints. Then define $\Omega'$ as follows. For each atomic event $\omega$, if $B$ is not true under $\omega$, then shift the probability mass of $\omega$ to 0; otherwise, shift the probability mass of $\omega$ to any minimal event $A_i$ underneath it. This preserves all UNC constraints as well as the objective function. □

For some types of bad events, there are certain symmetries among classes of variables. In this case, one can assume that the distribution $\Omega$ is symmetric in these variables; hence the probabilities of all such events can be collapsed into a single term.

PROPOSITION 5.40. *Given a group $G \subseteq S_k$, where $S_k$ is the symmetric group on $k$ letters, define the group action of $G$ on a probability distribution $\Omega$ by permutation of the indices and on $\lambda$ by permutation of the coordinates. Suppose $B, \lambda$ are closed under the action of $G$. Then there is a worst-case probability distribution $\Omega$ which is closed under $G$. For this probability distribution $\Omega$, we only need to keep track of a single unknown quantity $q'$ for each* orbit *of $G$.*

PROOF. Given a worst-case distribution $\Omega$, let $\Omega' = \frac{1}{|G|}\sum_{g\in G} g\Omega$. As $B$ is closed under $G$, each of the distributions $g\Omega$ has the same probability of the bad event $B$. As $\lambda$ is closed under $G$, all the UNC constraints are preserved in each $g\Omega$. □

**5.C.3. Comparing the Assignment LLL and PRA.** Although it is not obvious in the form we have stated it, there are connections between the PRA, and in particular Theorem 5.12, with the Assignment LLL as given in Theorem 5.34. Of course, the latter is nonconstructive, while the former is usually a polynomial-time

algorithm. Moreover, for all the *applications* in this chapter, the PRA leads to efficient algorithms, and in most cases, to bounds which are (significantly) improved compared to the Assignment LLL.

However, we have not been able to prove a general theorem to the effect that the PRA is a constructive form of the Assignment LLL or is always better. Indeed, there appear to be some parameter regimes of Theorem 5.19 in which Theorem 5.34 can give slightly better bounds. In order to explain some links between the Assignment LLL and our resampling approach, we next give some intuitive connections between the two.

Note that in Theorem 5.12, the values of $p$ is not significant in itself, only the "inflated" probability distribution $\lambda$. A way to interpret this result is that we are "oversampling" each element $x$. We imagine that there is a process in which individual indicator variable $Z_x$ is Bernoulli-$\lambda_x$, and the $Z_x$ variables are negatively correlated, but we relax the restriction that exactly one element is selected from each variable.

A fractional hitting-set $Q$ provides an upper bound on the probability of the bad event $\mathcal{B}$. Furthermore, this upper bound depends only on the negative correlation of these variables:

PROPOSITION 5.41. *For each element $x = (i, j) \in \mathcal{X}$, we define the indicator variable $Z_x$ which is 1 is $x$ is selected and 0 otherwise. Suppose that the indicator variables $Z_x$ are each individually Bernoulli-$\lambda$ and are negatively correlated. (We are provided no other information on their distribution). Suppose that $Q$ is a fractional hitting-set for $Q$.*

*Then the probability of any bad event $B$ is at most $P(B) \leq \sum_{Y \subseteq B} Q(Y)\lambda^Y$.*

*Furthermore, let $p^*$ denote the maximum possible value of $P(B)$, over all distributions $\Omega$ on the indicator variables which satisfy these two properties, namely that the indicator variables are individually Bernoulli-$\lambda$ and are negatively correlated. Then there is a fractional hitting-set $Q$ with $p^* = \sum_{Y \subseteq B} Q(Y)\lambda^Y$.*

PROOF. Consider the following process: we draw the variables $Z$ as indicated. If the bad event $B$ occurs, we select some violated subset $Y \subseteq B$ with probability proportional to $Q(Y)$. Otherwise we do not draw. The probability of selecting a subset is $P(B)$. But, we can also count it as

$$\begin{aligned} P(B) &= \sum_{Y \subseteq B} P(Y \mid B) \\ &\leq \sum_{Y} P(\forall y \in Y \quad Z_y = 1) P(\text{select } Y) \\ &\leq \sum_{Y} \lambda^Y Q(Y) \end{aligned}$$

We prove that there is a fractional hitting-set $Q$ achieving $p^* = \sum_{Y \subseteq B} Q(Y)\lambda^Y$ by LP duality. We can view the probability of each atomic event $\omega \in \Omega$ as a linear unknown. Then the constraint that the variables are negatively correlated is a linear constraint, and the objective function $P(B)$ is linear. It is not hard to see that a feasible dual corresponds to a fractional hitting-set. $\square$

We can imagine that we are drawing the variables $Z_x$, not by selecting exactly one element from each variable, but according to some more complicated distribution of which we know two things: the marginal distribution of each element is $\lambda_x$; and the elements are negatively correlated.

In this case, the term $G(Q, \lambda)$ is measuring the probability of a bad event. The term $G_x(Q, \lambda)$ is measuring how much of this probability is "due to" the variable $x$. If the variable $x$ does not affect the bad event at all, then $G_x(Q, \lambda) = 0$. If the bad event is equivalent to $x$ being selected (i.e. $B = \{x\}$), then $G_x(Q, \lambda) = \lambda_x$. The standard LLL would not distinguish between a variable affecting the bad event by "a little" or "a lot", but the Partial Resampling Algorithm interpolates smoothly between these extremes.

One important difference between the Assignment LLL and the Partial Resampling Algorithm is that for the Assignment LLL, we *compute* $P^*$ based on the structure

of the bad-events. For the PRA, we *choose* the fractional hitting-set $Q$. For a given bad-event $B$, there is not necessarily a "best" choice of $Q$; a choice of $Q$ can affect the whole dependency structure of the bad-events, and this can have complicated interactions with other bad-events. In general, the optimal choice for the Assignment LLL is not necessarily optimal for the PRA.

Furthermore, when we are analyzing the assignment LLL, we do not ask for bounds on the full set of bad-events $\mathcal{B}$, but rather we use bounds on the set $\mathcal{B}_x$ which are affected by some $x$. In effect, we are able to choose a separate fractional hitting-set for each such $x$; while in the PRA, we are forced to select one fixed fractional hitting-set. Although the PRA is generally stronger, in this one regard it can be slightly weaker than the assignment LLL.

CHAPTER 6

# Partial resampling to approximate covering integer programs

In this section, we consider *positive covering integer programs* – or simply covering integer programs (CIPs) – defined as follows (with $\mathbf{Z}_+$ denoting the set of non-negative integers). We have solution variables $x_1, \ldots, x_n \in \mathbf{Z}_+$, and for $k = 1, \ldots, m$, a system of $m$ *covering constraints* of the form:

$$\sum_i A_{ki} x_i \geq a_k$$

Here $A_k$ is an $n$-long non-negative vector; by scaling, we can assume that $A_{ki} \in [0, 1]$ and $a_k \geq 1$. We can write this more compactly as $A_k \cdot x \geq a_k$. We may optionally have constraints on the size of the solution variables, namely, that we require $x_i \in \{0, 1, \ldots, d_i\}$; these are referred to as the *multiplicity constraints*. Finally, we have some linear objective function, represented by a vector $C \in [0, \infty)^n$. Our goal is to minimize $C \cdot x$, subject to the multiplicity and covering constraints.

This generalize the set cover problem, which can be viewed as a special case in which $a_k = 1, A_{ki} \in \{0, 1\}$. Solving set cover or integer programs exactly is NP-hard [**61**], so a common strategy is to obtain a solution which is approximately optimal. There are at least three ways one may obtain an approximate solution, where OPT denotes the optimal solution-value for the given instance:

(1) the solution $x$ may violate the optimality constraint, that is, $C \cdot x >$ OPT;

(2) $x$ may violate the multiplicity constraint: i.e., $x_i > d_i$ for some $i$;

(3) $x$ may violate the covering constraints: i.e., $A_k \cdot x < a_k$ for some $k$.

These three criteria are in competition. For our purposes, we will demand that our solution $x$ completely satisfies the covering constraints. We will seek to satisfy the multiplicity constraints and optimality constraint as closely as possible. Our emphasis

will be on the optimality constraints that is, we seek to ensure that

$$C \cdot x \leq \beta \times \mathrm{OPT}$$

where $\beta \geq 1$ is "small". The parameter $\beta$, in this context, is referred to as the *approximation ratio*. More precisely, we will derive a randomized algorithm with the goal of satisfying $\mathbf{E}[C \cdot x] \leq \beta \times \mathrm{OPT}$, where the expectation is taken over our algorithm's randomness.

Many approximation algorithms for set cover and its extensions give approximation ratios as a function of $m$, the total number of constraints: e.g., it is known that the greedy algorithm has approximation ratio $(1 - o(1) \ln m$ [**107**]. We often prefer a *scale-free* approximation ratio, that does not depend on the problem size but only on its structural properties. Two cases that are of particular interest are when the matrix $A$ is *row-sparse* (a bounded number of variables per constraint) or *column-sparse* (each variable appears in a bounded number of constraints). We will be concerned solely with the column-sparse setting here. The row-sparse setting, which generalizes problems such as vertex cover, typically leads to very different types of algorithms than the column-sparse setting.

Two common parameters used to measure the column sparsity of such systems are the maximum $l_0$ and $l_1$ norms of the columns; that is,

$$\Delta_0 = \max_i \#k : A_{ki} > 0, \qquad \Delta_1 = \max_i \sum_k A_{ki}$$

Since the entries of $A$ are in $[0, 1]$, we have $\Delta_1 \leq \Delta_0$; it is also possible that $\Delta_1 \ll \Delta_0$.

Approximation algorithms for column-sparse CIP typically yield approximation ratios which are a function of $\Delta_0$ or $\Delta_1$, and possibly other problem parameters as well. These algorithms fall into two main classes. First, there are greedy algorithms: they start by setting $x = 0$, and then increment $x_i$ where $i$ is chosen in some way which "looks best" in a myopic way for the residual problem. These were first developed by [**27**] for set cover, and later analysis (see [**37**]) showed that they give essentially

optimal approximation ratios for set cover. These were extended to CIP in [**40**] and [**31**], showing an approximation ratio of $1 + \ln \Delta_0$. These greedy algorithms are often powerful, but they are somewhat rigid. For instance, it is difficult to adapt these algorithms to take multiplicity constraints into account, or to deal with several objective functions.

A more flexible type of approximation algorithm is based on *linear relaxation*. This replaces the constraint $x_i \in \{0, 1, \ldots, d_i\}$ with the weaker constraint $x_i \in [0, d_i]$. The set of feasible points to this linear relaxation is a polytope, and one can find the exact optimal fractional solution $\hat{x}$. As this is a relaxation, we have $C \cdot \hat{x} \leq \mathrm{OPT}$. It thus suffices to turn the solution $\hat{x}$ into a random integral solution $x$ satisfying $\mathbf{E}[C \cdot x] \leq \beta(C \cdot \hat{x})$. *Randomized rounding* is often employed to transform solutions to the linear relaxation back to feasible integral solutions. The simplest scheme, first applied to this context by [**90**], is to simply draw $x_i$ as *independent* Bernoulli$(\alpha \hat{x}_i)$, for some $\alpha > 1$. When this is used, simple analysis using Chernoff bounds shows that $A_k \cdot x \geq a_k$ simultaneously for all $k$ when $\alpha \geq 1 + c_0(\frac{\log m}{a_k} + \sqrt{\frac{\log m}{a_k}})$, where $c_0 > 0$ is some absolute constant. Thus, the overall solution $C \cdot x$ is within a factor of $1 + O(\frac{\log m}{a_{\min}} + \sqrt{\frac{\log m}{a_{\min}}})$ from the optimum, where $a_{\min} = \min_k a_k \geq 1$.

In [**108**], Srinivasan gave a scale-free method of randomized rounding (ignoring multiplicity constraints), based on the FKG inequality and some proof ideas behind the Lovász Local Lemma. This gave an approximation ratio of $1 + O\big(\frac{\log(\Delta_0+1)}{a_{\min}} + \sqrt{\frac{\log a_{\min}}{a_{\min}} + \frac{\log(\Delta_0+1)}{a_{\min}}}\big)$. The rounding scheme, by itself, only gave a positive (exponentially small) probability of achieving the desired approximation ratio. Srinivasan also gave a polynomial-time derandomization using the method of conditional expectations. The algorithm of Srinivasan can potentially cause a large violation in the multiplicity constraint. In [**67**], Kolliopoulos & Young modified the algorithm of [**108**] to trade off between the approximation ratio and the violation of the multiplicity constraints. For a given parameter $\epsilon \in (0, 1]$, they gave an algorithm which violates each multiplicity constraint "$x_i \leq d_i$" to at most "$x_i \leq \lceil (1 + \epsilon) d_i \rceil$", with an

approximation ratio of $O(1+\frac{\log(\Delta_0+1)}{a_{\min}\cdot\epsilon^2})$. Kolliopoulos & Young also gave a second algorithm which exactly meets the multiplicity constraints and achieves approximation ratio $O(\log(\Delta_0))$.

There has been prior work applying the LLL and the MT algorithm to packing integer programs, such as [**72**]. One technical problem with the LLL is that it only depends on whether bad-events affect each other, not the degree to which they do so. Bad-events which are only slightly correlated are still considered as dependent by the LLL. Thus, a weakness of the LLL for integer programs with arbitrary coefficients (i.e. allowing $A_{ki} \in [0,1]$), is that potentially all the entries of $A_{ki}$ could be extremely small yet non-zero, causing every constraint to affect each other by a tiny amount. For this reason, typical applications of the LLL to column-sparse integer programs have been phrased in terms of the $l_0$ column norm $\Delta_0$. We have already seen in Section 5.4 how to apply the Partial Resampling Algorithm (PRA) to assignment-packing" integer programs. The resulting bounds depend on $\Delta_1$.

**6.0.4. Our contributions.** In this chapter, we give a new randomized rounding scheme for CIP. This combines techniques developed in Chapter 5 and Chapter 4.

We show the following result:

THEOREM 6.1. *Suppose we are given a covering system with a fractional solution* $\hat{x}$. *Let* $\gamma = \frac{\ln(\Delta_1+1)}{a_{min}}$. *Then our randomized algorithm yields a solution* $x \in \mathbf{Z}_+^n$ *satisfying the covering constraints with probability one, and with*

$$\mathbf{E}[x_i] \leq \hat{x}_i\big(1+\gamma+4\sqrt{\gamma}\big)$$

*The expected running time of this rounding algorithm is* $O(mn)$.

Note that this automatically implies that $\mathbf{E}[C\cdot x] \leq \beta C\cdot\hat{x} \leq \beta \times \mathrm{OPT}$ for $\beta = 1+\gamma+4\sqrt{\gamma}$.

Our algorithm has several advantages over previous techniques.

(1) We are able to give approximation ratios in terms of $\Delta_1$, the maximum $l_1$-norm of the columns of $A$. Such bounds are always stronger than those phrased in terms of the corresponding $l_0$-norm.

(2) When $\Delta_1$ is small, our approximation ratios is asymptotically smaller than that of [**108**]. In particular, we avoid the $\sqrt{\frac{\log a_{\min}}{a_{\min}}}$ term in our approximation ratio.

(3) When $\Delta_1$ is large, then our approximation ratio is roughly $\gamma$; this is asymptotically optimal (including having the correct coefficient), and improves on [**108**].

(4) This algorithm is quite efficient, essentially as fast as reading in the matrix $A$.

We view it as interesting that one can "boil down" the parameters $\Delta_1, a_{\min}$ into a single parameter $\gamma$, which seems to completely determine the behavior of our algorithm.

Our partial resampling algorithm in its simplest form could significantly violate the multiplicity constraints. By choosing slightly different parameters for our algorithm (but making no changes otherwise), we can ensure that the multiplicity constraints are nearly satisfied, at the cost of a worsened approximation ratio:

THEOREM 6.2. *Suppose we are given a covering system with a fractional solution* $\hat{x}$. *Let* $\gamma = \frac{\ln(\Delta_1+1)}{a_{min}}$. *For any given* $\epsilon \in (0, 1]$, *our algorithm yields a solution* $x \in \mathbf{Z}_+^n$ *satisfying the covering constraints with probability one, and with*

$$x_i \leq \lceil \hat{x}_i(1+\epsilon) \rceil, \qquad \mathbf{E}[x_i] \leq \hat{x}_i(1 + 4\sqrt{\gamma} + 4\gamma/\epsilon)$$

This is an asymptotic improvement over the approximation ratio of [**67**], in three different ways:

(1) It depends on the $\ell_1$-norm of the columns, not the $\ell_0$ norm;

(2) When $\gamma$ is large, it is smaller by a full factor of $1/\epsilon$;

(3) When $\gamma$ is small, it gives an approximation ratio which approaches 1, at a rate independent of $\epsilon$.

We also give matching lower bounds on the achievable approximation ratios. The formal statements of these results contain numerous qualifiers and technical conditions.

(1) When $\gamma$ is large, then assuming the Exponential-Time Hypothesis, any polynomial-time algorithm to solve the CIP (ignoring multiplicity constraints) must have approximation ratio $\gamma - O(\log \gamma)$.
(2) When $\gamma$ is large, then assuming $P \neq NP$, any polynomial-time algorithm to solve the CIP while respecting the multiplicity constraints within a $1+\epsilon$ multiplicative factor, must have approximation ratio $\Omega(\gamma/\epsilon)$.
(3) When $\gamma$ is small, then the integrality gap of the CIP is $1 + \Omega(\gamma)$.

Finally, we give an extension to covering programs with multiple linear criteria. Our extension is much simpler than the algorithm of [**108**]; we show that *even conditional on our solution $x$ satisfying all the covering constraints*, not only do we have $\mathbf{E}[C_l \cdot x] \leq \beta C_l \cdot \hat{x}$ but that in fact the values of $C_l \cdot x$ are concentrated, roughly equivalent to the $x_i$ being independently distributed as Bernoulli with probability $\beta \hat{x}_i$. Thus, for each $l$ there is a very high probability that we have $C_l \cdot x \approx C_l \cdot \hat{x}$ and in particular there is a good probability that we have $C_l \cdot x \approx C_l \cdot \hat{x}$ simultaneously for all $l$.

THEOREM 6.3 (Informal). *Suppose we are given a covering system with a fractional solution $\hat{x}$ and with $r$ objective functions $C_1, \ldots, C_r$, whose entries are in $[0,1]$. Let $\gamma = \frac{\ln(\Delta_1+1)}{a_{min}}$. Then our solution $x$ satisfies the covering constraints with probability one; with probability at least $1/2$,*

$$\forall l \qquad C_l \cdot x \leq \beta(C_l \cdot \hat{x}) + O(\sqrt{\beta(C_l \cdot \hat{x}) \ln r})$$

*where $\beta = 1 + \gamma + 4\sqrt{\gamma}$. (A similar result is possible, if we also want to ensure that $x_i \leq \lceil \hat{x}_i(1+\epsilon) \rceil$; then the approximation ratio is $1 + 4\sqrt{\gamma} + 4\gamma/\epsilon$.)*

This significantly improves on [**108**], in terms of both the approximation ratio as well as the running time. Roughly speaking, the algorithm of [**108**] gave an approximation ratio of $O(1 + \frac{\log(1+\Delta_0)}{a_{\min}})$ (worse than the approximation ratio in the single-criterion setting) and a running time of $n^{O(\log r)}$ (polynomial time only when $r$, the number of objective functions, is constant).

**6.0.5. Outline.** In Section 6.1, we develop a randomized rounding algorithm when the fractional solution satisfies $\hat{x} \in [0, 1/\alpha]^n$; here $\alpha \geq 1$ is a key parameter which we will discuss how to select in later sections. This randomized rounding produces produces a binary solution vector $x \in \{0,1\}^n$, for which $\mathbf{E}[x_i] \approx \alpha \hat{x}_i$. This algorithm is the Partial Resampling variant of the MT algorithm of Chapter 5 combined with the analysis of lopsidependency in Chapter 4. However, it is not possible to combine results in a truly "black-box" way from these chapters. So we will give a self-contained and unified presentation of these algorithms here. Many of their technical complications can be dramatically simplified for the CIP setting.

In Section 6.2, we will develop a deterministic quantization scheme to handle fractional solutions of arbitrary size, using the algorithm of Section 6.1 as a subroutine. We will show an upper bound on the sizes of the variables $x_i$ in terms of the fractional $\hat{x}_i$. We will also show an upper bound on $\mathbf{E}[x_i]$, which we state in a generalized form without making reference to column-sparsity or other properties of the matrix $A$.

In Section 6.3, we consider the case in which we have a lower bound $a_{\min}$ on the RHS constraint vectors $a_k$, as well as an upper bound $\Delta_1$ on the $\ell_1$-norm of the columns of $A$. Based on these values, we set key parameters of the rounding algorithm, to obtain good approximation ratios as a function of $a_{\min}, \Delta_1$. We give approximation ratios for both the case in which the the multiplicity constraints are ignored, and the case in which they are violated by a multiplicative factor of $1 + \epsilon$.

In Section 6.4, we construct a variety of lower bounds on achievable approximation ratios. These are based on integrality gaps as well as hardness results. These show that the approximation ratios developed in Section 6.3 are essentially optimal for most values of $\Delta_1, a_{\min}, \epsilon$, particularly when $\Delta_1$ is large.

In Section 6.5, we show that our randomized rounding scheme obeys a negative correlation property, allowing us to show concentration bounds on the sizes of the objective functions $C_l \cdot x$. This significantly improves on the algorithm of [**108**]; we show asymptotically better approximation ratios in many regimes, and we also give a polynomial-time algorithm regardless of the number of objective functions.

### 6.1. The RELAXATION algorithm

We first consider the case when all the values of $\hat{x}$ are small. In this case, we present an algorithm which we label *RELAXATION*. Initially, this algorithm draws each $x_i$ as an independent Bernoulli trials with probability $p_i = \alpha \hat{x}_i$, for some parameter $\alpha > 1$. This will satisfy many of the covering constraints, but there will still be some left unsatisfied. We loop over all such constraint; whenever a constraint $k$ is unsatisfied, we modify the solution as follows: for each variable $i$ which has $x_i = 0$, we set $x_i$ to be an independent Bernoulli random variable with probability $p_i = \sigma A_{ki} \alpha \hat{x}_i$. Here $\sigma \in [0, 1]$ is another parameter which we will also discuss how to select.

**Algorithm 1** pseudocode for the algorithm RELAXATION

```
function RELAXATION(x̂, A, a, σ, α)          ▷ Approximates 0-1 ILPs
    for i from 1, ..., n do                  ▷ Initialization
        x_i ~ Bernoulli(α x̂_i)
    for k from 1, ..., m do                  ▷ Partial resampling scheme
        while A_k · x < a_k do
            for i from 1, ..., n do
                if x_i = 0 then
                    x_i ~ Bernoulli(σ A_ki α x̂_i)
    return x
```

Whenever we encounter an unsatisfied constraint $k$, and we draw new values for the variables, we refer to this as *resampling* the constraint $k$. There is an alternative way of looking at the resampling procedure. Instead of setting each variable $x_i =$

1 with probability $\sigma A_{ki}\alpha\hat{x}_i$, we instead suppose that we first select a subset $Y \subseteq [n]$ of the variables, where each variable $i$ currently satisfying $x_i = 0$ goes into $Y$ independently with probability $\sigma A_{ki}$. Then, for each variable $i \in Y$, we draw $x_i \sim$ Bernoulli($\alpha\hat{x}_i$). It is clear that this two-part sampling procedure is equivalent to the one-step procedure described in Algorithm 1. We now let $Y_{kj}$ denote the $j$th chosen resampled set for constraint $k$ (if this exists). We say that variable $i$ is *resampled* if $i \in Y_{kj}$.

The witness trees are particularly simple, in our context: they are just linear trees. (We will later generalize the notion of witness tree to handle multi-criteria programs.) More formally, a witness tree will be one of the following:

(1) The empty list, which we denote $\cdot$;

(2) A single index $k \in [m]$, with a list of resampled sets $\langle Y_{k1}, \ldots, Y_{kj}\rangle$ for $j \geq 1$.

We do not claim in the second case, that $Y_{kj}$ is the *last* resampled set for constraint $k$. Indeed, it is possible that this resampling procedure does not terminate and some constraint has an infinite number of resampled sets.[1]

Fundamental to the correctness of our analysis is the following *Witness Tree Lemma*:

LEMMA 6.4 (Witness Tree Lemma). *Let $Z_1, \ldots, Z_j$ be subsets of $[n]$. The probability of encountering the witness tree $\langle Z_{k1}, \ldots, Z_{kj}\rangle$ for a constraint $k$ is at most*

$$P(\langle Z_1, \ldots, Z_j\rangle \text{ is a witness tree for constraint } k) \leq \prod_{l=1}^{j} f_k(Z_l)$$

*where we define*

$$f_k(Z) = (1-\sigma)^{-a_k} \prod_i (1 - A_{ki}\sigma) \prod_{i \in Z} \frac{(1-p_i)A_{ki}\sigma}{1 - A_{ki}\sigma}$$

[1]The standard method of generating witness trees for the MT algorithm may need to list the resampled sets for *multiple* constraints. Restricting to these single-constraint witness trees is the main contribution of Chapter 4 which in this context allows us to show simpler and stronger bounds on the behavior of our algorithm.

PROOF. If there are fewer than $j$ resamplings of constraint $k$, then this witness tree cannot occur. So suppose that there are (at least) $j$ resamplings of constraint $k$.

First, consider some variable $i$ that appears $r$ times among the sets $Z_1, \ldots, Z_l$. We only resample variables which are currently equal to zero, so it must be that during the first $r$ resamplings of the variable $x_i$ (where the first resampling is taken to be the initial sampling stage), that we set $x_i = 0$. Note that some or all of these resamplings may have taken place for earlier constraints $k' < k$. Regardless of which constraint causes these resamplings, every resampled variable is set equal to one independently of all past events with probability $p_i$. Thus, the probability that variable $i$ is set equal to zero for its first $r$ resamplings is $(1-p_i)^r$. The total probability of such events is

$$\prod_i (1-p_i)^{\#\ l \text{ s.t. } i \in Z_l} = \prod_{l=1}^{j} \prod_{i \in Z_l} (1-p_i) \tag{25}$$

which accounts for the term $\prod_{i \in Z}(1-p_i)$ in $f_k(Z)$.

Now consider the probability that each set $Z$ is chosen to be $Y_{kl}$, conditional on having chosen all the previous sets $Y_{k1}, \ldots, Y_{k(l-1)}$ and on any choice of resampled values for the variables. Let us denote $x'$ by the value of the variables $x$ at the time this $l$th resampling of constraint $k$ occurs. As we put each $i \in Y_{kl}$ with probability $A_{ki}\sigma$ independently, the probability that all $i \in Z$ go into $Y_{kl}$ is $\prod_{i \in Z} A_{ki}\sigma$. By the same token, if $x'_i = 0$, then $i$ avoids going into $Y_{kl}$ with probability $1 - A_{ki}\sigma$. Therefore, the overall probability of selecting $Y_{kl} = Z$ is given by

$$\begin{aligned}
P(Y_{kl} = Z) &\leq \prod_{i \in Z} A_{ki}\sigma \prod_{i \notin Z, x'_i = 0} (1 - A_{ki}\sigma) \\
&= \Big(\prod_{i \in Z} A_{ki}\sigma\Big)\Big(\prod_{i \notin Z}(1 - A_{ki}\sigma)\Big)\Big(\prod_{x'_i = 1}(1 - A_{ki}\sigma)^{-1}\Big) \\
&\qquad \text{only variables with } x'_i = 0 \text{ exist in } Y_{kl} \\
&= \prod_i (1 - A_{ki}\sigma) \prod_{i \in Z} \frac{A_{ki}\sigma}{1 - A_{ki}\sigma} \prod_{x'_i = 1} (1 - A_{ki}\sigma)^{-1}
\end{aligned}$$

We now observe that constraint $k$ can only be resampled if and only if it is currently unsatisfied, so it must be that $A_k x' \leq a_k$. By Proposition 6.28, we thus have:

$$\prod_{x'_i=1} (1 - A_{ki}\sigma)^{-1} \leq (1-\sigma)^{-a_k}$$

further implying:

$$P(Y_{kl} = Z) \leq (1-\sigma)^{-a_k} \prod_i (1 - A_{ki}\sigma) \prod_{i \in Z} \frac{A_{ki}\sigma}{1 - A_{ki}\sigma} \tag{26}$$

The total probability of observing the witness tree $\langle Z_1, \ldots, Z_j \rangle$ is the product of (25) and (26), over all $l = 1, \ldots, j$. This gives us the claimed result. □

PROPOSITION 6.5. *For any constraint $k$ define*

$$s_k = (1-\sigma)^{-a_k} e^{-\sigma\alpha A_k \cdot \hat{x}}.$$

.

*Then we have $\sum_{Z \subseteq [n]} f_k(Z) \leq s_k$.*

PROOF. We have

$$\begin{aligned}
\sum_{Z \subseteq [n]} f_k(Z) &= \sum_{Z \subseteq [n]} (1-\sigma)^{-a_k} \prod_i (1 - A_{ki}\sigma) \prod_{i \in Z} \frac{(1-p_i)A_{ki}\sigma}{1 - A_{ki}\sigma} \\
&= (1-\sigma)^{-a_k} \prod_i (1 - A_{ki}\sigma) \sum_{Z \subseteq [n]} \prod_{i \in Z} \frac{(1-p_i)A_{ki}\sigma}{1 - A_{ki}\sigma} \\
&= (1-\sigma)^{-a_k} \prod_i (1 - A_{ki}\sigma) \prod_{i=1}^{n} \Big(1 + \frac{(1-p_i)A_{ki}\sigma}{1 - A_{ki}\sigma}\Big) \\
&= (1-\sigma)^{-a_k} \prod_i (1 - A_{ki}p_i\sigma) \\
&\leq (1-\sigma)^{-a_k} e^{-\sigma \sum_i A_{ki}p_i} \\
&= (1-\sigma)^{-a_k} e^{-\sigma\alpha A_k \cdot \hat{x}}
\end{aligned}$$

□

To gain some intuition about this expression, note that if we set $\sigma = 1 - 1/\alpha$ (which is not necessarily the optimal choice for the overall algorithm), then we have

$$s_k = \alpha^{a_k} e^{-A_k \cdot \hat{x}(\alpha - 1)}$$

and this can be recognized as the Chernoff lower-tail bound. Namely, this is an upper bound on the probability that a sum of independent $[0, 1]$-random variables, with mean $\alpha A_k \cdot \hat{x}$, will become as small as $a_k$. This makes sense: for example at the very first step of the algorithm (before any resamplings are performed), then $A_k \cdot x$ is *precisely* a sum of independent Bernoulli variables with mean $\alpha A_k \cdot \hat{x}$. The event we are measuring (the probability that a constraint $k$ is resampled) is *precisely* the event that this sum is smaller than $a_k$.

Proposition 6.6 gives a bound on what is effectively the running time of the algorithm:

PROPOSITION 6.6. *The expected number of resamplings steps made by the algorithm RELAXATION is at most* $\sum_k \frac{1}{e^{\sigma \alpha A_k \cdot \hat{x}} (1-\sigma)^{a_k} - 1}$.

PROOF. Consider the probability that there are $l$ resamplings of constraint $k$. For this to occur, then a necessary condition is that there are sets $Z_1, \ldots, Z_l$ such that $\langle Z_1, \ldots, Z_l \rangle$ is a witness tree for $k$. This has probability $f_k(Z_1) \ldots f_k(Z_l)$. Taking a union-bound on $l$, we have:

$$P(\text{There are} \geq l \text{ resamplings of constraint } k) \leq \sum_{Z_1, \ldots, Z_l} f_k(Z_1) \ldots f_k(Z_l) = s_k^l$$

Thus, the expected number of resamplings of constraint $k$ is at most $\sum_{l=1}^{\infty} s_k^l = s_k/(1 - s_k)$. □

We can also give a bound, crucially, on the distribution of the variables $x_i$ at the *end* of the resampling process.

THEOREM 6.7. *For any $i \in [n]$, the probability this algorithm sets $x_i = 1$ is at most*

$$P(x_i = 1) \leq \alpha \hat{x}_i \Big(1 + \sigma \sum_k \frac{A_{ki}}{e^{\sigma \alpha A_k \cdot \hat{x}} (1-\sigma)^{a_k} - 1}\Big)$$

PROOF. For $x_i = 1$, the necessary conditions are either $x_i$ was set to one during the initial sampling stage, or it was caused on the $j$th resampling of some constraint $k$. In the latter case, we must have $i \in Y_{kj}$. The probability of the first event is $p_i$.

Now, let us consider the probability of the second event. There must be sets $Z_1, \ldots, Z_j$ with $i \in Z_j$, such that $\langle Z_1, \ldots, Z_j \rangle$ is the witness tree for the $j$th resampling of constraint $k$. By Lemma 6.4, this has probability $f_k(Z_1) \ldots f_k(Z_j)$. Suppose that variable $i$ appears $r$ times among the sets $Z_1, \ldots, Z_j$. The probability calculated in Lemma 6.4 is based on the fact that for the first $r$ resamplings of variable $i$ we must choose $x_i = 0$. However, another necessary condition is that we select $x_i = 1$ during the $j$th resampling of constraint $k$. This event has probability $p_i$, conditional on all the events discussed in Lemma 6.4. Hence, the overall probability of setting $x_i = 1$ at the $j$th resampling of constraint $k$ is at most $p_i f_k(Z_1) \ldots f_k(Z_j)$.

Thus, summing over $k, j$, we have:

$$\begin{aligned}
&P(x_i = 1) \\
&\quad \leq p_i + \sum_k \sum_j \sum_{\substack{Z_1, \ldots, Z_j \\ i \in Z_j}} p_i f_k(Z_1) \ldots f_k(Z_j) \\
&\quad \leq p_i \Big(1 + \sum_k \sum_{j=0}^{\infty} s_k^j \sum_{Z \ni i} f_k(Z)\Big) \\
&\quad \leq p_i \left(1 + \sum_k \frac{1}{1-s_k} \sum_{Z \ni i} (1-\sigma)^{-a_k} \prod_{i'} (1 - A_{ki'}\sigma) \prod_{i' \in Z} \frac{(1-p_{i'})A_{ki'}\sigma}{1 - A_{ki'}\sigma}\right) \\
&\quad \leq p_i \left(1 + \sum_k \frac{1}{1-s_k} (1-\sigma)^{-a_k} \Big(\frac{(1-p_i)A_{ki}\sigma}{1 - A_{ki}\sigma}\Big) \prod_{i'} (1 - A_{ki'}\sigma) \prod_{i' \neq i} \Big(1 + \frac{(1-p_{i'})A_{ki'}\sigma}{1 - A_{ki'}\sigma}\Big)\right) \\
&\quad \leq p_i \Big(1 + \sum_k \frac{1}{1-s_k} (1-\sigma)^{-a_k} \sigma A_{ki} (1-p_i) e^{-\sigma(\alpha A_k \cdot \hat{x} - A_{ki} p_i)}\Big)
\end{aligned}$$

$$= p_i\Big(1 + \sum_k \frac{\sigma A_{ki} s_k (1-p_i) e^{\sigma A_{ki} p_i}}{1 - s_k}\Big)$$

We now note that $A_{ki} \leq 1, \sigma \leq 1$ and so by Proposition 6.29 we have $(1 - p_i)e^{\sigma A_{ki} p_i} \leq 1$ giving:

$$P(x_i = 1) \leq p_i\Big(1 + \sum_k \frac{s_k}{1 - s_k} A_{ki}\sigma\Big) \leq \alpha \hat{x}_i\Big(1 + \sigma \sum_k \frac{A_{ki}}{e^{\sigma\alpha A_k \cdot \hat{x}}(1-\sigma)^{a_k} - 1}\Big)$$

□

## 6.2. Extension to the case where $\hat{x}_i$ is large

**6.2.1. Overview.** In the previous section, we described the RELAXATION algorithm under the assumption that $\hat{x}_i \leq 1/\alpha$. This assumption was necessary because each variable $i$ is chosen to be drawn as a Bernoulli random variable with probability $p_i = \alpha \hat{x}_i$. In this section, we give a rounding scheme to cover fractional solutions $\hat{x}$ of unbounded size. We first give an overview of this process.

Our goal is to extend the approximation ratio $\rho_i = \alpha\Big(1 + \sigma \sum_k \frac{A_{ki}}{e^{\sigma\alpha A_k \cdot \hat{x}}(1-\sigma)^{a_k} - 1}\Big)$ of Section 6.1. First, note that if we have a variable $i$, and a solution to the LP with fractional value $\hat{x}_i$, we can sub-divide it into two new variables $y_1, y_2$ with fractional values $\hat{y}_1, \hat{y}_2$ such that $\hat{y}_1 + \hat{y}_2 = \hat{x}_i$. Now, whenever the variable $x_i$ appears in the covering system, we replace it $y_1 + y_2$. This process of sub-dividing variables can force all the entries in the fractional solution to be arbitrarily small. We can round the RELAXATION algorithm of this subdivided fractional solution, obtaining an integral solution $y_1, y_2$ and hence $x_i = y_1 + y_2$. Observe that the approximation ratios for the two new variables both equal to $\rho_i$ itself. Thus $\mathbf{E}[x_i] = \mathbf{E}[y_1 + y_2] \leq \rho_i y_1 + \rho_i y_2 \leq \rho_i \hat{x}_i$.

By subdividing the fractional solution, we can always ensure that we obtain the same approximation for the general case (in which $\hat{x}$ is unbounded) as in the case in which $\hat{x}$ is restricted to entries of size at most $1 - 1/\alpha$. However, this may violate

the multiplicity constraints: in general, if we subdivide a fractional solution $\hat{x}_i$ into $\hat{y}_1, \ldots, \hat{y}_l$, and then set $x_i = y_1 + \cdots + y_l$, then $x_i$ could become as large as $l$.

There is another, simpler way to deal with large values $\hat{x}_i$: for any variable with $\hat{x}_i \geq 1/\alpha$, simply set $x_i = 1$. Then, we certainly are guaranteed that $\mathbf{E}[x_i] \leq \alpha\hat{x}_i \leq \rho_i\hat{x}_i$. Let us see what problems this procedure might cause. Consider some variable $i$ with $\hat{x}_i = r \geq 1/\alpha$.[2] Because we have fixed $x_i = 1$, we may remove this variable from the covering system. When we do so, we obtain a residual problem $A', a'$, in which the $i$th column of $A$ is replaced by zero and all the RHS vectors $a_k$ are replaced by $a'_k = a_k - A_{ki}$.

Suppose that variable $i$ appears in constraint $k$ with another variable $i'$ with $A_{ki} = 1$. We want to bound $\mathbf{E}[x'_i]$ in terms of $\beta_{i'}$; to do so, we want to show that constraint $k$ contributes $\frac{A_{ki'}}{e^{\sigma\alpha A_k \cdot \hat{x}}(1-\sigma)^{a_k}-1}$ to $\mathbf{E}[x'_i]$. Now, in the residual problem, we replace $a_k$ with $a_k - 1$ and we replace $A_k \cdot \hat{x}$ with $A_k \cdot \hat{x} - r$. Thus, constraint $k$ contributes the following to $\beta'_{i'}$:

$$\frac{A_{ki'}}{e^{\sigma\alpha(A_k\cdot\hat{x}-r)}(1-\sigma)^{a_k-1}-1} = \frac{A_{ki'}}{e^{\sigma\alpha(A_k\cdot\hat{x})}(1-\sigma)^{a_k}e^{-\sigma\alpha r}(1-\sigma)^{-1}-1}$$

Observe that if $r > \frac{-\ln(1-\sigma)}{\alpha\sigma}$, then this is *larger* than the original contribution term we wanted to show, namely $\rho_i = \frac{A_{ki'}}{e^{\sigma\alpha A_k\cdot\hat{x}}(1-\sigma)^{a_k}}$. Thus, there is a critical cut-off value $\theta = \frac{-\ln(1-\sigma)}{\alpha\sigma}$; when $\hat{x}_i > \theta$, then forcing $x_i = 1$ gives a good approximation ratio for variable $i$ but may have a worse approximation ratio for other variables which interact with it.

We can now combine these two methods for handling large entries of $\hat{x}_i$. For any variable $i$, we first subdivide variable $i$ into multiple variables $\hat{y}_1, \ldots \hat{y}_l$ with fractional value $\theta$, along with one further entry $\hat{y}_{l+1} \in [0, \theta]$. We immediately set $y_1, \ldots, y_l = 1$. If $\hat{y}_{l+1} > 1/\alpha$, we set $y_{l+1} = 1$ as well, otherwise we will apply the RELAXATION algorithm for it. At the end of this procedure, we know that $x_i = y_1 + \cdots + y_{l+1} \leq (l+1) = \lceil \frac{\hat{x}_i}{\theta} \rceil$. We also know that $\mathbf{E}[x_i] \leq \alpha(\hat{y}_1 + \cdots + \hat{y}_l) + \rho_i\hat{y}_{l+1} \leq$

[2]To gain intuition, the reader may consider the case in which $r > 1$. In this case, it is obvious that this is a bad rounding procedure. It is instructive to trace through exactly why it fails badly.

$\rho_i(\hat{y}_1 + \cdots + \hat{y}_{l+1}) = \rho_i \hat{x}_i$. Thus, we get a good approximation ratio and a good bound on the multiplicity of $x_i$.

**6.2.2. The ROUNDING algorithm.** For each variable $i$, let $v_i = \lfloor \hat{x}_i/\theta \rfloor$, where we define

$$\theta = \frac{-\ln(1-\sigma)}{\alpha\sigma}$$

We define $F_i = \hat{x}_i - v_i\theta$ which we can write as $F_i = \hat{x}_i \mod \theta$. We also define:

$$G_i = \begin{cases} 0 & \text{if } F_i < 1/\alpha \\ 1 & \text{if } F_i \geq 1/\alpha \end{cases}, \qquad \hat{x}'_i = \begin{cases} F_i & \text{if } F_i < 1/\alpha \\ 0 & \text{if } F_i \geq 1/\alpha \end{cases}$$

We form the residual problem $a'_k = a_k - \sum_i A_{ki}(G_i + v_i)$. We then run the RELAXATION algorithm on the residual problem, which satisfies the condition that $x'_i \in [0, 1/\alpha]^n$. This is summarized in Algorithm 2.

**Algorithm 2** pseudocode for the algorithm ROUNDING

```
function ROUNDING(x̂, A, σ, α)          ▷ Approximates arbitrary ILPs
    Set θ = −ln(1−σ)/(ασ).
    for i from 1, ..., n do
        v_i = ⌊x̂_i/θ⌋
        F_i = x̂_i mod θ.
        G_i = 1 if F_i ≥ 1/α, G_i = 0 otherwise.
        x̂'_i = 0 if F_i ≥ 1/α, x̂'_i = F_i otherwise.
    for k from 1, ..., m do
        Set a'_k = a_k − Σ_i A_ki(G_i + v_i)
    Compute x' = RELAXATION(x̂', A, a', σ, α)
    Return x = G + v + x'
```

We begin by showing a variety of simple bounds on the variables before and after the quantization steps.

PROPOSITION 6.8. *Suppose that $x' \in \{0,1\}^n$ satisfies the residual covering constraints, that is, $A_k \cdot x' \geq a'_k$ for all $k = 1, \ldots, m$.*

*Then the solution vector returned by the ROUNDING algorithm, defined by $x = G + v + x'$, satisfies the original covering constraints. Namely, $A_k \cdot x \geq a_k$ for all $k = 1, \ldots, m$.*

PROOF. For each $k$ we have:

$$A_k \cdot x = A_k \cdot (x' + G + v) = A_k \cdot x' + \sum_i A_{ki}(G_i + v_i) \geq a'_k + \sum_i A_{ki}(G_i + v_i) = a_k$$

□

PROPOSITION 6.9. *For any $i$ we have*

$$\hat{x}_i - v_i\theta - G_i\theta \leq \hat{x}'_i \leq \hat{x}_i - v_i\theta - G_i/\alpha$$

PROOF. If $G_i = 0$, then both of the bounds hold with equality. So suppose $G_i = 1$. In this case, we have $1/\alpha \leq \hat{x}_i - v_i\theta \leq \theta$. So $x_i - v_i\theta - G_i/\alpha \geq \theta - 1/\alpha \geq 0$ and $x_i - v_i\theta - G_i\theta \leq \theta - \theta = 0$ as required. □

PROPOSITION 6.10. *For any $i$, at the end of the procedure ROUNDING, we have*

$$x_i \leq \left\lceil \hat{x}_i \frac{\alpha\sigma}{-\ln(1-\sigma)} \right\rceil$$

PROOF. First, suppose $\hat{x}_i$ is not a multiple of $\theta$. Then $x_i = x'_i + G_i + \lfloor x_i/\theta \rfloor$. Note that if $G_i = 1$, then $\hat{x}'_i = 0$ which implies that $x'_i = 0$. So $G_i + v_i \leq 1$ and hence $x_i \leq 1 + \lfloor x_i/\theta \rfloor = \lceil x_i/\theta \rceil$.

Next, suppose $\hat{x}_i$ is a multiple of $\theta$. Then $G_i = \hat{x}'_i = 0$ and so $x'_i = 0$ and we have $x_i = \lfloor x_i/\theta \rfloor = \lceil x_i/\theta \rceil$. □

The next result shows that the quantization steps can only *decrease* the inflation factor for the RELAXATION algorithm. Proposition 6.11 is the reason for our choice of $\theta$.

PROPOSITION 6.11. *For any constraint $k$, we have*

$$(1-\sigma)^{a'_k} e^{\sigma\alpha A_k \cdot \hat{x}'} \geq (1-\sigma)^{a_k} e^{\sigma\alpha A_k \cdot \hat{x}}$$

PROOF. Let $r = \sum_i A_{ki}(G_i + v_i)$. By definition, we have $a'_k = a_k - r$. We also have:

$$\begin{aligned} A_k \cdot \hat{x}' &= \sum_i A_{ki}\hat{x}'_i \\ &\geq \sum_i A_{ki}(\hat{x}_i - v_i\theta - G_i\theta) \qquad \text{by Proposition 6.9} \\ &= a_k - r\theta \end{aligned}$$

Then

$$\begin{aligned} (1-\sigma)^{a'_k} e^{\sigma\alpha A_k \cdot \hat{x}'} &= (1-\sigma)^{a_k - r} e^{\sigma\alpha A_k \cdot \hat{x}'} \\ &\geq (1-\sigma)^{a_k - r} e^{\sigma\alpha(a_k - r\theta)} \\ &= (1-\sigma)^{-a_k} e^{-\sigma\alpha a_k} \times ((1-\sigma)e^{\theta\sigma\alpha})^{-r} \\ &= (1-\sigma)^{-a_k} e^{-\sigma\alpha a_k} \end{aligned}$$

□

We can now show an overall bound on the behavior of the ROUNDING algorithm

THEOREM 6.12. *At the end of the ROUNDING algorithm, we have for each variable $i$*

$$\mathbf{E}[x_i] \leq \alpha\hat{x}_i\Big(1 + \sigma \sum_k \frac{A_{ki}}{e^{\sigma\alpha a_k}(1-\sigma)^{a_k} - 1}\Big)$$

*The expected number of resamplings for the RELAXATION algorithm is at most $\sum_k \frac{1}{e^{\sigma\alpha a_k}(1-\sigma)^{a_k}-1}$.*

PROOF. By Theorem 6.7, the probability that $x'_i = 1$ is at most

$$\begin{aligned} P(x'_i = 1) &\leq \alpha\hat{x}'_i\Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a'_k} e^{\sigma\alpha A_k \cdot \hat{x}'} - 1}\Big) \\ &\leq \alpha\hat{x}'_i\Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a_k} e^{\sigma\alpha a_k} - 1}\Big) \qquad \text{by Proposition 6.11} \end{aligned}$$

Hence we estimate $\mathbf{E}[x_i]$ by:

$$\begin{aligned}
\mathbf{E}[x_i] &= v_i + G_i + \mathbf{E}[x_i'] \\
&\le v_i + G_i + \alpha\hat{x}_i'\Big(1 + \sigma\sum_k \frac{A_{ki}}{(1-\sigma)^{a_k}e^{\sigma\alpha a_k} - 1}\Big) \\
&\le v_i + G_i + \alpha(\hat{x}_i - \theta v_i - G_i/\alpha)\Big(1 + \sigma\sum_k \frac{A_{ki}}{(1-\sigma)^{a_k}e^{\sigma\alpha a_k} - 1}\Big) \qquad \text{by Proposition 6.9} \\
&\le v_i(1-\alpha\theta) + \alpha\hat{x}_i\Big(1 + \sigma\sum_k \frac{A_{ki}}{(1-\sigma)^{a_k}e^{\sigma\alpha a_k} - 1}\Big) \\
&\le \alpha\hat{x}_i\Big(1 + \sigma\sum_k \frac{A_{ki}}{(1-\sigma)^{a_k}e^{\sigma\alpha a_k} - 1}\Big) \qquad \text{as } \alpha\theta \ge 1
\end{aligned}$$

This shows the bound on $\mathbf{E}[x_i]$. The bound on the expected number of resamplings is similar. □

### 6.3. Bounds in terms of $a_{\min}, \Delta_1$

So far, we have given bounds on the behavior of ROUNDING algorithm which are as general as possible. Theorem 6.12 can be applied to systems, in which there may be multiple types of variables and constraints. However, we can obtain a simpler bound by reducing these to two simple parameters, namely $\Delta_1$, the maximum $l_1$-norm of any column of $A$, and $a_{\min} = \min_k a_k$. We will first assume that $a_{\min} \ge 1, \Delta_1 \ge 1$. Later, Theorem 6.15 will show that we can always ensure that this holds with a simple pre-processing step.

THEOREM 6.13. *Suppose we are given a covering system with $\Delta_1 \ge 1, a_{min} \ge 1$ and with a fractional solution $\hat{x}$. Let $\gamma = \frac{\ln(\Delta_1+1)}{a_{min}}$.*

*Then with appropriate choices of $\sigma, \alpha$ we may run the ROUNDING algorithm on this system to obtain a solution $x \in \mathbf{Z}_+^n$ satisfying*

$$\mathbf{E}[x_i] \le \hat{x}_i\big(1 + \gamma + 4\sqrt{\gamma}\big), \qquad x_i \le \left\lceil \hat{x}_i \frac{\frac{1}{2}\gamma + \sqrt{\gamma}}{\ln(1+\sqrt{\gamma})} \right\rceil$$

*The expected running time of this algorithm is $O(mn)$.*

PROOF. We set $\sigma = 1 - 1/\alpha$ and apply Theorem 6.12. We have for each variable $i$:

$$\begin{aligned}
\mathbf{E}[x_i] &\leq \hat{x}_i \alpha \Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a_k} e^{\sigma \alpha a_k} - 1}\Big) \\
&= \hat{x}_i \alpha \Big(1 + (1 - 1/\alpha) \sum_k \frac{A_{ki}}{e^{a_k(\alpha-1)} \alpha^{-a_k} - 1}\Big) \\
&\leq \hat{x}_i \alpha \Big(1 + (1 - 1/\alpha) \sum_k \frac{A_{ki}}{e^{a_{\min}(\alpha-1)} \alpha^{-a_{\min}} - 1}\Big) \\
&\leq \hat{x}_i \Big(\alpha + (\alpha - 1) \frac{\Delta_1}{e^{a_{\min}(\alpha-1)} \alpha^{-a_{\min}} - 1}\Big)
\end{aligned}$$

Now substituting $\alpha = 1 + \gamma + 2\sqrt{\gamma}$ and $a_{\min} = \ln(\Delta_1 + 1)/\gamma$ gives

$$\mathbf{E}[x_i] \leq \hat{x}_i \Big(1 + \gamma + 2\sqrt{\gamma} + (2\sqrt{\gamma} + \gamma) \frac{\Delta_1}{(\Delta_1 + 1)^{\frac{2\sqrt{\gamma} + \gamma - 2\ln(1+\sqrt{\gamma})}{\gamma}} - 1}\Big)$$

Proposition 6.30 shows that this is decreasing function of $\Delta_1$. We are assuming $a_{\min} \geq 1$, which implies that $\Delta_1 \geq e^\gamma - 1$. We can thus obtain an upper bound by substituting $\Delta_1 = e^\gamma - 1$, yielding

$$\mathbf{E}[x_i] \leq \hat{x}_i \Big(1 + \gamma + 2\sqrt{\gamma} + \frac{(e^\gamma - 1)\left(\gamma + 2\sqrt{\gamma}\right)}{\frac{e^{\gamma + 2\sqrt{\gamma}}}{\left(\sqrt{\gamma}+1\right)^2} - 1}\Big) \tag{27}$$

Some simple analysis of the RHS of (27) shows that we have

$$\mathbf{E}[x_i] \leq \hat{x}_i (1 + \gamma + 4\sqrt{\gamma})$$

To show the bound on the size of $x_i$, we apply Proposition 6.10, giving us

$$x_i \leq \left\lceil \hat{x}_i \frac{\alpha\sigma}{-\ln(1-\sigma)} \right\rceil = \left\lceil \hat{x}_i \frac{\frac{1}{2}\gamma + \sqrt{\gamma}}{\ln(1 + \sqrt{\gamma})} \right\rceil$$

Next, we will analyze the runtime of this procedure. The initial steps of rounding and forming the residual can be done in time $O(mn)$. By Theorem 6.12, the expected

number of resampling steps made by the RELAXATION algorithm is at most

$$\begin{aligned}\mathbf{E}[\text{Resampling Steps}] &\leq \sum_k \frac{1}{e^{a_k(\alpha-1)}\alpha^{-a_k}-1} \\ &\leq \frac{m}{e^{a_{\min}(\alpha-1)}\alpha^{-a_{\min}}-1} \\ &\leq \frac{m}{(\Delta_1+1)^{\frac{\gamma+2\sqrt{\gamma}-2\ln(1+\sqrt{\gamma})}{\gamma}}-1} \\ &\leq \frac{m}{\Delta_1} \leq m\end{aligned}$$

In each resampling step, we must draw a new random value for all the variables; this can be easily done in time $O(n)$. The algorithm in its entirety is bounded by $O(mn)$ as required. □

In Theorem 6.13, we may violate the multiplicity constraints considerably. By adjusting our parameters, we may have better control of the multiplicity constraints.

THEOREM 6.14. *Suppose we are given a covering system with $\Delta_1 \geq 1, a_{min} \geq 1$, as well as a fractional solution $\hat{x}$. Let $\gamma = \frac{\ln(\Delta_1+1)}{a_{min}}$.*

*Let $\epsilon \in [0,1]$ be given. Then, with an appropriate choice of $\sigma, \alpha$ we may run the ROUNDING algorithm on this system to obtain a solution $x \in \mathbf{Z}_+^n$ satisfying*

$$x_i \leq \lceil \hat{x}_i(1+\epsilon) \rceil, \qquad \mathbf{E}[x_i] \leq \hat{x}_i(1+4\sqrt{\gamma}+4\gamma/\epsilon)$$

*The expected run-time is $O(mn)$.*

PROOF. First, suppose $\gamma \leq \epsilon^2/2$. In this case, we apply Theorem 6.13. We are guaranteed that $x_i \leq \lceil \hat{x}_i \frac{\gamma/2+\sqrt{\gamma}}{\ln(1+\sqrt{\gamma})} \rceil$ and some simple analysis shows that this is at most $\lceil \hat{x}_i(1+\epsilon) \rceil$. We then have $\mathbf{E}[x_i] \leq 1+4\sqrt{\gamma}+\gamma \leq 1+4\sqrt{\gamma}+4\gamma/\epsilon$ as desired.

Next, suppose $\gamma \geq \epsilon^2/2$. We set $\alpha = \frac{-(1+\epsilon)\ln(1-\sigma)}{\sigma}$, where $\sigma \in (0,1)$ is a parameter to be determined. Then by Proposition 6.10, we have $x_i \leq \lceil \hat{x}_i(1+\epsilon) \rceil$ at the end of the ROUNDING algorithm.

We apply Theorem 6.12 to estimate $\mathbf{E}[x_i]$:

$$\begin{aligned}\mathbf{E}[x_i] &\leq \hat{\alpha} x_i \Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a_k} e^{\sigma \alpha a_k} - 1}\Big) \\ &= \hat{\alpha} x_i \Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{-a_k \epsilon} - 1}\Big) \\ &\leq \hat{\alpha} x_i \Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{-a \epsilon} - 1}\Big) \\ &\leq \hat{\alpha} x_i \Big(1 + \sigma \frac{\Delta_1}{(1-\sigma)^{-a \epsilon} - 1}\Big)\end{aligned}$$

Now set $\sigma = 1 - e^{-\gamma/\epsilon}$; observe that this is indeed in the range $(0, 1)$. This ensures that $(1-\sigma)^{-a\epsilon} = \Delta_1 + 1$ and hence we have

$$\mathbf{E}[x_i] \leq \hat{x}_i \alpha (1+\sigma) = \hat{x}_i \Big(\epsilon^{-1}(2 + \frac{1}{e^{\gamma/\epsilon} - 1})(1+\epsilon)\gamma\Big)$$

Some simple calculus shows that this coefficient $\epsilon^{-1}(2 + \frac{1}{e^{\gamma/\epsilon}-1})(1+\epsilon)\gamma$ is at most $1 + \epsilon + (2 + 2/\epsilon)\gamma$. By our assumption that $\epsilon \in [0, 1]$ and our assumption that $\epsilon^2/2 \leq \gamma$, this is at most $1 + \sqrt{2\gamma} + 4\gamma/\epsilon$ as desired.

The bound on the running time follows the same lines as Theorem 6.13. □

We now show how to ensure that $a_{\min} \geq \Delta_1 \geq 1$:

THEOREM 6.15. *Suppose we are given a covering system $A, a$ with $\gamma = \ln(\Delta_1 + 1)/a_{min}$. Then, in time $O(mn)$, one can produce a modified system $A', a'$ which satisfies the following properties:*

(1) *The integral solutions of $A, a$ are precisely the same as the integral solutions of $A', a'$;*
(2) *$a'_{min} \geq 1$ and $\Delta'_1 \geq 1$;*
(3) *We have $\gamma' \leq \gamma$, where $\gamma' = \ln(\Delta'_1 + 1)/a'_{min}$.*

PROOF. First, suppose that there is some entry $A_{ki}$ with $A_{ki} > a_k$. In this case, set $A'_{ki} = a_k$. Observe that any integral solution to the constraint $A_k \cdot x \geq a_k$ also satisfies $A'_k \cdot x \geq a_k$, and vice-versa. This step can only decrease $\Delta_1$ and hence $\gamma' \leq \gamma$.

After this step, one can assume that $A_{ki} \leq a_k$ for all $k, i$. Now suppose there are some constraints with $a_k \leq 1$. In this case, replace row $A_k$ with $A'_k = A_k/a_k$ and replace $a_k$ with $a'_k = 1$. Because of our assumption that $A_{ki} \leq a_k$ for all $k, i$, the new row of the matrix still satisfies $A'_k \in [0,1]^n$. This step ensures that $a'_k \geq 1$ for all $k$. Also, every column in the matrix is scaled up by at most $1/a_k \leq 1/a_{\min}$, so we have $\Delta'_1 \leq \Delta_1/a_{\min}$ and $a'_{\min} = 1$. We then have

$$\gamma' = \ln(\Delta' + 1)/a'_{\min} = \ln(\Delta_1/a_{\min} + 1) \leq \frac{\ln(\Delta_1 + 1)}{a_{\min}} = \gamma.$$

Finally, suppose that $\Delta_1 \leq 1$. In this case, observe that we must have $A_{ki} \leq \Delta_1$ for all $k, i$. Thus, we can scale up both $A, a$ by $1/\Delta_1$ to obtain $A' = A/\Delta_1, a' = a/\Delta_1$. This gives $\Delta' = 1, a'_{\min} = a_{\min}/\Delta_1$

$$\gamma' = \frac{\ln(1+1)}{a_{\min}/\Delta_1} \leq \frac{\ln(\Delta_1 + 1)}{a_{\min}} = \gamma$$

□

### 6.4. Lower bounds on approximation ratios

In this section, we provide lower bounds on the performance of algorithms to solve covering integer programs. These bounds fall into two categories. First, we show hardness results, namely that there is no polynomial-time algorithm which can achieve significantly better approximation ratios than we do. These are based on Feige's celebrated result on the inapproximability of set cover [**37**], which was later improved by Moshkovitz [**88**]. Next, we show integrality gap constructions. Our rounding algorithm transforms a solution to the LP relaxation to an integral solution; we show that there are some CIP instances for which the optimal integral solution has an objective-function value that is close to our approximation bound times the objective-function value of any optimal fractional solution. This implies that any algorithm *which is based on the LP relaxation* cannot have an improved approximation ratio.

The formal statements of these results contain numerous qualifiers and technical conditions. So, we will summarize our results informally here:

(1) Under the Exponential Time Hypothesis (ETH), when $\gamma$ is large any polynomial-time algorithm to solve the CIP (ignoring multiplicity constraints) must have approximation ratio $\gamma - \ln \gamma + \Omega(1)$. Furthermore, there is an integrality gap of $\gamma - O(\log \gamma)$ (no intractability assumptions needed). By contrast, the algorithm of Theorem 6.13 achieves approximation ratio $\gamma + O(\sqrt{\gamma})$.

(2) Under the assumption that $P \neq NP$, when $\gamma$ is large any polynomial-time algorithm to solve the CIP while respecting the multiplicity constraints within a $1+\epsilon$ multiplicative factor, must have approximation ratio $\Omega(\gamma/\epsilon)$. By contrast, the algorithm of Theorem 6.14 achieves approximation ratio $O(\gamma/\epsilon)$.

(3) When $\gamma$ is small, the integrality gap of the CIP is $1 + \Omega(\gamma)$; by contrast, the algorithm of Theorem 6.13 achieves approximation ratio $1 + O(\sqrt{\gamma})$.

We note that the parameter $\gamma$ depends on two parameters $\Delta_1, a_{\min}$. Thus, in order to show these results, we must show hardness across a wide range of the parameters $\Delta_1, a_{\min}$. By contrast, typical hardness results for set cover only depend on a single parameter (such as $\Delta_1$ or $n$).

**6.4.1. Hardness results.** Our hardness results are all reductions from the construction of Feige [**37**], which was later tightened by Moshkovitz [**88**]. They gave the following nearly-tight hardness results for approximating set cover:

THEOREM 6.16 ([**88**]). *Suppose we are given set cover instances on a ground set* $[n]$ *with optimum solution of value* $T$*. Then:* [3]

(A) *Under the assumption* $P \neq NP$*, there is no polynomial-time algorithm which can find a solution of value* $\leq T \times c \ln n$*, for any constant* $c < 1$*.*

[3]These results do not follow from the original, conference version of [**88**]. They follow immediately from a result stated in the full journal version (unpublished), although they are not stated explicitly.

(B) *Under the exponential time hypothesis (i.e., that any algorithm for SAT requires time at least $2^{\Omega(n)}$), there is some constant $c > 0$, such that no polynomial-time algorithm can find a solution of value $\leq T \times (\ln n - c \ln \ln n)$.*

Note that, as shown by [**107**], the greedy algorithm for set cover achieves approximation ratio $\ln n - \ln \ln n + \Theta(1)$. Thus, the approximation ratio of Theorem 6.16(B) is nearly tight (up to a coefficient of $\ln \ln n$).

We will show that other variants of the covering integer program can be reduced to set cover. This will show hardness results for CIP, which closely match the bounds achieved by our algorithms.

PROPOSITION 6.17. *There is some constant $c > 0$ for which the following holds. Let $a \geq 1$ be any fixed integer. Assuming the exponential time hypothesis, there is no polynomial-time algorithm $\mathcal{A}$ with the following behavior: given a CIP with $a_{min} \geq a$ and optimum solution of value $T$, then $\mathcal{A}$ finds a solution $x \in \mathbf{Z}_+^n$, of value $\leq \beta T$, where we have*

$$\beta = \frac{\ln(\Delta_1 + 1) - c \ln \ln(\Delta_1 + 1)}{a}.$$

*Observe that $\beta \geq (1 - o(1))\gamma$, hence CIP cannot be approximated within $(1 - o(1))\gamma$, even with an arbitrary violation of the multiplicity constraints.*

PROOF. Suppose for contradiction that for every $c > 0$ there exists such an $\mathcal{A}$. Now suppose we are given some set cover instance, with optimum solution $v$, on some ground set $[n]$. Let $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq 2^{[n]}$. Now, for each element $k \in [n]$ in the ground set, we have a constraint

$$\sum_{i : k \in S_i} x_i \geq a$$

and we have an objective function $C \cdot x = \sum x_i$; that is, each variable has weight one.

The resulting CIP instance contains $n$ constraints.[4] Furthermore, we may assume that none of the sets $S_i$ are the full set $[n]$, as otherwise this would have a trivial solution of weight 1. So we may assume $\Delta_1 \leq n-1$, as well as $a_{\min} \geq a$. Observe that if we are given a solution $\mathcal{S}_0$ to this set cover instance, of weight $v$, then the corresponding CIP has a solution defined by

$$x_i = \begin{cases} a & \text{if } S_i \in \mathcal{S}_0 \\ 0 & \text{otherwise} \end{cases}$$

which has weight $T = av$.

Now, we run $\mathcal{A}$ on this resulting system, and we obtain a solution $x'$ of weight $\leq \beta T$. Now construct the solution $\mathcal{S}_0'$ to the original set cover instance:

$$\mathcal{S}_0' = \{S_i \mid x_i' \geq 1\}$$

It is not hard to see that $\mathcal{S}_0'$ is a valid solution to the original set cover instance and $|\mathcal{S}_0'| \leq \sum_i x_i'$. So our algorithm has overall approximation ratio

$$\begin{aligned} \frac{|\mathcal{S}_0'|}{|\mathcal{S}_0|} &\leq \frac{\sum_i x_i'}{v} \leq \frac{\beta T}{v} \leq \frac{\beta a v}{v} \leq \beta a \\ &\leq \frac{\ln(\Delta_1 + 1) - c \ln\ln(\Delta_1 + 1)}{a} a \\ &\leq \ln n - c \ln\ln n \end{aligned}$$

Thus, we have constructed an approximation algorithm for set cover with approximation ratio $\ln n - c \ln\ln n$ for every $c > 0$. This contradicts Theorem 6.16(B). $\square$

A slight modification of this argument, taking advantage of a construction of [**67**], can give approximation hardness when the multiplicity constraints are violated by at most a factor of $(1+\epsilon)$.

[4] It is somewhat confusing that for set cover, the standard terminology uses $m$ for the number of sets, which correspond to variables, and $n$ for the size of the ground set, which corresponds to constraints. For CIP, one uses the opposite terminology: there are $m$ constraints on $n$ variables.

PROPOSITION 6.18. *Let $a \geq 1$ and $\epsilon \in (0,1)$ be any fixed real numbers. Assuming $P \neq NP$ there is no polynomial-time algorithm $\mathcal{A}$ with the following behavior: given a CIP with $a_{min} \geq a$ and optimum solution of value $T$ satisfying $x_i \in \{0, \ldots, d_i\}$, then $\mathcal{A}$ finds a solution $x$ satisfying $x_i \in \{0, \ldots, \lceil d_i(1+\epsilon)\rceil\}$ of value $\leq \beta T$, where*

$$\beta = \frac{\ln(\Delta_1 + 1)}{2a\epsilon}$$

*Hence CIP cannot be approximated within $o(\frac{\gamma}{\epsilon})$ as long as the multiplicity constraints are respected within a multiplicative factor of $(1+\epsilon)$.*

PROOF. Suppose there exists such an $\mathcal{A}$. Now suppose we are given some set cover instance, with optimum solution $v$, on some ground set $[n]$. Let $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq 2^{[n]}$. Now, for each element $k \in [n]$ in the ground set, we have a constraint

$$\frac{a}{K(1+\epsilon)+1} x_{m+k} + \sum_{i:k \in S_i} x_i \geq a$$

and we have an objective function $C \cdot x = \sum_{i=1}^{m} x_i$; that is, each variable $x_1, \ldots, x_m$ has weight one, and each variable $x_{m+1}, \ldots, x_{m+n}$ has weight zero. We set $d_i = \infty$ for $i = 1, \ldots, m$ and we set $d_i = K$ for $i = m+1, \ldots, m+n$; here $K$ is a large integer parameter, which we will specify shortly. (In particular, for $K$ sufficiently large, all the coefficients in this constraint are in the range $[0,1]$.)

The resulting CIP instance contains $n$ constraints and we may assume $\Delta_1 \leq n-1$, as well as $a_{\min} \geq a$. Observe that if we are given a solution $\mathcal{S}_0$ to this set cover instance, of weight $v$, then the corresponding CIP has a solution defined by

$$x_i = \begin{cases} a - \frac{aK}{K(1+\epsilon)+1} & \text{if } S_i \in \mathcal{S}_0 \text{ and } i \leq m \\ K & \text{if } m+1 \leq i \leq m+n \\ 0 & \text{otherwise} \end{cases}$$

which has weight $T = (a - \frac{aK}{K(1+\epsilon)+1})v$.

We run $\mathcal{A}$ on this resulting system, and we obtain a solution $x'$ of weight $\leq \beta T$. Now construct the following solution $\mathcal{S}_0'$ to the original set cover instance:

$$\mathcal{S}_0' = \{S_i \mid x_i' \geq 1\}$$

which has weight $|S_0'| \leq \sum_{i=1}^m x_i'$.

We claim that this satisfies the set cover instance; for any $k \in [n]$, there is some $i \in [n]$ with $S_i \in \mathcal{S}_0', S_i \ni k$ iff $\sum_{i:k\in S_i} x_i' \geq 1$; we can estimate the latter in

$$\begin{aligned}\sum_{i:k\in S_i} x_i' &\geq a - \frac{aK}{K(1+\epsilon)+1} x'_{m+k} \\ &\geq a - a\frac{\lceil K(1+\epsilon)\rceil}{K(1+\epsilon)+1} \\ &> a - a\frac{K(1+\epsilon)+1}{K(1+\epsilon)+1} = 0\end{aligned}$$

Thus, we have that $\sum_{i:k\in S_i} x_i' > 0$. Since $x'$ is an integral vector, it follows that $\sum_{i:k\in S_i} x_i' \geq 1$ and the covering constraint is satisfied.

So our algorithm has overall approximation ratio

$$\begin{aligned}\frac{|\mathcal{S}_0'|}{|\mathcal{S}_0|} \leq \frac{\sum_{i=1}^m x_i'}{v} \leq \frac{\beta T}{v} \leq \frac{\beta(a - \frac{aK}{K(1+\epsilon)+1})v}{v} &\leq \beta a(1 - \frac{K}{K(1+\epsilon)+1}) \\ &\leq \frac{\ln(\Delta_1+1)}{2a\epsilon} a(1 - \frac{K}{K(1+\epsilon)+1}) \\ &\leq \tfrac{1}{2}\ln n(\frac{1}{\epsilon} - \frac{K}{\epsilon K(1+\epsilon)})\end{aligned}$$

Now suppose we take $K \geq \frac{1-\epsilon}{\epsilon^2}$. Then, one can verify that $(\frac{1}{\epsilon} - \frac{K}{\epsilon K(1+\epsilon)}) \leq 1$ and hence we have

$$\frac{|\mathcal{S}_0'|}{|\mathcal{S}_0|} \leq \tfrac{1}{2}\ln n$$

Thus, we have constructed an approximation algorithm for set cover with approximation ratio $\frac{1}{2}\ln n$, which is forbidden by Theorem 6.16(A). □

**6.4.2. Integrality gaps.** We next show a variety of integrality gaps for the CIP. These constructions work as follows: we give a CIP instance, as well as an upper

bound on the weight of the fractional solution $\hat{T}$ and a lower bound on the weight of any integral solution $T$. This automatically implies that any algorithm which convert a fractional solution into an integral solution, as our algorithm does, must cause the weight to increase by at least $T/\hat{T}$.

These methods show only a limited type of hardness — namely, they only restrict the approximation ratio of LP-based algorithms. However, they are unconditional and self-contained results, which do not require strong complexity assumptions such as ETH.

We show an integrality gap which matches Proposition 6.17 when $\gamma$ is large. We are also able to show an integrality gap for the regime in which $\gamma \to 0$.

PROPOSITION 6.19. *Let $a \geq 1, D \geq 1$ be given. There is a covering program, with $a_{min} = a, \Delta_1 \leq D$, and which satisfies the following property. Let $\hat{T}$ be the optimal value of this covering program, subject to the constraints $x \in \mathbf{R}_+^n$ and let $T$ be the optimal value of the covering program, subject to the constraints $x \in \mathbf{Z}_+^n$. Then we have*

$$T/\hat{T} \geq \frac{\ln(D+1)}{a} - O(\frac{\log\log(D+1)}{a}) \geq \gamma - O(\log\gamma)$$

PROOF. First, we claim that we can assume that $D$ is larger than any desired constant. For, suppose $D \leq D_0$. Then, for some constant $c > 0$, we have $\ln(D+1) - c\log\log(D+1) \leq 1$ for all $D \leq D_0$. We certainly have $T/\hat{T} \geq 1$, so we have $T/\hat{T} \geq \ln(D+1) - c\log\log(D+1) \geq \frac{\ln(D+1)-c\log\log(D+1)}{a}$. Likewise, we can assume that $\ln(D+1) \geq a$. We will make both of these simplifications for the remainder of the proof.

There are $m = \lfloor D \rfloor$ constraints, which we will form randomly as follows: we select exactly $s$ positions $i_1, \ldots, i_s$ uniformly at random in $s$ without replacement, where $s = \lceil pn \rceil$; here $n \to \infty$ and $p \to 0$ as functions of $D$. We then set $A_{ki_1} = \cdots = A_{ki_s} = 1$; all other entries of $A_k$ are set to zero. The RHS vector is always equal to $a$. The

objective function $C$ is defined by $C \cdot x = \sum x_i$; that is, each variable is assigned weight one.

We can form a fractional solution $\hat{x}$ by setting $\hat{x}_i = \frac{a}{s}$. As each constraint contains exactly $s$ entries with coefficient one, this satisfies all the covering constraints. Thus, the optimal fractional solution has value $\hat{T} \leq na/s = a/p$.

Now suppose we fix some integral solution of weight $\sum x_i = t$. Let $I \subseteq [n]$ denote the support of $x$, that is, the values $i \in [n]$ such that $x_i > 0$; we have $|I| = r \leq t$. In each constraint $k$, there is a probability of $\binom{n-r}{s}/\binom{n}{s}$ that $A_{ki} = 0$ for all $i \in I$. If this occurs, then certainly $A \cdot x = 0$ and the covering constraint is violated. Thus, the probability that $x$ satisfies constraint $k$ is at most $1 - \frac{\binom{n-r}{s}}{\binom{n}{s}}$. As all the constraints are independent, the total probability that $x$ satisfies all $m$ constraints is at most:

$$\begin{aligned} P(x \text{ satisfies all constraints and has weight } t) &\leq (1 - \frac{\binom{n-r}{s}}{\binom{n}{s}})^m \\ &\leq \exp(-m\frac{\binom{n-t}{s}}{\binom{n}{s}}) \\ &\leq \exp(-m(\frac{n-s-(t-1)}{n})^t) \\ &\leq \exp(-m(1-p-t/n)^t) \qquad \text{as } s \leq pn+1 \end{aligned}$$

We want to ensure that there are *no* good integral solutions. To upper-bound the probability that there exists such a good $x$, we take a union-bound over all integral $x$. In fact, our estimate only depended on specifying the support of $x$, not the values it takes on there, so we only need to take a union bound over all subsets of $[n]$ of cardinality $\leq t$. There are at most $\sum_{r=0}^{t} \binom{n}{r} \leq n^t$ such sets, and thus we have

$$\begin{aligned} P(\text{Some } x \text{ satisfies all constraints}) &\leq n^t \exp(-m(1-p-t/n)^t) \\ &\leq \exp(t \ln n - m(1-p)^t + mt^2/n) \end{aligned}$$

We now set $n = mt$, and obtain

$$\begin{aligned} P(\text{Some } x \text{ satisfies all constraints}) &\leq \exp(t(1+\ln(mt)) - m(1-p)^t) \\ &\leq \exp(t^2 \ln m - m\exp(-pt - p^2 t)) \\ &\quad \text{for } m, p, t \text{ sufficiently small} \end{aligned}$$

If this expression is $< 1$, then that implies that there is a positive probability that no integral solution exists. Hence, we can ensure that all integral solutions satisfy $T > t$. Now, some simple analysis shows that this expression is $< 1$ when $p = 1/\ln m$ and $t = p^{-1}(\ln m - 10 \ln\ln m)$ and $m$ sufficiently large. Thus we have

$$\begin{aligned} T/\hat{T} &\geq \frac{p^{-1}(\ln m - 10\ln\ln m)}{a/p} \\ &\geq \frac{\ln m - O(\log\log m)}{a} \\ &\geq \frac{\ln(D+1)}{a} - O(\log(\frac{\log(D+1)}{a})) \end{aligned}$$

as we have claimed.

□

Proposition 6.19 does not give a useful bound when $a > \ln(D+1)$. Proposition 6.20, which is based on a construction of [**115**], covers that case:

PROPOSITION 6.20. *For any $g \in (0,1)$ and $D \geq 2^{14/g}$, there is a covering program which satisfies $\Delta_1 \leq D, \frac{\ln(D+1)}{a_{min}} \leq g$, and which satisfies also the following integrality gap property: Let $\hat{T}$ be the optimal value of this covering program, subject to the constraints $x \in \mathbf{R}_+^n$ and let $T$ be the optimal value of the covering program, subject to the constraints $x \in \mathbf{Z}_+^n$. Then we have*

$$T/\hat{T} \geq 1 + g/8 \geq 1 + \Omega(\gamma)$$

*In particular, it is not possible to show an approximation ratio for LP rounding of the form* $1 + o(\frac{\log(\Delta+1)}{a_{min}})$.

PROOF. We set $n = 2^q - 1$ where $q = \lfloor 1 + \log_2 D \rfloor$. We will view the integers from $1, \ldots, n$ as corresponding to the non-zero binary strings of length $q$. Thus, if $i, i' \in \{1, \ldots, s\}$, then we write $i \cdot i'$ to denote the binary dot-product. Namely if we have $i = i_0 + 2i_1 + 4i_2 + \ldots$ and $i' = i'_0 + 2i'_1 + 4i'_2 + \ldots$ where $i_j, i'_j \in \{0, 1\}$, then we define $i \cdot i' = \oplus_{l=0}^{q-1} i_l i'_l$.

The covering system is defined as follows: For each $k \in \{1, \ldots, n\}$ we have a constraint

$$\sum_{i:(k \cdot i)=0} x_i \geq a,$$

The objective function is $C \cdot x = \sum_{i=1}^{n} x_i$.

For each $i = 1, \ldots, n$, we have $\sum_k (k \cdot i) = 2^{q-1} \leq D$. Thus $\Delta_1 \leq D$ as desired. Also, we have $\frac{\ln(D+1)}{a_{\min}} = \frac{g \ln(D+1)}{\lfloor 1 + \log_2 D \rfloor - 1} < g$. Thus, the covering program satisfies the stated bounds on $a_{\min}, D$.

We form the fractional solution $\hat{x}$ by setting $\hat{x}_i = \frac{a}{2^{q-1}}$ for $i = 1, \ldots, n$. This shows that the optimal fractional solution has value $\hat{T} \leq \frac{(2^q - 1)a}{2^{q-1}} \leq 2a$.

Now consider some integral solution $x \in \mathbf{Z}_+^n$ with $\sum_i x_i = T$. We can write $x$ as a sum of basis vectors, $\vec{x} = e_{y_1} + \cdots + e_{y_T}$, where $y_1, \ldots, y_T$ are not necessarily distinct. Consider the quantity

$$V = \sum_k \sum_{1 \leq i_1 < \cdots < i_{q-1} \leq T} \mathcal{I}((k \cdot y_{i_1}) = \ldots (k \cdot y_{i_{q-1}}) = 0)$$

We count $V$ in two different ways. First, for any $i_1, \ldots, i_{q-1}$, by linear algebra over $GF(2)$ there must exist at least one $k \neq 0$ which is orthogonal to all $y_{i_1}, \ldots, y_{i_{q-1}}$. Hence we have $V \geq \binom{T}{q-1}$.

Second, for any $k$, there are at most $T - a$ choices of $y_i$ which are orthogonal to $k$. Thus we have $V \leq (2^q - 1)\binom{T-a}{q-1}$. We have shown a lower bound on $V$ and an upper bound on $V$. The lower bound on $V$ must be smaller than the upper bound

on $V$, or otherwise we would have a contradiction. Thus, a necessary condition for $x$ to satisfy the covering constraints is that

$$\frac{\binom{T}{q-1}}{\binom{T-a}{q-1}(2^q-1)} \leq 1 \tag{28}$$

We claim that we must have $T \geq (q-1)(2/g+1/4)$. As the LHS of (29) is a decreasing function of $T$, it suffices to show that (29) is violated for $T = (q-1)(2/g+1/4)$. Rearranging some terms and recalling that $a = (q-1)/g$, we see that it suffices to show that

$$\frac{\binom{(q-1)(2/g+1/4)}{q-1}}{\binom{(q-1)(1/g+1/4)}{q-1}(2^q-1)} > 1 \tag{29}$$

We use the bounds $2^q - 1 \leq 2^q$ and the bound on the factorial $\sqrt{2\pi}r!r^{r+\frac{1}{2}}e^{-r} \leq r! \leq er!r^{r+\frac{1}{2}}e^{-r}$, to obtain the following condition, which implies (29):

$$2^{-q}(4-3g)^{\frac{-3gq+5g+4q-4}{4g}}(8-3g)^{\frac{3gq-5g-8q+8}{4g}}(g+4)^{-\frac{(g+4)q+g-4}{4g}}(g+8)^{\frac{(g+8)q+g-8}{4g}} > \frac{e^2}{2\pi} \tag{30}$$

We can increase the RHS of (30) slightly to $e$ to simplify the calculations, and take the logarithm to solve for $q$. This gives us the following condition, which implies (30):

$$q > 1 + \frac{2g\big(-2+\ln(4-3g)-\ln(8-3g)-\ln(g+4)+\ln(g+8)-2\ln 2\big)}{4g\ln 2-(4-3g)\ln(4-3g)+(8-3g)\ln(8-3g)+(4+g)\ln(4+g)-(8+g)\ln(8+g)} \tag{31}$$

The RHS of (31) is a function of $g$ alone. Simple but tedious analysis (see Proposition 6.33) shows that it is at most $14/g$.

But note that $q = \lfloor 1 + \log_2 D \rfloor \geq \log_2 D$; thus, our bound on the size of $D$ guarantees that indeed $q > 14/g$. So (31) $\Rightarrow$ (30) $\Rightarrow$ (29) $\Rightarrow T \geq (q-1)(2/g+1/4)$. The integrality gap is then given by

$$T/\hat{T} \geq \frac{(q-1)(2/g+1/4)}{2a} = \frac{2a+ag/4}{2a} = 1 + g/8$$

□

## 6.5. Multi-criteria Programs

One extension of the covering integer program framework is the presence of multiple linear objectives. Suppose now that instead of a single linear objective, we have multiple objectives $C_1 \cdot x, \ldots, C_r \cdot x$. We also may have some over-all objective function $D$ defined by the following:

$$D(x_1, \ldots, x_n) = D(C_1 \cdot x, \ldots, C_r \cdot x)$$

For example, we might have $D = \max_l C_l \cdot x$ or we might have $D = \sum_l (C_l \cdot x)^2$.

We note that the greedy algorithm, which is powerful for set cover, is not obviously useful in this case. However, depending on the precise form of the function $D$, it may be possible to solve the fractional relaxation to optimality. For example, if $D = \max_l C_l \cdot x$, then this amounts to a linear program of the form $\min t$ subject to $C_l \cdot x \leq t$.

For our purposes, the exact algorithm used for this fractional relaxation is not relevant. Suppose we are given some solution $\hat{x}$. We now want to find a solution $x$ such that we have *simultaneously* $C_l \cdot x \approx C_l \cdot \hat{x}$. Showing bounds on the expectations alone is not sufficient — it might be the case that $\mathbf{E}[C_l \cdot x] \leq \beta C_l \cdot \hat{x}$, but the random variables $C_1 \cdot x, \ldots, C_r \cdot x$ are negatively correlated.

In [**108**], Srinivasan gave a construction which provided this type of simultaneous approximation guarantee. This algorithm was based on randomized rounding, which succeeded only with an exponentially small probability. Srinivasan also gave a derandomization of this process, leading to a somewhat efficient algorithm. Some technical difficulties with this algorithm lead to worsened approximation ratios compared to the single-criterion setting, roughly of the order $O(1 + \frac{\log(\Delta_0+1)}{a_{\min}})$, and running times of the order $O(n^{\log r})$. In particular, this was only polynomial if $r$ was constant.

In this section, we will show that at the end of the ROUNDING algorithm, the values of $C_l \cdot x$ are concentrated around their means. This will establish that there is a good probability that we have $C_l \cdot x \approx \mathbf{E}[C_l \cdot x]$, *simultaneously* for all $l$. Thus, our

algorithm automatically gives good approximation ratios for multi-criteria problems; the ratios are essentially the same as for the single-criterion setting, and there is no extra computational burden.

We begin by showing that the values of $x$ produced by the RELAXATION algorithm we obtain are *negatively correlated.* We will show how to form witness trees that explain not just why we have $x_i = 1$ (as we have seen in Lemma 6.4), but that explain simultaneously why we have $x_{i_1} = \cdots = x_{i_s} = 1$.

PROPOSITION 6.21. *Let $R \subseteq [n]$. Suppose that at some point in the RELAXATION algorithm we have $x_i = 1$ for all $i \in R$.*

*Then there is set $S \subseteq [m]$, with $|S| \leq |R|$, and a collection of sets $\mathcal{Z} = \{Z_{s,j} \mid s \in S, j = 1, \ldots, w_s\}$, which which satisfy the following properties:*

(A1) *There is some $R' \subseteq R$ and a* bijective *function $f : R' \to S$, such $r \in Z_{f(r),w_{f(r)}}$ for all $r \in R$.*

(A2) *We have $Y_{s,j} = Z_{s,j}$ for all $s \in S$ and $j \leq w_s$.*

(A3) *For each $i \in R$, suppose that $i \notin Z_{s,j}$ for all $s, j$. Then the initial sampling stage of the RELAXATION algorithm sets $x_i = 1$.*

(A4) *Suppose that $i \in R \cap Z_{s,j}$ but $i \notin Z_{(s',j')}$ for any $(s', j') >_{lex} (s, j)$. Then the $j$th resampling for the constraint $s$ during the RELAXATION algorithm sets $x_i = 1$.*

*We say that the listing $S, w, \mathcal{Z}$ collectively form a* witness tree *for the events $\bigwedge_{i \in R} x_i = 1$.*[5]

PROOF. Let $R_0 \subseteq R$ denote the variables in $R$ that were *not* set to one at the initial sampling stage of the RELAXATION algorithm. Each such element $r \in R_0$

[5]Property (A4) is the only place in our paper where we are making essential use of the fact that the RELAXATION algorithm resamples the constraints in order, and only moves on to constraint $k+1$ when constraint $k$ has become satisfied. In fact, everything still works if the RELAXATION algorithm resamples constraints in an arbitrary order. It is even allowed to begin resampling constraint $k$, move on to constraint $k'$, and then go back to constraint $k$. We can show this using techniques in Chapter 4 to analyze the dynamic evolution of witness trees. We omit these complications from this chapter for simplicity.

must have become true during the $j_r$th resampling of some constraint $k_r$. Set $S = \{k_r \mid r \in R_0\}$ and set $w_s$ to be the maximum value of $j_r$, over all $r$ with $k_r = s$. We define $\mathcal{Z}$ by $Z_{s,j} = Y_{s,j}$ for $s \in S, j = 1, \ldots, w_s$.

To show property (A1): We define $R'$ by selecting, for each $s \in S$, exactly one element $r \in R_0$ with $j_r = w_s, k_r = s$. (There is always at least one such element $r$; if there are multiple, we select one arbitrarily.) Define the function $f : R' \to S$ by mapping each such $k_r$ to the corresponding $s \in S$.

Property (A2) is clear from the way we have chosen $\mathcal{Z}$.

To see Property (A3), suppose at the initial sampling stage, we had $x_i = 0$ for some $i \in R$. Then a necessary condition for $x_i = 1$ is that it is set during during the $j'$th resampling of some constraint $k'$. Then necessarily $k' \in S$ and so there must exist some $r \in R'$ with $k_r = k', j_r \geq j'$ (possibly $r = i$). So the set $Y_{k',j'}$ will placed into $\mathcal{Z}$ and $i \in Y_{k',j'}$, which contradicts the hypotheses of (A3). A similar argument applies to show Property (A4). □

PROPOSITION 6.22. *Suppose we are given some $R \subseteq [n]$, and a list $S, w, \mathcal{Z}$ satisfying property (A1) of Proposition 6.21. Then the the probability that the RELAXATION algorithm satisfies Properties (A2) — (A4) is at most $\prod_{i \in R} p_i \prod_{s \in S} \prod_{j=1}^{w_s} f_s(Z_{s,j})$.*

PROOF. This is a generalization of Lemma 6.4. Properties (A3), (A4) specify, for each $i \in R$, exactly one specific time during the execution of the RELAXATION algorithm at which we set $x_i = 1$. These have probabilities exactly $\prod_{i \in R} p_i$. Also, Property (A2) specifies for each $s \in S$ the first $w_s$ sets which are resampled for constraint $s$; these contribute probabilities $\prod_{j=1}^{w_s} f_s(Z_{s,j})$. □

We can use this result to show a type of negative-correlation property for the variables at the end of the RELAXATION algorithm:

THEOREM 6.23. *If we have $\hat{x}_i \leq 1/\alpha$ for all $i \in [n]$, then for any subset $R \subseteq [n]$, the output of the RELAXATION algorithm satisfies*

$$P(\bigwedge_{i \in R} x_i = 1) \leq \prod_{i \in R} \rho_i$$

*where, for each $i \in [n]$, we define*

$$\rho_i = \alpha \hat{x}_i \Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a_k} e^{\sigma \alpha A_k \cdot \hat{x}} - 1}\Big)$$

PROOF. If the event $\bigwedge_{i \in R} x_i = 1$ occurs, then by Proposition 6.21, there must exist $S, w, \mathcal{Z}$ satisfying (A1) – (A4). To bound the probability of this event, we take a union bound over all $S, w, \mathcal{Z}$ satisfying (A1). The probability that any such $S, w, \mathcal{Z}$ satisfies (A2) – (A4) is at most $F(S, w, \mathcal{Z}) = \prod_{i \in R} p_i \prod_{s \in S} \prod_{j=1}^{v_s} f_s(Z_{s,j})$. Thus we can write

$$P(\bigwedge_{i \in R} x_i = 1) \leq \sum_{\substack{S, w, \mathcal{Z} \\ \text{satisfying } (A1)}} Q(S, w, \mathcal{Z})$$

We enumerate $S, w, \mathcal{Z}$ satisfying (A1) as follows. We begin by setting $S = \emptyset$. For each $i \in R$, we may either choose to do nothing, or we may place $i$ into $R'$ and we may select some integer $k_i \in [m] - S$ to place into $S$, along with some integer $w_i > 0$ and some choice of sets $Z_{i,1}, \ldots, Z_{i,w_i}$ with $i \in Z_{i,w_i}$.

If we place $k_i$ into $[m] - S$, then it multiplies the value of $F$ by $p_i \prod_{j=1}^{w_i} f_{k_i}(W_{i,j})$; if we do not choose any such $k_i$, it multiplies the value of $F$ by $p_i$.

Let $S_i$ denote the value of $S$ after stages $1, \ldots, i-1$, that is, $S = \{k_1, \ldots, k_{i-1}\}$. Enumerating the possible values for $k_i$ and $w_i$ gives:

$$\begin{aligned}\sum_{\substack{S,w,\mathcal{Z} \\ \text{satisfying } (A1)}} Q(S,w,\mathcal{Z}) &\leq \prod_{i\in R} p_i\Big(1 + \sum_{k_i\in[m]-S_i} \sum_{w_i>0} \sum_{\substack{Z_{i,1},\ldots,Z_{i,w_i} \\ i\in Z_{i,w_i}}} \prod_{j=1}^{w_i} f_{k_i}(Z_{i,j})\Big) \\ &\leq \prod_{i\in R} p_i\Big(1 + \sum_{k_i\in[m]} \sum_{w_i>0} \sum_{\substack{Z_{i,1},\ldots,Z_{i,w_i} \\ i\in Z_{i,w_i}}} \prod_{j=1}^{w_i} f_{k_i}(Z_{i,j})\Big) \\ &= \prod_{i\in R}\Big(p_i(1 + \sum_{k} \sum_{j>0} \sum_{\substack{Z_1,\ldots,Z_j \\ i\in Z_j}} f_k(Z_1)\ldots f_k(Z_j))\Big)\end{aligned}$$

We have already seen the term $p_i\big(1 + \sum_k \sum_{j>0} \sum_{\substack{Z_1,\ldots,Z_j \\ i\in Z_j}} f_k(Z_1)\ldots f_k(Z_j)\big)$: in Theorem 6.7 it was shown that it is at most $\rho_i$. Thus we have $\sum Q \leq \prod_{i\in R} \rho_i$ and the theorem is proved. □

We can now show a concentration phenomenon for $C \cdot x$. In order to obtain the simplest such bounds, we can make an assumption that the entries of $C$ are in the range $[0,1]$. In this case, we can use the Chernoff upper-tail function to give estimates for the concentration of $C \cdot x$. We recall the definition of the Chernoff function:

Definition 6.24 (The Chernoff separation function for upper-tail bounds). *For $t \geq \mu$ with $\delta = \delta(\mu,t) = t/\mu - 1 \geq 0$, the Chernoff upper-tail bound is defined as*

$$\textit{Chernoff-}U(\mu,t) = \Big(\frac{e^\delta}{(1+\delta)^{1+\delta}}\Big)^\mu \tag{32}$$

*That is to say Chernoff-$U(\mu,t)$ is the Chernoff bound that a sum of $[0,1]$-bounded and independent random variables with mean $\mu$ will be above $t$.*

Corollary 6.25. *Suppose that all entries of $C_l$ are in the interval $[0,1]$ and that all entries $\hat{x}$ satisfy $\hat{x}_i \leq 1/\alpha$. Then, after running the RELAXATION algorithm, the probability of the event $C_l \cdot x > t$ is at most Chernoff-$U(C_l \cdot \rho, t)$.*

Proof. The value of $C_l \cdot x$ is a sum of random variables $C_{li}x_i$ which are in the range $[0,1]$. These random variables obey a negative-correlation property as shown in

Theorem 6.23. This implies that they obey the same upper-tail Chernoff bounds as would a sum of random variables $X_i$ which are *independent* and satisfy $\mathbf{E}[X_i] = \rho_i$. $\square$

We next need to show concentration for the ROUNDING algorithm.

THEOREM 6.26. *Suppose that all entries of $C_l$ are in $[0,1]$. Then, after the ROUNDING algorithm, the probability of the event $C_l \cdot x > t$ is at most Chernoff-U($C_l \cdot \rho, t$).*

PROOF. Let $v_i, G_i, \hat{x}'_i, a'_k, x'$ be the variables which occur during the ROUNDING algorithm. We have

$$\begin{aligned}
P(C_l \cdot x > t) &= P(C_l \cdot (v\theta + G + x') > t) \\
&= P(C_l \cdot x' > t - C_l \cdot (v\theta + G)) \\
&\leq \text{Chernoff-U}(C_l \cdot \rho', t - C_l \cdot (v\theta + G)) \\
&= \text{Chernoff-U}\Big(\alpha \sum_i C_{li} \hat{x}'_i \big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a'_k} e^{\sigma\alpha A_k \cdot \hat{x}'} - 1}\big), t - C_l \cdot (v\theta + G)\Big) \qquad (33)
\end{aligned}$$

By Proposition 6.31, Chernoff-U$(\mu, t)$ is always an increasing function of $\mu$. So we can show an upper bound for this expression by giving an upper bound for the $\mu$ term in the (33). We first apply Propositions 6.9, 6.11 which give:

$$\hat{x}'_i \Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a'_k} e^{\sigma\alpha A_k \cdot \hat{x}'} - 1}\Big) \leq (\hat{x}_i - v_i\theta - G_i/\alpha)\Big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a_k} e^{\sigma\alpha a_k} - 1}\Big)$$

Substituting this upper bound into (33) yields:

$$\begin{aligned}
P(C_l \cdot x > t) \leq \text{Chernoff-U}\Big(&\alpha \sum_i C_{li} (\hat{x}_i - v_i\theta - G_i/\alpha)\big(1 + \sigma \sum_k \frac{A_{ki}}{(1-\sigma)^{a_k} e^{\sigma\alpha a_k} - 1}\big), \\
&t - C_l \cdot (v\theta + G)\Big) \\
\leq \text{Chernoff-U}\Big(&\sum_i C_{li} (\rho_i - (v_i\alpha\theta + G_i)), t - C_l \cdot (v\theta + G)\Big) \\
\leq \text{Chernoff-U}\Big(&(C_l \cdot \rho) - (C_l \cdot (v\theta + G)), t - (C_l \cdot (v\theta + G))\Big) \\
\leq \text{Chernoff-U}(&C_l \cdot \rho, t) \qquad \text{by Proposition 6.32}
\end{aligned}$$

□

In the column-sparsity setting, we obtain the following result which extends Theorem 6.14:

COROLLARY 6.27. *Suppose we are given a covering system as well as a fractional solution $\hat{x}$. Let $\gamma = \frac{\log(\Delta_1+1)}{a_{min}}$. Suppose that the entries of $C_l$ are in $[0,1]$. Then, with an appropriate choice of $\sigma, \alpha$ we may run the ROUNDING algorithm in expected time $O(mn)$ to obtain a solution $x \in \mathbf{Z}_+^n$ such that*

$$P(C_l \cdot x > t) \leq \textit{Chernoff-U}(\beta C_l \cdot \hat{x}, t)$$

*for $\beta = 1 + \gamma + 4\sqrt{\gamma}$.*

*If one wishes to ensure also that $x_i \leq \lceil \hat{x}_i(1+\epsilon) \rceil$ for $\epsilon \in (0,1)$, then one can obtain a similar result with an approximation factor $\beta = 1 + 4\sqrt{\gamma} + 4\gamma/\epsilon$.*

### 6.A. Some technical lemmas

PROPOSITION 6.28. *Given set $S$, $x_i \in [0,1]$, and $a \in (0,1)$, we have*

$$\prod_{i \in S} (1 - a x_i)^{-1} \leq (1-a)^{-\sum_{i \in S} x_i}$$

PROOF. By compactness, $\prod_{i \in S}(1-ax_i)^{-1}$ attains a maximum on the hyperplane defined by $\sum_{i \in S} x_i = s$. We claim that this maximum occurs when at most one $x_i$ is fractional. To show this, suppose that for contradiction there two variables with fractional values. Without loss of generality, say $0 < x_1 \leq x_2 < 1$. Suppose we decrease $x_1$ by $\delta$ and increase $x_2$ by $\delta$, for $\delta > 0$ sufficiently small. We then have

$$\begin{aligned} \frac{\prod_{i \in S}(1-ax_i')^{-1}}{\prod_{i \in S}(1-ax_i)^{-1}} &= \frac{(1-ax_1)(1-ax_2)}{(1-a(x_1-\delta))(1-a(x_2+\delta))} \\ &= \frac{(1-ax_1)(1-ax_2)}{(1-ax_1)(1-ax_2) - a^2\delta(\delta - x_1 + x_2)} \end{aligned}$$

As $x_2 \leq x_1$, the term $a^2\delta(\delta - x_1 + x_2)$ is positive, and hence this is $> 1$, which contradicts the maximality of $x$.

Thus, giving a fixed value for $\sum_{i\in S} x_i = s$, the maximum value of $\prod_{i\in S}(1-ax_i)^{-1}$ occurs when one variable is fractional and the rest are equal to one. So suppose $s = z + r$ where $z \in \mathbf{Z}_+$ and $r \in (0,1)$. Substituting this into the product and an application of Bernoulli's inequality, gives:

$$\prod_{i\in S}(1-ax_i)^{-1} = (1-a)^{-z}(1-ra)^{-1} \leq (1-a)^{-z}(1-a)^{-r} = (1-a)^{-\sum_{i\in S} x_i}$$

□

PROPOSITION 6.29. *Suppose $x \in [0,1], y \in [0,1]$. Then we have $(1-x)e^{yx} \leq 1$.*

PROOF. We note that the expression $(1-x)e^{yx}$ is monotonically increasing with respect to $y$ on the domain $y \in [0,1]$. Thus we have $(1-x)e^{yx} \leq (1-x)e^{x}$. This then reduces to the well-known inequality $1 - x \leq e^{-x}$. □

PROPOSITION 6.30. *For any $\gamma > 0$, define*

$$f(x) = \frac{x}{(x+1)^{\frac{2\sqrt{\gamma}+\gamma-2\ln(1+\sqrt{\gamma})}{\gamma}} - 1}$$

*Then $f(x)$ is a decreasing function of $x$ for $x > 0$.*

PROOF. At $x = 0$, both numerator and denominator are equal to zero. So it suffices to show that the denominator grows faster than the numerator, that is, that the derivative of the denominator is always $\geq 1$. We compute the derivative of the denominator:

$$\begin{aligned} R &= \frac{\left(\gamma + 2\sqrt{\gamma} - 2\ln\left(\sqrt{\gamma}+1\right)\right)(x+1)^{\frac{2\sqrt{\gamma}-2\ln(1+\sqrt{\gamma})}{\gamma}}}{\gamma} \\ &\geq \frac{(\gamma+0)(x+1)^{\frac{0}{\gamma}}}{\gamma} \qquad \text{as } y \geq \ln(1+y) \text{ for } y > 0 \\ &= 1 \end{aligned}$$

as desired.

□

PROPOSITION 6.31. *For any* $0 \leq \mu \leq \mu' \leq t$ *we have Chernoff-*$U(\mu, t) \leq$ *Chernoff-*$U(\mu', t)$.

PROOF. Note that the Chernoff separation function for upper-tailed bounds is monotonically increasing on the range $[0, t)$ with respect to $\mu$ thus our assumption $0 \leq \mu \leq \mu' \leq t$ necessarily implies Chernoff-U$(\mu, t) \leq$ Chernoff-U$(\mu', t)$. □

PROPOSITION 6.32. *For any* $0 \leq \mu \leq t$ *and any* $r \leq \mu$, *we have Chernoff-*$U(\mu, t) \leq$ *Chernoff-*$U(\mu - r, t - r)$.

PROOF. The directional derivative of Chernoff-U$(\mu, t)$ along the unit vector $\hat{u} = (1, 1)$ is

$$e^{t-\mu}\mu^{t-1}t^{-t}(t - \mu + \mu(\log \frac{\mu}{t}))$$

Setting $t = \mu(1 + \delta)$, this is equal to $e^{t-\mu}\mu^t t^{-t}(\delta - \log(1 + \delta))$, which is positive for $\delta \geq 0$. Thus, Chernoff-U$(\mu, t)$ is decreasing along the direction $(-1, -1)$, i.e. Chernoff-U$(\mu - r, t - r) \leq$ Chernoff-U$(\mu, t)$. □

PROPOSITION 6.33. *For any* $g \in (0, 1)$ *we have*

(34)
$$1 + \frac{2g\big(-2 + \ln(4 - 3g) - \ln(8 - 3g) - \ln(g + 4) + \ln(g + 8) - 2\ln 2\big)}{4g \ln 2 - (4 - 3g)\ln(4 - 3g) + (8 - 3g)\ln(8 - 3g) + (4 + g)\ln(4 + g) - (8 + g)\ln(8 + g)} < 14/g$$

PROOF. Let us first consider the denominator in this expression, $f(g) = 4g \ln 2 - (4 - 3g)\ln(4 - 3g) + (8 - 3g)\ln(8 - 3g) + (4 + g)\ln(4 + g) - (8 + g)\ln(8 + g)$. Note that $f''(g)$ is a rational function, and simple algebra shows that its only root is at $g = -16/9$. As $f''(0) = -1$, this implies that $f''(g) < 0$ for all $g \in (0, 1)$. Thus, $f'(g)$ is decreasing in this range. As $f'(0) = 0$, this implies that $f'(g) < 0$ for $g \in (0, 1)$. As $f(0) = 0$, this further implies that $f(g) < 0$ for $g \in (0, 1)$.

We may thus cross-multiply (34), taking into account the fact that the denominator is negative. Thus to show (34) is suffices to show that $h(g) < 0$, where we define

$$h(g) = (-5g^2 + 46g - 56)\ln(4 - 3g) + (5g^2 - 50g + 112)\ln(8 - 3g)$$
$$+ (g^2 + 10g + 56)\ln(g + 4) + (-g^2 - 6g - 112)\ln(g + 8) + 4g^2 + 56g \ln 2$$

Simple calculus shows that $h'''(g)$ is a rational function of $g$, and it has no roots in the range $(0, 1)$. As $h'''(0) = -75/8$, this implies that $h'''(g) < 0$ for all $g \in (0, 1)$. As $h''(0) = -0.454$, this implies that $h''(g) < 0$ for all $g \in (0, 1)$. As $h'(0) = 0$, this implies that $h'(g) < 0$ for all $g \in (0, 1)$. As $h(0) = 0$, this implies that $h(g) < 0$ for all $g \in (0, 1)$. $\square$

CHAPTER 7

# A constructive algorithm for the LLL on permutations

One major relative of the LLL that has eluded constructive versions, is the "lopsided" version of the LLL (with the single exception of the varible-assignment setting). A natural setting for the lopsided LLL is the probability space defined by random permutations, which has been used for for Latin transversals [**17, 35, 110**], hypergraph packing [**77**], certain types of graph coloring [**20**], and in proving the existence of certain error-correcting codes [**64**]. However, the MT framework does not give constructive algorithms in these cases. In this chapter, develop a randomized polynomial-time algorithm to construct such permutation(s) whose existence is guaranteed by the lopsided LLL, leading to several algorithmic applications in combinatorics.

Suppose we define a probability space $\Omega$ by selecting the permutations $\pi_1, \ldots, \pi_N$ uniformly at random where each $\pi_k$ is a permutation on the set $[n_k] = \{1, \ldots, n_k\}$. We have seen in Proposition 1.6 that a dependency graph can be defined among atomic events on permutations.

Thus, we suppose that that the set of bad events $\mathcal{B}$ consists of *atomic bad-events*. We identify each $B \in \mathcal{B}$ with a set of tuples $B = \{(k_1, x_1, y_1), \ldots, (k_r, x_r, y_r)\}$; $B$ is true iff we have $(\pi_{k_1}(x_1) = y_1) \wedge \cdots \wedge (\pi_{k_r}(x_r) = y_r)$. (Complex bad-events can usually be decomposed into atomic bad-events, so this does not lose much generality).

We recall from Proposition 1.6 how the LLL applies to such events. We define the dependency graph among such events, by $B \sim B'$ if $B, B'$ overlap in one slice of the domain or range of a permutation; namely, that there are some $k, x, y_1, y_2$ with $(k, x, y_1) \in B, (k, x, y_2) \in B'$ *or* there are some $k, x_1, x_2, y$ with $(k, x_1, y) \in B, (k, x_2, y) \in B'$. Note that $B \sim B$ in this definition.

The following notation will be useful: for pairs $(x_1, y_1), (x_2, y_2)$, we write $(x_1, y_1) \sim (x_2, y_2)$ if $x_1 = x_2$ *or* $y_1 = y_2$ (or both). Another way to write $B \sim B'$ is that "there are $(k, x, y) \in B$, $(k, x', y') \in B'$ with $(x, y) \sim (x', y')$".

We will use the following notation at various points: we write $(k, x, *)$ to mean any (or all) triples of the form $(k, x, y)$, and similarly for $(k, *, y)$, or $(x, *)$ etc. Another way to write the condition $B \sim B'$ is that there are $(k, x, *) \in B, (k, x, *) \in B'$ or $(k, *, y) \in B, (k, *, y) \in B'$.

Now, as discussed in Chapter 1, the LLLL applies to the resulting probability space $\Omega$. As is standard for the LLLL, this only results in an exponentially small probability of selecting the desired permutations.

As mentioned above, a variety of papers have used this framework for proving the existence of various combinatorial structures. Unfortunately, the MT algorithm does not apply in this setting. The problem is that such algorithms have a more restrictive notion of when two bad-events are dependent; namely, that they share variables. So we do not have an algorithm to generate such permutations, we can merely show that they exist.

We develop an algorithmic analog of the LLL for permutations. The necessary conditions for our Swapping Algorithm are the same as for the probabilistic LLLL; however, we will construct such permutations in randomized polynomial (typically linear or near-linear) time. Our setting is far more complex than in similar contexts and requires many intermediate results first. The main complication is that when we encounter a bad event involving "$\pi_k(x) = y$", and we perform our algorithm's random swap associated with it, we could potentially be changing any entry of $\pi_k$. This is in marked contrast to the standard MT setting. We also develop RNC versions of our algorithms. Going from serial to parallel is fairly direct in [**87**]; our main bottleneck here is that when we resample an "independent" set of bad events, they could still influence each other.

**7.0.1. Applications.** We present algorithmic applications for four classical combinatorial problems: Latin transversals, rainbow Hamiltonian cycles, strong chromatic number, and edge-disjoint hypergraph packing. In addition to the improved bounds, we wish to highlight that our algorithmic approach can go beyond the LLLL: as we will see shortly, one of our (asymptotically-optimal) algorithmic results on Latin transversals, could not even have been shown nonconstructively using the lopsided LLL prior to this work.

The study of Latin squares and the closely-related Latin transversals is a classical area of combinatorics, going back to Euler and earlier [**30**]. Given an $m \times n$ matrix $A$ with $m \leq n$, a *transversal* of $A$ is a choice of $m$ elements from $A$, one from each row and at most one from any column. Perhaps the major open problem here is: given an integer $s$, under what conditions will $A$ have an *$s$-transversal*: a transversal in which no value appears more than $s$ times [**17, 33, 35, 103, 109, 110**]? The usual type of sufficient condition sought here is an upper bound $\Delta$ on the number of occurrences of any given value in $A$. That is, we ask: what is the maximum $\Delta = \Delta(s; m, n)$ such that any $m \times n$ matrix $A$ in which each value appears at most $\Delta$ times, is guaranteed to have an $s$-transversal? The case $s = 1$ is perhaps most studied, and 1-transversals are also called *Latin transversals*. The case $m = n$ is also commonly studied (and includes Latin squares as a special case), and we will also focus on these. It is well-known that $L(1; n, n) \leq n - 1$ [**109**]. In perhaps the first application of the lopsided LLL to random permutations, Erdős & Spencer showed that $L(1; n, n) \geq n/(4e)$ [**35**]. (Their paper shows that $L(1; n, n) \geq n/16$; the $n/(4e)$ lower-bound follows easily from their technique.) To our knowledge, this is the first $\Omega(n)$ lower-bound on $L(1; n, n)$. Alon asked if there is a constructive version of this result [**6**]. Building on [**35**] and using the connections to the LLL from [**101, 102**], Bissacot *et al.* showed nonconstructively that $L(1; n, n) \geq (27/256)n$ [**17**]. Our result makes this constructive.

The lopsided LLL has also been used to study the case $s > 1$ [**110**]. Here, we prove a result that is asymptotically optimal for large $s$, except for the lower-order $O(\sqrt{s})$

term: we show (algorithmically) that $L(s; n, n) \geq (s - O(\sqrt{s})) \cdot n$. An interesting fact is that this was not known even nonconstructively before: the LLLL roughly gives $L(s; n, n) \geq (s/e) \cdot n$. We also give faster serial and perhaps the first RNC algorithms with good bounds, for the strong chromatic number. Strong coloring is quite well-studied [**7, 13, 39, 55, 56**], and is in turn useful in *covering* a matrix with Latin transversals [**12**].

**7.0.2. Comparison with other LLLL algorithms.** Building on an earlier version of this work, there have been two papers which have developed generic frameworks for variations of the MT algorithm applied to other probability spaces. In [**1**], Achlioptas & Iliopoulos gave an algorithm which is based on a compression analysis for a random walk; this was improved for permutations and matchings by Kolmogorov [**68**]. In [**51**], Harvey & Vondrak gave a probabilistic analysis similar to the parallel MT algorithm. These frameworks both include the permutation LLL as well as some other combinatorial applications.

These papers both follow a proof strategy which analyzes the *full* permutation as it evolves over time, giving a far simpler proof that the Swapping Algorithm terminates quickly. However, this appears unable to prove the Witness Tree Lemma, Lemma 7.1, which provides far more precise estimates on the behavior of *parts* of the permutations.

The Witness Tree Lemma is at the heart of the analysis of the algorithm of Moser & Tardos, and it is this lemma which appears to be necessary for many of its extensions. These do not appear possible to derive from either [**1**] or [**51**]. See Section 7.5.3 for more details.

**7.0.3. Outline.** In Section 7.1 we introduce our Swapping Algorithm, a variant of the MT algorithm. In it, we randomly select our initial permutations; as long as some bad-event is currently true, we perform certain random swaps to randomize (or resample) them.

In Section 7.2, we introduce the key analytic tools to understand the behavior of the Swapping Algorithm, namely the witness tree and the witness subdag. The

witness subdag is a related concept, which is new here; it provides a history not for each resampling, but for each individual swapping operation performed during the resamplings. (It is more "granular" than the witness dags introduced in Chapter 3.)

In Section 7.3, we show how these witness subdags may be used to deduce partial information about the permutations. As the Swapping Algorithm proceeds in time, the witness subdags can also be considered to evolve over time. At each stage of this process, the current value of the witness subdags provides information about the current values of the permutations.

In Section 7.4, we use this process to make probabilistic predictions for certain swaps made by the Swapping Algorithm: namely, whenever the witness subdags change, the swaps must be highly constrained so that the permutations still conform to them. We calculate the probability that the swaps satisfy these constraints.

In Section 7.5, we put the analyses of Sections 7.2, 7.3, 7.4 together, to prove that our Swapping Algorithm terminates in polynomial time under the same conditions as the probabilistic LLL for permutations.

In Section 7.6, we introduce a parallel (RNC) algorithm corresponding to the Swapping Algorithm. This is similar in spirit to the Parallel MT algorithm. In the latter algorithm, one repeatedly selects a maximal independent set (MIS) of bad-events which are currently true, and resamples them in parallel. In our setting, bad-events which are "independent" in the LLL sense (that is, they are not connected via $\sim$), may still influence each other; a great deal of care must be taken to avoid these conflicts.

In Section 7.7, we describe a variety of combinatorial problems to which our Swapping Algorithm can be applied.

## 7.1. The Swapping Algorithm

We will analyze the following *Swapping Algorithm* algorithm to find a satisfactory $\pi_1, \ldots, \pi_N$:

(1) Generate the permutations $\pi_1, \ldots, \pi_N$ uniformly at random and independently.

(2) While there is some true bad-event:

(3) Choose some true bad-event $B \in \mathcal{B}$ arbitrarily. For each permutation that is involved in $B$, we perform a *swapping* of all the relevant entries. (We will describe the swapping subroutine "Swap" shortly.) We refer to this step as a *resampling* of the bad-event $B$.

Each permutation involved in $B$ is swapped independently, but if $B$ involves multiple entries from a single permutation, then all such entries are swapped *simultaneously*. For example, if $B$ consisted of triples $(k_1, x_1, y_1), (k_2, x_2, y_2), (k_2, x_3, y_3)$, then we would perform $\text{Swap}(\pi_1; x_1)$ and $\text{Swap}(\pi_2; x_2, x_3)$, where the "Swap" procedure is given next.

The swapping subroutine $\text{Swap}(\pi; x_1, \ldots, x_r)$ for a permutation $\pi : [t] \to [t]$ as follows:

Repeat the following for $i = 1, \ldots, r$:

- Select $x'_i$ uniformly at random among $[t] - \{x_1, \ldots, x_{i-1}\}$.
- Swap entries $x_i$ and $x'_i$ of $\pi$.

Note that at every stage of this algorithm all the $\pi_k$ are permutations, and if this algorithm terminates, then the $\pi_k$ must avoid all the bad-events. So our task will be to show that the algorithm terminates in polynomial time. We measure time in terms of a single iteration of the main loop of the Swapping Algorithm: each time we run one such iteration, we increment the time by one. We will use the notation $\pi_k^T$ to denote the value of permutation $\pi_k$ after time $T$. The initial sampling of the permutation (after Step (1)) generates $\pi_k^0$.

The swapping subroutine seems strange; it would appear more natural to allow $x'_i$ to be uniformly selected among $[t]$. However, the swapping subroutine is nothing more than than the Fisher-Yates Shuffle for generating uniformly-random

permutations. If we alloved $x'_i$ to be chosen from $[t]$ then the resulting permutation would be biased. The goal is to change $\pi_k$ in the minimal way to ensure that $\pi_k(x_1), \ldots, \pi_k(x_r), \pi_k^{-1}(y_1), \ldots, \pi_k^{-1}(y_r)$ are adequately randomized.

There are alternative methods for generating random permutations, and many of these can replace the Swapping subroutine without changing our analysis. We discuss a variety of such equivalencies in Appendix 7.A; these will be used in various parts of our proofs. We note that one class of algorithms that has a very different behavior is the commonly used method to generate random reals $r_i \in [0, 1]$, and then form the permutation by sorting these reals. When encountering a bad-event, one would resample the affected reals $r_i$. In our setting, where the bad-events are defined in terms of specific values of the permutation, this is not a good swapping method because a single swap can drastically change the permutation. When bad-events are defined in terms of the relative *rankings* of the permutation (e.g. a bad event is $\pi(x_1) < \pi(x_2) < \pi(x_3)$), then this is a better method and can be analyzed in the framework of the ordinary MT algorithm.

## 7.2. Witness trees and witness subdags

We analyze the Swapping Algorithm following the MT approach [**87**] of execution logs and witness trees. These witness trees are produced in the same manner as for the ordinary MT algorithm. For the remainder of this section, the dependence on the "justified" bad-event at time $t$ at the root of the tree will be understood; we will omit it from the notation.

The critical lemma that allows us to analyze the behavior of this algorithm is the *Witness Tree Lemma*:

LEMMA 7.1 (Witness Tree Lemma). *Let $\tau$ be a tree-structure, with nodes labeled $B_1, \ldots, B_s$. The probability that $\tau$ appears, is at most*

$$P(\tau \text{ appears}) \leq P_\Omega(B_1) \cdots P_\Omega(B_s)$$

*Note that the probability of the event $B$ within the space $\Omega$ can be computed as follows: if $B$ contains $r_1, \ldots, r_N$ elements from each of the permutations $1, \ldots, N$, then we have*

$$P_\Omega(B) = \frac{(n_1 - r_1)!}{n_1!} \cdots \frac{(n_N - r_N)!}{n_N!}$$

This lemma is superficially similar to the corresponding Lemma 2.8. However, the proof will be far more complex, and we will require many intermediate results first. The main complication is that when we encounter a bad-event involving $\pi_k(x) = y$, and we perform the random swap associated with it, then we could potentially be changing any entry of $\pi_k$. By contrast, in the usual MT algorithm, when we resample a variable, all the changes are confined to that variable. However, as we will see, the witness tree will leave us with enough clues about which swap was actually performed that we will be able to narrow down the possible impact of the swap.

The analysis in the next sections can be very complicated. We have two recommendations to make these proofs easier. First, the basic idea behind how to form and analyze these trees comes from [**87**]; the reader should consult that paper for results and examples which we omit here. Second, one can get most of the intuition behind these proofs by considering the situation in which there is a single permutation, and the bad-events all involve just a single element; that is, every bad-event has the form $\pi(x_i) = y_i$. In this case, the witness subdags (defined later) are more or less equivalent to the witness tree. (The main point of the witness subdag concept is, in effect, to reduce bad-events to their individual elements.) When reading the following proofs, it is a good idea to keep this special case in mind. In several places, we will discuss how certain results simplify in that setting.

The following proposition is the main reason the witness tree encodes sufficient information about the sequence of swaps:

PROPOSITION 7.2. *Suppose that at some time $t_0$ we have $\pi_k^{t_0}(X) \neq Y$, and at some later time $t_2 > t_0$ we have $\pi_k^{t_2}(X) = Y$. Then there must have occured at some intermediate time $t_1$ some bad-event including $(k, X, *)$ or $(k, *, Y)$.*

PROOF. Let $t_1 \in [t_0, t_2 - 1]'$ denote the earliest time at which we had $\pi^{t_1+1}(X) = Y$; this must be due to resampling some bad-event which includes $(k, x_1, y_1), \ldots, (k, x_r, y_r)$ (and possibly other elements from other permutations). Suppose that the swap which first caused $\pi(X) = Y$ was at swapping entry $x_i$, which at that time had $\pi_k(x_i) = y_i'$, with some $x''$.

After this swap, we have $\pi_k(x_i) = y''$ and $\pi_k(x'') = y_i'$. Evidently $x'' = X$ or $x_i = X$. In the second case, the bad event at time $t_1$ included $(k, X, *)$ as desired and we are done.

So suppose $x'' = X$ and $y_i' = Y$. So at the time of the swap, we had $\pi_k(x_i) = Y$. The only earlier swaps in this resampling were with $x_1, \ldots, x_{i-1}$; so at the beginning of time $t_1$, we must have had $\pi_k^{t_1}(x_j) = Y$ for some $j \leq i$. This implies that $y_j = Y$, so that the bad-event at time $t_1$ included $(k, *, Y)$ as desired. □

To explain some of the intuition behind why Proposition 7.2 implies Lemma 7.1, consider the case of a *singleton* witness tree.

COROLLARY 7.3. *Suppose that $\tau$ is a singleton node labeled by $B$. Then $P(\tau \text{ appears}) \leq P_\Omega(B)$.*

PROOF. Suppose $\hat{\tau}^T = \tau$. We claim that $B$ must have been true of the initial configuration. For suppose that $(k, x, y) \in B$ but in the initial configuration we have $\pi_k(x) \neq y$. At some later point in time $t \leq T$, the event $B$ must become true. By Proposition 7.2, then there is some time $t' < t$ at which we encounter a bad-event $B'$ including $(k, x, *)$ or $(k, *, y)$. This bad-event $B'$ occurs earlier than $B$, and $B' \sim B$. Hence, we would have placed $B'$ below $B$ in the witness tree $\hat{\tau}^T$. □

In proving Lemma 7.1, we will *not* need to analyze the interactions between the separate permutations, but rather we will be able to handle each permutation in a completely independent way. For a permutation $\pi_k$, we define the *witness subdag for permutation* $\pi_k$; this is a relative of the witness tree, but which only includes the information for a single permutation at a time.

DEFINITION 7.4 (witness subdags). *For a permutation $\pi_k$, a* witness subdag for $\pi_k$ *is defined to be a directed acyclic simple graph, whose nodes are labeled with pairs of the form $(x, y)$. If a node $v$ is labeled by $(x, y)$, we write $v \approx (x, y)$. This graph must in addition satisfy the following properties:*

(1) *If any pair of nodes overlaps in a coordinate, that is, we have $v \approx (x, y) \sim (x', y') \approx v'$, then nodes $v, v'$ must be comparable (that is, either there is a path from $v$ to $v'$ or vice-versa).*

(2) *Every node of $G$ has in-degree at most two and out-degree at most two.*

*We also may label the nodes with some auxiliary information, for example we will record that the nodes of a witness subdag correspond to bad-events or nodes in a witness tree $\tau$.*

*We will use the same terminology as for witness trees: vertices on the "bottom" are close to the source nodes of $G$ (appearing earliest in time), and vertices on the "top" are close to the sink nodes of $G$ (appear latest in time).*

The witness subdags that we will be interested in are derived from witness trees in the following manner.

DEFINITION 7.5 (Projection of a witness tree). *For a tree-structure $\tau$, we define the* projection of $\tau$ onto permutation $\pi_k$ *which we denote $Proj_k(\tau)$, as follows.*

*Suppose we have a node $v \in \tau$ which is labeled by some bad-event defined by $(k_1, x_1, y_1), \ldots, (k_r, x_r, y_r)$. For each $i$ with $k_i = k$, we create a corresponding node $v'_i \approx (x_i, y_i)$ in the graph $Proj_k(\tau)$. We also include some auxiliary information indicating that these nodes came from $v$, and in particular that all such nodes are part of the same bad-event.*

*We add edges to $Proj_k(\tau)$ as follows. For each node $v' \in Proj_k(\tau)$, labeled by $(x, y)$ and corresponding to $v \in \tau$, we examine all occurrences of nodes labeled $(k, x, *)$. All such occurrences are comparable; we find the node $u'$ corresponding to $u \in \tau$ which is closest to the source. In other words, we find the occurrence $u'$ which appears*

*"next-highest" in the dag. We create an edge from $v'$ to $u'$. Similarly, we find the next-highest occurrence $w \in \tau$ of a bad-event labeled by $(k, *, y)$; we create an edge from $v'$ to $w'$.*

*Note that $u, w$ must appear* strictly *higher in $\tau$, because of the way new bad-events are added to witness trees. This implies that $Proj_k(\tau)$ is acyclic. Also, note that it is possible that $u = w$; in this case we only add a single edge to $Proj_k(\tau)$.*

***Expository Remark:*** *In the special case when each bad-event contains a single element, the witness subdag is a "flattening" of the tree structure. Each node in the tree corresponds to a node in the witness subdag, and each node in the witness subdag points to the next highest occurrence of the domain and range variables.*

Basically, the projection of $\tau$ onto $k$ tells us all of the swaps of $\pi_k$ that occur. It also gives us some of the temporal information about these swaps that would have been available from $\tau$. If there is a path from $v$ to $v'$ in $\text{Proj}_k(\tau)$, then we know that the swap corresponding to $v$ must come before the swap corresponding to $v'$. It is possible that there are a pair of nodes in $\text{Proj}_k(\tau)$ which are incomparable, yet in $\tau$ there was enough information to deduce which event came first (because the nodes would have been connected through some other permutation). So $\text{Proj}_k(\tau)$ does discard some information from $\tau$, but it turns out that we will not need this information.

To prove Lemma 7.1, we will prove (almost) the following claim: Let $G$ be a witness subdag for permutation $\pi_k$; suppose the nodes of $G$ are labeled with bad-events $B_1, \ldots, B_s$. Then the probability that there is some $T > 0$ such that $G = \text{Proj}_k(\hat{\tau}^T)$, is at most

$$P(G = \text{Proj}_k(\hat{\tau}^T) \text{ for some } T > 0) \leq P_k(B_1) \cdots P_k(B_s) \tag{35}$$

where, for a bad-event $B$ we define $P_k(B)$ in a similar manner to $P_\Omega(B)$; namely that if the bad-event $B$ contains $r_k$ elements from permutation $k$, then we define $P_k(B) = \frac{(n_k - r_k)!}{n_k!}$.

Unfortunately, proving this directly runs into technical complications regarding the order of conditioning. It is simpler to just sidestep these issues. However, the reader should bear in mind (35) as the *informal* motivation for the analysis in Section 7.3.

## 7.3. The conditions on a permutation $\pi_{k^*}$ over time

In Section 7.3, we will fix a value $k^*$, and we will describe conditions that $\pi^t_{k*}$ must satisfy at various times $t$ during the execution of the Swapping Algorithm. *In this section, we are only analyzing a single permutation $k^*$. To simplify notation, the dependence on $k^*$ will be hidden henceforth; we will discuss simply $\pi, Proj(\tau)$, and so forth.*

This analysis can be divided into three phases.

(1) We define the *future-subgraph* at time $t$, denoted $G_t$. This is a kind of graph which encodes necessary conditions on $\pi^t$, in order for $\tau$ to appear, that is, for $\hat{\tau}^T = \tau$ for some $T > 0$. Importantly, these conditions, and $G_t$ itself, are independent of the precise value of $T$. We define and describe some structural properties of these graphs.

(2) We analyze how a future-subgraph $G_t$ imposes conditions on the corresponding permutation $\pi^t$, and how these conditions change over time.

(3) We compute the probability that the swapping satisfies these conditions.

We will prove (1) and (2) in Section 7.3. In Section 7.4 we will put this together to prove (3) for all the permutations.

**7.3.1. The future-subgraph.** Suppose we have fixed a target graph $G$, which could hypothetically have been produced as the projection of $\hat{\tau}^T$ onto $k*$. We begin the execution of the Swapping Algorithm and see if, so far, it is still possible that $G = \text{Proj}_{k^*}(\hat{\tau}^T)$, or if $G$ has been disqualified somehow. Suppose we are at time $t$ of this process; we will show that certain swaps must have already occurred at past times $t' < t$, and certain other swaps must occur at future times $t' > t$.

We define the *future-subgraph* of $G$ at time $t$, denoted $G_t$, which tells us all the future swaps that must occur.

DEFINITION 7.6 (The future-subgraph). *We define the future-subgraphs $G_t$ inductively. Initially $G_0 = G$. When we run the Swapping Algorithm, as we encounter a bad-event $(k_1, x_1, y_1), \ldots, (k_r, x_r, y_r)$ at time $t$, we form $G_{t+1}$ from $G_t$ as follows:*

(1) *Suppose that $k_i = k^*$, and $G_t$ contains a source node $v$ labeled $(x_i, y_i)$. Then $G_{t+1} = G_t - v$.*

(2) *Suppose that $k_i = k^*$, and $G_t$ has a source labeled $(x_i, y'')$ where $y'' \neq y_i$ or $(x'', y_i)$ where $x'' \neq x_i$. Then, as will be shown in shown in Proposition 7.7, we can immediately conclude $G$ is impossible; we set $G_{t+1} = \perp$, and we can abort the execution of the Swapping Algorithm.*

(3) *Otherwise, we set $G_{t+1} = G_t$.*

PROPOSITION 7.7. *For any time $t \geq 0$, let $\hat{\tau}^T_{\geq t}$ denote the witness tree built for the event at time $T$, but only using the execution log from time $t$ onwards. Then if $Proj(\hat{\tau}^T) = G$ we also have $Proj(\hat{\tau}^T_{\geq t}) = G_t$.*

*Note that if $G_t = \perp$, the latter condition is obviously impossible; in this case, we are asserting that whenever $G_t = \perp$, it is impossible to have $Proj(\hat{\tau}^T) = G$.*

PROOF. We omit $T$ from the notation, as usual. We prove this by induction on $t$. When $t = 0$, this is obviously true as $\hat{\tau}_{\geq 0} = \hat{\tau}$ and $G_0 = G$.

Suppose we have $\text{Proj}(\hat{\tau}) = G$; at time $t$ we encounter a bad-event $B$ defined by $(k_1, x_1, y_1), \ldots, (k_r, x_r, y_r)$. By inductive hypothesis, $\text{Proj}(\hat{\tau}_{\geq t}) = G_t$.

Suppose first that $\hat{\tau}_{\geq t+1}$ does not contain any bad-events $B' \sim B$. Then, by our rule for building the witness tree, we have $\hat{\tau}_{\geq t} = \hat{\tau}_{\geq t+1}$. Hence we have $G_t = \text{Proj}(\hat{\tau}_{\geq t+1})$. When we project this graph onto permutation $k$, there cannot be any source node labeled $(k, x, y)$ with $(x, y) \sim (x_i, y_i)$ as such node would be labeled with $B' \sim B$. Hence, according to our rules for updating $G_t$, we have $G_{t+1} = G_t$. So in

this case we have $\hat{\tau}_{\geq t} = \hat{\tau}_{\geq t+1}$ and $G_t = G_{t+1}$ and $\text{Proj}(\hat{\tau}_{\geq t}) = G_t$; it follows that $\text{Proj}(\hat{\tau}_{\geq t+1}) = G_{t+1}$ as desired.

Next, suppose $\hat{\tau}_{\geq t+1}$ does contain $B' \sim B$. Then bad-event $B$ will be added to $\hat{\tau}_{\geq t}$, placed below any such $B'$. When we project $\hat{\tau}_{\geq t}$, then for each $i$ with $k_i = k^*$ we add a node $(x_i, y_i)$ to $\text{Proj}(\hat{\tau}_{\geq t})$. Each such node is necessarily a source node; if such a node $(x_i, y_i)$ had a predecessor $(x'', y'') \sim (x_i, y_i)$, then the node $(x'', y'')$ would correspond to an event $B'' \sim B$ placed below $B$. Hence we see that $\text{Proj}(\hat{\tau}_{\geq t})$ is obtained from $\text{Proj}(\hat{\tau}_{\geq t})$ by adding source nodes $(x_i, y_i)$ for each $(k^*, x_i, y_i) \in B$.

So $\text{Proj}(\hat{\tau}_{\geq t}) = \text{Proj}(\hat{\tau}_{\geq t+1})$ plus the addition of source nodes for each $(k^*, x_i, y_i)$. By inductive hypothesis, $G_t = \text{Proj}(\hat{\tau}_{\geq t})$, so that $G_t = \text{Proj}(\hat{\tau}_{\geq t+1})$ plus source nodes for each $(k^*, x_i, y_i)$. Now our rule for updating $G_{t+1}$ from $G_t$ is to remove all such source nodes, so it is clear that $G_{t+1} = \text{Proj}(\hat{\tau}_{\geq t+1})$, as desired.

Note that in this proof, we assumed that $\text{Proj}(\hat{\tau}) = G$, and we never encountered the case in which $G_{t+1} = \perp$. This confirms our claim that whenever $G_{t+1} = \perp$ it is impossible to have $\text{Proj}(\hat{\tau}) = G$. □

By Proposition 7.7, the witness subdag $G$ and the future-subgraphs $G_t$ have a similar shape; they are all produced by projecting witness trees of (possibly truncated) execution logs. Note that if $G = \text{Proj}(\tau)$ for some tree $\tau$, then for any bad-event $B \in \tau$, either $B$ is not represented in $G$, or all the pairs of the form $(k^*, x, y) \in B$ are represented in $G$ and are incomparable there.

The following structural decomposition of a witness subdag $G$ will be critical.

Definition 7.8 (Alternating paths). *Given a witness subdag $G$, we define an* alternating path *in $G$ to be a simple path which alternately proceeds forward and backward along the directed edges of $G$. For a vertex $v \in G$, the* forward (respectively backward) path *of $v$ in $G$, is the maximal alternating path which includes $v$ and all the forward (respectively backward) edges emanating from $v$. Because $G$ has in-degree and out-degree at most two, every vertex $v$ has a unique forward and backward path*

*(up to reflection); this justifies our reference to "the" forward and backward path. These paths may be even-length cycles.*

*Note that if $v$ is a source node, then its backward path contains just $v$ itself. This is an important type of alternating path which should always be taken into account in our definitions.*

One type of alternating path, which is referred to as the *W-configuration*, plays a particularly important role.

DEFINITION 7.9 (The W-configuration). *Suppose $v \approx (x, y)$ has in-degree at most one, and the backward path contain an* even *number of edges, terminating at vertex $v' \approx (x', y')$. We refer to this alternating path as a* W-configuration. *(See Figure 7.3.1.)*

*Any W-configuration can be written (in one of its two orientations) as a path of vertices labeled*

$$(x_0, y_1), (x_1, y_1), (x_2, y_1), \ldots, (x_s, y_s), (x_s, y_{s+1});$$

*here the vertices $(x_1, y_1), \ldots, (x_s, y_s)$ are at the "base" of the W-configuration. Note here that we have written the path so that the $x$-coordinate changes, then the $y$-coordinate, then $x$, and so on. When written this way, we refer to $(x_0, y_{s+1})$ as the* endpoints *of the W-configuration.*

*If $v \approx (x, y)$ is a source node, then it defines a W-configuration with endpoints $(x, y)$. This should not be considered a triviality or degeneracy, rather it will be the most important type of W-configuration.*

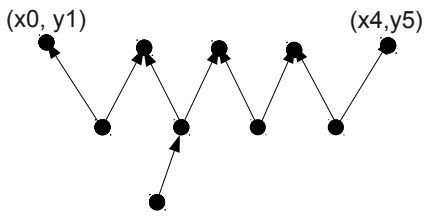


FIGURE 2. A W-configuration of length 9, with endpoints $(x_0, y_5)$.

**7.3.2. The conditions on $\pi^t_{k^*}$ encoded by $G_t$.** At any $t$, the future-subgraph $G_t$ gives certain necessary conditions on $\pi$ in order for some putative $\tau$ to appear. Proposition 7.10 describes a certain set of conditions that plays a key role in the analysis.

PROPOSITION 7.10. *Suppose we have a witness subdag $G$ for permutation $\pi$. Let $\pi^t$ denote the value of permutation $\pi$ at time $t$ and $G_t$ be the future-subgraph at time $t$. The following condition is necessary to have $G = Proj(\hat{\tau}^T)$ for some $T > 0$:*

For every W-configuration in $G_t$ with endpoints $(x_0, y_{s+1})$, we must have $\pi^t(x_0) = y_{s+1}$.

*For example, if $v \approx (x, y)$ is a source node of $G_t$, then $\pi^t(x) = y$.*

PROOF. We prove this by induction on $s$. The base case is $s = 0$; in this case we have a source node $(x, y)$. Suppose $\pi^t(x) \neq y$. In order for $\hat{\tau}^T$ to contain some bad-event containing $(k^*, x, y)$, we must at some point $t' > t$ have $\pi^{t'}(x) = y$; let $t'$ be the minimal such time. By Proposition 7.2, we must encounter a bad-event containing $(k^*, x, *)$ or $(k^*, *, y)$ at some intervening time $t'' < t'$. If this bad-event contains $(k^*, x, y)$ then necessarily $\pi^{t''}(x) = y$ contradicting minimality of $t'$. So there is a bad-event $(k^*, x, \neq y)$ or $(k^*, \neq x, y)$ earlier than the earliest occurrence of $\pi(x) = y$. This event $(k^*, x, \neq y)$ or $(k^*, \neq x, y)$ projects to a source node $(x, \neq y)$ or $(\neq x, y)$ in $G_t$. But then $(x, y)$ cannot also be a source node of $G_t$.

We now prove the induction step. Suppose we have a W-configuration with base $(x_1, y_1), \ldots, (x_s, y_s)$. At some future time $t' \geq t$ we must encounter a bad-event involving some subset of the source nodes, say the bad event includes $(x_{i_1}, y_{i_1}), \ldots, (x_{i_r}, y_{i_r})$ for $1 \leq r \leq s$. As these were necessarily source nodes, we had $\pi^{t'}(x_{i_1}) = y_{i_1}, \ldots, \pi^{t'}(x_{i_r}) = y_{i_r}$ for $l = 1, \ldots, r$. After the swaps, the updated $G_{t+1}$ has $r+1$ new W-configurations which are all strictly smaller than $s$. By inductive hypothesis, the updated permutation $\pi^{t'+1}$ must then satisfy $\pi^{t'+1}(x_0) = y_{i_1}, \pi^{t'+1}(x_{i_1}) = y_{i_2}, \ldots, \pi^{t'+1}(x_{i_r}) = y_{s+1}$.

By Proposition 7.29, we may suppose without loss of generality that the resampling of the bad event first swaps $x_{i_1}, \ldots, x_{i_r}$ in that order. Let $\pi'$ denote the result

of these swaps; there may be additional swaps to other elements of the permutation, but we must have $\pi^{t+1}(x_{i_l}) = \pi'(x_{i_l})$ for $l = 1, \ldots, r$.

In this case, we see that evidently $x_{i_1}$ swapped with $x_{i_2}$, then $x_{i_2}$ swapped with $x_{i_3}$, and so on, until eventually $x_{i_r}$ was swapped with $x'' = (\pi^{t'})^{-1} y_{s+1}$. At this point, we have $\pi'(x'') = y_{i_1}$. At the end of this process, we must have $\pi^{t'+1}(x_0) = y_{i_1}$. We claim that we must have $x'' = x_{i_0}$. For, if $x''$ is subsequently swapped, then we would have $\pi^{t'+1}(x) = y_{i_1}$ where $x$ is one of the source nodes in the current bad-event; but $x_0$ is at the *top* of the W-configuration and is not a source node.

This implies that we must have $(\pi^{t'})^{-1} y_s = x'' = x_0$; that is, that $\pi^{t'}(x_0) = y_s$. This in turn implies that $\pi^t(x_0) = y_{s+1}$. For, by Proposition 7.2, otherwise we would have encountered a bad-event involving $(x_0, *)$ or $(*, y_{s+1})$; in either case, we would have a predecessor node $(x_0, *)$ or $(*, y_{s+1})$ in $G_t$, which contradicts that we have a W-configuration. □

Proposition 7.10 can be viewed equally as a definition:

Definition 7.11 (Active conditions of a future-subgraph). *We refer to the conditions implied by Proposition 7.10 as the* active conditions *of the graph $G_t$. More formally, we define*

$$\mathit{Active}(G) = \{(x, y) \mid \ (x, y) \text{ are the end-points of a } W\text{-configuration of } G\}$$

*We also define $A_k^t$ to be the cardinality of $\mathit{Active}(G_t)$, that is, the number of active conditions of permutation $\pi_k$ at time $t$. (The subscript $k$ may be omitted in context, as usual.)*

When we remove source nodes $(x_1, y_1), \ldots, (x_r, y_r)$ from $G_t$, the new active conditions of $G_{t+1}$ are related to $(x_1, y_1), \ldots, (x_r, y_r)$ in a particular way.

Lemma 7.12. *Suppose $G$ is a future-subgraph with source nodes $v_1 \approx (x_1, y_1), \ldots, v_r \approx (x_r, y_r)$. Let $H = G - v_1 - \cdots - v_r$ denote the graph obtained from $G$ by removing*

*these source nodes. Then there is a set* $Z \subseteq \{(x_1, y_1), \ldots, (x_r, y_r)\}$ *with the following properties:*

(1) *There is an* injective *function* $f : Z \to \mathit{Active}(H)$*, with the property that* $(x, y) \sim f((x, y))$ *for all* $(x, y) \in Z$

(2) $|\mathit{Active}(H)| = |\mathit{Active}(G)| - (r - |Z|)$

***Expository remark:*** *We have recommended bearing in mind the special case when each bad-event consists of a single element. In this case, we would have* $r = 1$*; and the stated theorem would be that either* $|\mathit{Active}(H)| = |\mathit{Active}(G)| - 1$*; OR we have* $|\mathit{Active}(H)| = |\mathit{Active}(G)|$ *and* $(x_1, y_1) \sim (x_1', y_1') \in \mathit{Active}(H)$*.*

*Intuitively, we are saying that every node* $(x, y)$ *we are removing is either explicitly constrained in an "independent way" by some new condition in the graph* $H$ *(corresponding to* $Z$*), or it is almost totally unconstrained. We will never have the bad situation in which a node* $(x, y)$ *is constrained, but in some implicit way depending on the previous swaps.*

PROOF. Let $H_i$ denote the graph $G - v_1 - \cdots - v_i$. We will recursively build up the sets $Z^i, f^i$, where $Z_i \subseteq \{(x_1, y_1), \ldots, (x_i, y_i)\}$, and which satisfy the given conditions up to stage $i$.

Now, suppose we remove the source node $v_i$ from $H_{i-1}$. Observe that $(x_i, y_i) \in$ Active$(H_{i-1})$, but (unless there is some other vertex with the same label in $G$), $(x_i, y_i) \notin$ Active$(H_i)$. Thus, the most obvious change when we remove $v_i$ is that we destroy the active condition $(x_i, y_i)$. This may add or subtract other active conditions as well.

We will need to update $Z^{i-1}, f^{i-1}$. Most importantly, $f^{i-1}$ may have mapped $(x_j, y_j)$ for $j < i$, to an active condition of $H_{i-1}$ which is destroyed when $v_i$ is removed. In this case, we must re-map this to a new active condition.

Note that we cannot have $f^{i-1}(x_j, y_j) = (x_i, y_i)$ for $j < i$, as $x_i \neq x_j$ and $y_i \neq y_j$. So, the fact that $(x_i, y_i)$ has been removed from the list of active conditions does not, so far, invalidate $f^{i-1}$ remains a valid mapping from $Z^{i-1}$.

There are now a variety of cases depending on the forward-path of $v_i$ in $H_{i-1}$.

(1) This forward path consists of a cycle, or the forward path terminates on both sides in forward-edges. This is the easiest case. Then no more active conditions of $H_{i-1}$ are created or destroyed. We update $Z^i = Z^{i-1}$, $f^i = f^{i-1}$. One active condition is removed, in net, from $H_{i-1}$; hence $|\text{Active}(H_i)| = |\text{Active}(H_{i-1})| - 1$.

(2) This forward path contains a forward edge on one side and a backward edge on the other. For example, suppose the path has the form

$$(X_1, Y_1), (X_1, Y_2), (X_2, Y_2), \ldots, (X_s, Y_{s+1}),$$

where the vertices $(X_1, Y_1), \ldots, (X_s, Y_s)$ are at the base, and the node $(X_1, Y_1)$ has out-degree 1, and the node $(X_s, Y_{s+1})$ has in-degree 1. Suppose that we remove the source node $(x_i, y_i) = (X_j, Y_j)$ for $1 \leq j \leq s$. (See Figure 3.) In this case, we do not destroy any W-configurations, but we create a new W-configuration with endpoints $(X_j, Y_{s+1}) = (x_i, Y_{s+1})$.

We now update $Z^i = Z^{i-1} \cup \{(x_i, y_i)\}$. We define $f^i = f^{i-1}$ plus we map $(x_i, y_i)$ to the new active condition $(x_i, Y_{s+1})$. In net, no active conditions were added or removed, and $|\text{Active}(H_i)| = |\text{Active}(H_{i-1})|$.

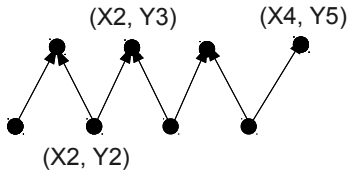


FIGURE 3. When we remove $(X_2, Y_2)$, we create a new W-configuration with endpoints $(X_2, Y_5)$.

(3) This forward path was a W-configuration $(X_0, Y_1), (X_1, Y_1), \ldots, (X_s, Y_s), (X_s, Y_{s+1})$ with $(X_1, Y_1), \ldots, (X_s, Y_s)$ on the base, and we had $(x_i, y_i) = (X_j, Y_j)$. This is the most complicated situation; in this case, we destroy the original W-configuration with endpoints $(X_0, Y_{s+1})$ but create two new W-configurations

with endpoints $(X_0, Y_j)$ and $(X_j, Y_{s+1})$. We update $Z^i = Z^{i-1} \cup \{(x_i, y_i)\}$. We will set $f^i = f^{i-1}$, except for a few small changes as follows.

Now, suppose $f^{i-1}(x_l, y_l) = (X_0, Y_{s+1})$ for some $l < i$; so either $x_l = X_0$ or $y_l = Y_{s+1}$. If it is the former, we set $f^i(x_l, y_l) = (X_0, Y_j), f^i(x_i, y_i) = (X_j, Y_{s+1})$. If it is the latter, we set $f^i(x_l, y_l) = (X_j, Y_{s+1}), f^i(x_i, y_i) = (X_0, Y_j)$.. If $(f^{i-1})^{-1}(X_0, Y_{s+1}) = \emptyset$ then we simply set $f^i(x_i, y_i) = (X_0, Y_j)$.

In any case, $f^i$ is updated appropriately, and in the net no active conditions are added or removed, so we have $|\text{Active}(H_i)| = |\text{Active}(H_{i-1})|$.

□

### 7.4. The probability that the swaps are all successful

In the previous sections, we determined necessary conditions for the permutations $\pi^t$, depending on the graphs $G_t$. In this section, we finish by computing the probability that the swapping subroutine causes the permutations to, in fact, satisfy all such conditions.

Proposition 7.13 states the key randomness condition satisfied by the swapping subroutine. The basic intutition behind this is as follows: suppose $\pi : [n] \to [n]$ is a fixed permutation with $\pi(x) = y$, and we call $\pi' = \text{Swap}(\pi; x_1, \ldots, x_r)$. Then $\pi'(x_1)$ has a uniform distribution over $[n]$. Similarly, $\pi'^{-1}(y_1)$ has a uniform distribution over $[n]$. However, the joint distribution is *not* uniform — there is essentially only one degree of freedom for the two values. In general, any subset of the variables $\pi'(x_1), \ldots, \pi'(x_r), \pi'^{-1}(y_1), \ldots, \pi^{-1}(y_r)$ will have the uniform distribution, *as long as the subset does not simultaneously contain* $\pi'(x_i), \pi'^{-1}(y_i)$ *for some* $i \in [r]$.

PROPOSITION 7.13. *Suppose* $n, r, s, q$ *are non-negative integers obeying the following constraints:*

(1) $0 \leq s \leq \min(q, r)$

(2) $q + (r - s) \leq n$

*Let $\pi$ be a fixed permutation of $[n]$, and let $x_1, \ldots, x_r \in [n]$ be distinct, and let $y_i = \pi(x_i)$ for $i = 1, \ldots, r$. Let $(x'_1, y'_1), \ldots, (x'_q, y'_q)$ be a given list with the following properties:*

(3) *All $x'$ are distinct; all $y'$ are distinct*

(4) *For $i = 1, \ldots, s$ we have $x_i = x'_i$ or $y_i = y'_i$.*

*Let $\pi' = Swap(\pi; x_1, \ldots, x_r)$. Then the probability that $\pi'$ satisfies all the constraints $(x', y')$ is at most*

$$P(\pi'(x'_1) = y'_1 \wedge \cdots \wedge \pi'(x'_q) = y'_q) \leq \frac{(n-r)!(n-q)!}{n!(n-q-r+s)!}$$

***Expository remark:*** *Consider the special case when each bad-event contains a single element. In that case, we have $r = 1$. There are two possibilities for $s$; either $s = 0$ in which case this probability on the right is $1 - q/n$ (i.e. the probability that $\pi'(x_1) \neq y'_1, \ldots, y'_q$); or $s = 1$ in which case this probability is $1/n$ (i.e. the probability that $\pi'(x_1) = y'_1$).*

PROOF. Define the function $g(n, r, s, q) = \frac{(n-r)!(n-q)!}{n!(n-q-r+s)!}$. We will prove this proposition by induction on $s, r$. There are a few cases we handle separately:

(1) Suppose $s > 0$ and $x_1 = x'_1$. Then, in order to satisfy the desired conditions, we must swap $x_1$ to $x'' = \pi^{-1}(y'_1)$; this occurs with probability $1/n$. The subsequent $r - 1$ swaps starting with the permutation $\pi(x_1\ x'')$ must now satisfy the conditions $\pi'(x'_2) = y'_2, \ldots, \pi'(x_q) = y'_q$. We claim that we have $(x_i, \pi(x_1\ x'')x_i) \sim (x'_i, y'_i)$ for $i = 2, \ldots, s$. If $x'' \neq x_2, \ldots, x_s$, this is immediately clear. Otherwise, suppose $x'' = x_j$. If $x_j = x'_j$, then we again still have $(x_j, \pi(x_1\ x'')x_j) \sim (x'_j, y'_j)$. If $y_j = y'_j$, then this implies that $y'_1 = y_j = y'_j$, which contradicts that the $y'_j \neq y'_1$ .

So we apply the induction hypothesis to $\pi(x_1\ x'')$; in the induction, we subtract one from $n, q, r, s$. This gives

$$P(\pi'(x'_1) = y'_1 \wedge \cdots \wedge \pi(x'_q) = y'_q) \leq \tfrac{1}{n} g(n-1, r-1, s-1, q-1) = g(n, r, s, q)$$

as desired.

(2) Similarly, suppose $s > 0$ and suppose $y_1 = y'_1$. By Proposition 7.30, we would obtain the same distribution if we executed $(\pi')^{-1} = \text{Swap}(\pi^{-1}; y_1, \ldots, y_r)$. Hence we have

$$P(\pi'(x'_1) = y'_1 \wedge \cdots \wedge \pi(x'_q) = y'_q) = P((\pi')^{-1}(y'_1) = x'_1 \wedge \cdots \wedge (\pi')^{-1}(y'_q) = x'_q)$$

Now, the right-hand side has swapped the roles of $x_1/y_1$; in particular, it now falls under the previous case (1) already proved, and so the right-hand side is at most $g(n, r, s, q)$ as desired.

(3) Suppose $s = 0$ and that there is some $i \in [r], j \in [q]$ with $(x_i, y_i) \sim (x'_j, y'_j)$. By Proposition 7.29, we can assume without loss of generality that $(x_1, y_1) \sim (x'_1, y'_1)$. So, in this case, we are really in the case with $s = 1$; by apply the induction hypothesis so that

$$P(\pi'(x'_1) = y'_1 \wedge \cdots \wedge \pi(x'_q) = y'_q) \leq g(n, r, 1, q) = \frac{g(n, r, 0, q)}{n - q - r + 1} \leq g(n, r, s, q)$$

Here, we are using our hypothesis that $n \geq q + (r - s) = q + r$.

(4) Finally, suppose $s = 0$ and $x_1, \ldots, x_r$ are distinct from $x'_1, \ldots, x'_q$ and $y_1, \ldots, y_q$ are distinct from $y'_1, \ldots, y'_q$. In this case, a necessary (although not sufficient) condition to have $\pi'(x'_1) = y'_1, \ldots, \pi(x'_q) = y'_q$ is that there are some $y''_1, \ldots, y''_r$, distinct from each other and distinct from $y'_1, \ldots, y'_q$, with the property that $\pi'(x_i) = y''_i$ for $j = 1, \ldots, r$. By the union bound, we have

$$P(\pi'(x'_1) = y'_1 \wedge \cdots \wedge \pi(x'_q) = y'_q) \leq \sum_{y''_1, \ldots, y''_r} P(\pi'(x_1) = y''_1 \wedge \cdots \wedge \pi(x_r) = y''_r)$$

For each individual summand, we apply the induction hypothesis; the summand has probability at most $g(n, r, r, q)$. As there are $(n-q)!/(n-q-r)!$ possible values for $y''_1, \ldots, y''_r$, the total probability is at most $(n-q)!/(n-q-r)! \times g(n, r, r, q) = g(n, r, s, q)$.

□

We apply Proposition 7.13 to upper-bound the probability that the Swapping Algorithm successfully swaps when it encounters a bad event.

PROPOSITION 7.14. *Suppose we encounter a bad-event $B$ at time $t$ containing elements $(k, x_1, y_1)$, $\ldots$, $(k, x_r, y_r)$ from permutation $k$ (and perhaps other elements from other permutations). Then the probability that $\pi_k^{t+1}$ satisfies all the active conditions of its future-subgraph, conditional on all past events and all other swappings at time $t$, is at most*

$$P(\pi_k^{t+1} \text{ satisfies } Active(G_{k,t+1})) \leq P_k(B) \frac{(n_k - A_k^{t+1})!}{(n_k - A_k^t)!}.$$

*Recall that we have defined $A_k^t$ to be the number of active conditions in the future-subgraph corresponding to permutation $\pi_k$ at time $t$, and we have defined $P_k(B) = \frac{(n_k - r)!}{n_k!}$.*

***Expository remark:*** *Consider the special case when each bad-event consists of a single element. In this case, we would have $P_k(B) = 1/n$. The stated theorem is now: either $A^{t+1} = A^t$, in which case the probability that $\pi$ satisfies its swapping condition is $1/n$; or $A^{t+1} = A^t - 1$; in which case the probability that $\pi$ satisfies its swapping condition is $1 - A^{t+1}/n$.*

PROOF. Let $H$ denote the future-subgraph $G_{k,t+1}$ after removing the source nodes corresponding to $(x_1, y_1), \ldots, (x_r, y_r)$. Using the notation of Lemma 7.12, we set $s = |Z|$ and $q = A_k^{t+1}$. We have Active$(H) = \{(x'_1, y'_1), \ldots, (x'_q, y'_q)\}$.

For each $(x, y) \in Z$, we have $y = \pi^t(x)$, and there is an injective function $f : Z \to$ Active$(H)$ and $(x, y) \sim f((x, y))$. By Proposition 7.29, we can assume without loss of

generality $Z = \{(x_1, y_1), \ldots, (x_s, y_s)\}$ and $f(x_i, y_i) = (x'_i, y'_i)$. In order to satisfy the active conditions on $G_{k,t+1}$, the swapping must cause $\pi^{t+1}(x'_i) = y'_i$ for $i = 1, \ldots, q$.

By Lemma 7.12, we have $A_k^t = A_k^{t+1} + (r - s) = q + (r - s)$. Note that $A_k^t \leq n$. So all the conditions of Proposition 7.13 are satisfied. Thus this probability is at most $\frac{(n_k-r)!}{n_k!} \times \frac{(n_k-q)!}{(n_k-q-r+s)!} = \frac{(n_k-r)!(n_k-A_k^{t+1})!}{n_k!(n_k-A_k^t)!}$. □

We have finally all the pieces necessary to prove Lemma 7.1.

LEMMA 7.1. *Let $\tau$ be a tree-structure, with nodes labeled $B_1, \ldots, B_s$. The probability that $\tau$ appears is at most*

$$P(\tau \text{ appears}) \leq P_\Omega(B_1) \cdots P_\Omega(B_s)$$

PROOF. The Swapping Algorithm, as we have defined it, begins by selecting the permutations uniformly at random. One may also consider fixing the permutations to some arbitrary (not random) value, and allowing the Swapping Algorithm to execute from that point onward. We refer to this as *starting at an arbitrary state of the Swapping Algorithm.* We will prove the following by induction on $\tau'$: The probability, starting at an arbitrary state of the Swapping Algorithm, that the subsequent swaps would produce the subtree $\tau'$, is at most

$$P(\hat{\tau}^T = \tau' \text{ for some } T \geq 0) \leq \prod_{B \in \tau'} P_\Omega(B) \times \prod_{k=1}^{N} \frac{n_k!}{(n_k - |\text{Active}(\text{Proj}_k(\tau'))|)!}. \tag{36}$$

When $\tau' = \emptyset$, the RHS of (36) is equal to one so this is vacuously true.

To show the induction step, note that in order for $\tau'$ to be produced as the witness tree for some $T \geq 0$, it must be that some $B$ is resampled, where some node $v \in \tau'$ is labeled by $B$. Suppose we condition on that $v$ is the first such node, resampled at time $t$. A necessary condition to have $\hat{\tau}^T = \tau'$ for some $T \geq t$ is that $\pi^{t+1}$ satisfies all the active conditions on $G_{t+1}$. By Proposition 7.14, the probability that $\pi^{t+1}$ satisfies these conditions is at most $\prod_k P_k(B) \frac{(n_k-A_k^{t+1})!}{(n_k-A_k^t)!}$.

Next, if this event occurs, then subsequent resamplings must cause $\hat{\tau}^T_{\geq t+1} = \tau' - v$. To bound the probabilit of this, we use the induction hypothesis. Note that the induction hypothesis gives a bound conditional on *any* starting configuration of the Swapping Algorithm, so we may multiply these probabilities. Thus

$$\begin{aligned}
P(\hat{\tau}^T = \tau' \text{ for some } T > 0) &\leq \prod_k P_k(B) \frac{(n_k - A_k^{t+1})!}{(n_k - A_k^t)!} \\
&\quad \times \prod_{B \in \tau' - v} P_\Omega(B) \times \prod_{k=1}^{N} \frac{n_k!}{(n_k - |\text{Active}(\text{Proj}_k(\tau' - v))|)!} \\
&= \prod_{B \in \tau'} P_\Omega(B) \prod_k \frac{(n_k - A_k^{t+1})!}{(n_k - A_k^t)!} \frac{n_k!}{(n_k - |\text{Active}(\text{Proj}_k(\tau' - v))|)!} \\
&= \prod_{B \in \tau'} P_\Omega(B) \prod_k \frac{n_k!}{(n_k - A_k^t)!} \quad \text{as } A_k^{t+1} = |\text{Active}(\text{Proj}_k(\tau' - v))|
\end{aligned}$$

completing the induction argument.

We now consider the necessary conditions to produce the *entire* tree-structure $\tau$, and not just fragments of it. First, the *original permutations* $\pi_k^0$ must satisfy the active conditions of the respective witness subdags $\text{Proj}_k(\tau)$. For each permutation $k$, this occurs with probability $\frac{(n_k - A_k^0)!}{n_k!}$. Next, the subsequent sampling must be compatible with $\tau$; by (36) this has probability at most $\prod_{B \in \tau} P_\Omega(B) \times \prod_{k=1}^{N} \frac{n_k!}{(n_k - A_k^0)!}$. Again, note that the bound in (36) is conditional on any starting position of the Swapping Algorithm, hence we may multiply these probabilities. In total we have

$$P(\hat{\tau}^T = \tau \text{ for some } T \geq 0) \leq \prod_k \frac{(n_k - A_k^0)!}{n_k!} \times \prod_{B \in \tau} P_\Omega(B) \times \prod_{k=1}^{N} \frac{n_k!}{(n_k - A_k^0)!} = \prod_{B \in \tau} P_\Omega(B).$$

We note one counter-intuitive aspect to this proof. The natural way of proving this lemma would be to identify, for each bad-event $B \in \tau$, some necessary event occuring with probability at most $P_\Omega(B)$. This is the general strategy used in Chapter 2 and related constructive LLL variants such as [**1**], [**51**]. This is *not* the proof we employ here; there is an additional factor of $(n_k - A_k^0)!/n!$ which is present for the original

permutation and is gradually "discharged" as active conditions disappear from the future-subgraphs. □

### 7.5. The constructive LLL for permutations

Now that we have proved the Witness Tree Lemma, the remainder of the analysis is essentially the same as for the MT algorithm [**87**]. Using arguments and proofs from Chapter 2, with our key lemma, we can now easily show our key theorem:

THEOREM 7.15. *Suppose there is some assignment of weights* $\mu : \mathcal{B} \to [0, \infty)$ *which satisfies, for every* $B \in \mathcal{B}$ *the condition*

$$\mu(B) \geq P_\Omega(B)\theta(B)$$

*Then the Swapping Algorithm terminates with probability one. The expected number of iterations in which we resample* $B$ *is at most* $\mu(B)$*. (Recall the definition of* $\theta(B)$ *for an event* $E$ *from Section 2.2; here* $\theta(B)$ *is shorthand for* $\theta_G(B)$*, where* $G$ *is the canonical dependency graph on permutations.)*

In the "symmetric" case, this gives us the well-known LLL criterion:

COROLLARY 7.16. *Suppose each bad-event* $B \in \mathcal{B}$ *has probability at most* $p$*, and is dependent with at most* $d$ *bad-events (including itself). Then if* $epd \leq 1$*, the Swapping Algorithm terminates with probability one; the expected number of resamplings of each bad-event is* $O(1)$*.*

Some extensions of the LLL, such as the Moser-Tardos distribution bounds or the Partial Resampling Algorithm, follow almost immediately here. There are a few extensions which require slightly more discussion:

**7.5.1. Lopsidependency.** It is possible to slightly restrict the notion of lopsidependency for permutations. We can re-define the relation $\sim$ on bad-events as follows: for $B, B' \in \mathcal{B}$, we have $B \sim B'$ iff

(1) $B = B'$, *or*

(2) there is some $(k, x, y) \in B, (k, x', y') \in B'$ with either $x = x', y \neq y'$ or $x \neq x', y = y'$.

In particular, bad-events which share the same triple $(k, x, y)$, are *not* caused to be dependent.

Proving that the Swapping Algorithm still works in this setting requires only a slight change in our definition of $\text{Proj}_k(\tau)$. Now, the tree $\tau$ may have multiple copies of any given triple $(k, x, y)$ on a single level. When this occurs, we create the corresponding nodes $v \approx (x, y) \in \text{Proj}_k(\tau)$; edges are added between such nodes in an arbitrary (but consistent) way. The remainder of the proof remains as before.

This change is not compatible with the parallel algorithm we develop in Section 7.6, however.

**7.5.2. LLL for injective functions.** The analysis of [**77**] considers a slightly more general setting for the LLL, in which we select random *injections* $f_k : [m_k] \to [n_k]$, where $m_k \leq n_k$. In fact, our Swapping Algorithm can be extended to this case. We simply define a permutation $\pi_k$ on $[n_k]$, where the entries $\pi_k(m_k + 1), \ldots, \pi_k(n_k)$ are "dummies" which do not participate in any bad-events. The LLL criterion for the extended permutation $\pi_k$ is exactly the same as the corresponding LLL criterion for the injection $f_k$. Because all of the dummy entries have the same behavior, it is not necessary for the Swapping Algorithm to keep track of the dummy entries exactly; they are needed only for the analysis.

**7.5.3. Comparison with the approach of Achlioptas & Iliopoulos, Harvey & Vondrak, and Kolmogorov.** Achlioptas & Iliopoulos [**1**], Harvey & Vondrak [**51**], and Kolmogorov [**68**] gave generic frameworks for analyzing variants of the MT algorithm, applicable to different types of combinatorial configurations. These frameworks can include vertex-colorings, permutations, Hamiltonian cycles of graphs, spanning trees, matchings, and other settings. For the case of permutations, both of

these frameworks give the same algorithm as our Swapping Algorithm and show that it terminates under the same conditions as we do, which in turn are the same conditions as the probabilistic LLL.

The key difference between the our approach and the other frameworks is that the latter enumerate the entire history of all resamplings to the permutations. By contrast, our proof is based on the Witness Tree Lemma; this is a much more succint structure, which ignores most of the resamplings, and only enumerates the few resamplings that are necessary to justify a single item in the execution log. The other papers are simpler than ours; a huge part of the complexity of our proof lies in the need to argue that the bad-events which were ignored by the witness tree do not affect the stochastic probabilities. (The ignored bad-events *do* interact with the variables we need to track for the witness tree, but do so in a "neutral" way.)

*If our only goal is to prove that the Swapping Algorithm terminates in polynomial time, then the other frameworks give a better and simpler approach.*

However, the Witness Tree Lemma allows much more precise estimate for many types of events. The main reason for this precision is the following: suppose we want to show that some event $E$ has a low probability of occuring during or after the execution of the Swapping Algorithm. The proof strategy of Moser & Tardos is to take a union-bound over all tree-structures that correspond to this event. In this case, we are able to show a probability bound which is proportional to the total weight of all such tree-structures. This can be a relatively small number as only the witness trees connected to $E$ are relevant. Our analysis, which is also based on witness trees, is able to show similar types of bounds.

However, the analyses of Achlioptas & Iliopoulos, Harvey & Vondrak, and Kolmogorov (herafter the "alternate analyses") are not based on witness trees, but the much larger set of *full execution logs*. The number of possible execution logs can be exponentially larger than the number of witness trees. It is very inefficient to take a

union bound over all such logs. Hence, the alternate analyses give bounds which are exponentially weaker than the ones we provide.

Many properties of the Swapping Algorithm depend on the fine degree of control provided by the Witness Tree Lemma, and it seems difficult to obtain them from the alternate LLLL approaches. We list a few of these properties here.

**The LLL criterion without slack.** As a simple example of the problems caused by taking a union bound over execution logs, suppose that we satisfy the LLL criterion without slack, say $epd = 1$. In this case, we show that the expected time for our Swapping Algorithm to terminate is $O(m)$. By contrast, the alternate analyses require satisfying the LLL criterion with slack $ep(1+\epsilon)d = 1$, and achieve a termination time of $O(m/\epsilon)$. They require this slack term in order to damp the exponential growth in the number of execution logs.[1]

**Arbitrary choice of which bad-event to resample.** The Swapping Algorithm as we have stated it is actually under-determined, in that the choice of which bad-event to resample is arbitrary. By contrast, in both Achlioptas & Iliopoulos and Harvey & Vondrak, there is a fixed priority on the bad-events. (The work of [**68**] removes this restriction in certain special cases of the Achlioptas & Iliopoulous setting, including for random permutations and matchings.) This freedom can be quite useful. For example, in Section 7.6 we consider a parallel implementation of our Swapping Algorithm. We will select which bad-events to resample in a very complicated and randomized way. However, the correctness of the parallel algorithm will follow from the fact that it simulates some serial implementation of the Swapping Algorithm.

**The Moser-Tardos distribution.** The Witness Tree Lemma allows us to analyze the so-called "Moser-Tardos (MT) distribution," first discussed by [**48**]. The LLL and its algorithms ensure that bad-events $\mathcal{B}$ cannot possibly occur. In other words, we know that the configuration produced by the LLL has the property that

[1]Harvey & Vondrak show that if the symmetric LLL criterion is satisfied without slack, then the Shearer criterion is satisfied with slack $\epsilon = O(1/m)$. Thus, they would achieve a running time of $O(m^2)$ without slack. This extra slack, which our analysis does not need, is not present in all settings of the LLL.

no $B \in \mathcal{B}$ is true. In many applications of the LLL, we may wish to know more about such configurations, other than they exist.

There are a variety of reasons we might want this; we give two examples for the ordinary, variable-based LLL. Suppose that we have some weights for the values of our variables, and we define the objective function on a solution $\sum_i w(X_i)$; in this case, if we are able to estimate the probability that a variable $X_i$ takes on value $j$ in the *output* of the LLL (or MT algorithm), then we may be able to show that configurations with a good objective function exist. A second example is when the number of bad-events becomes too large, perhaps exponentially large. In this case, the MT algorithm cannot test them all. However, we may still be able to ignore a subset of the bad events, and argue that the probability that they are true at the end of the MT algorithm is small even though they were never checked. Thus, by proving the Witness Tree Lemma, we automatically show that Theorem 2.16 holds for the Swapping Algorithm as well.

As one concrete example of this, we will show in Section 9.3 how to accelerate the Swapping Algorithm in some applications by efficiently searching for true bad-events. These results are almost immediate from our Witness Tree Lemma, but would not be possible to show using the other approaches.

**Bounds on the depth of the resampling process.** One key requirement for parallel variants of the MT algorithm appears to be that the resampling process has logarithmic depth. This is equivalent to showing that there are no deep witness trees. This follows easily from the Witness Tree Lemma but appears to be very difficult in the other LLLL frameworks.

**Partial resampling.** We have developed in Chapter 5 a partial resampling variant of the MT algorithm. To analyze this variant, we developed an alternate type of witness tree, which only records the variables which were actually resampled. Ignoring the other variables can drastically prunes the space of witness trees. Again, this does not seem to be possible in other LLLL frameworks in which the *full* execution

log must be recorded. We will see an example of this in Theorem 7.24; we do not know of any way to show Theorem 7.24 using the alternate analyses.

### 7.6. A parallel version of the Swapping Algorithm

In this section, we will describe a parallel algorithm for the Swapping Algorithm, which runs along the same lines as the parallel MT algorithm. However, everything is more complicated than in the case of the ordinary LLL. In the MT algorithm, events which are not connected to each other cannot affect each other in any way. For the permutation LLL, such events can interfere with each other, but do so rarely.

Consider the following example. Suppose that at some point we have two active bad-events, "$\pi_k(1) = 1$" and "$\pi_k(2) = 2$" respectively, and so we decide to resample them simultaneously (since they are not connected to each other, and hence constitute an independent set). When we are resampling the bad-event $\pi_k(1) = 1$, we may swap 1 with 2; in this case, we are automatically fixing the second bad-event as well. The sequential algorithm, in this case, would only swap a single element. The parallel algorithm should likewise *not* perform a second swap for the second bad-event, or else it would be over-sampling. Avoiding this type of conflict is quite tricky.

Let $n = n_1 + \cdots + n_N$; since the output of the algorithm will be the contents of the permutations $\pi_1, \ldots, \pi_N$, this algorithm should be measured in terms of $n$, and we must show that this algorithm runs in $\log^{O(1)} n$ time. We will make the following assumptions in this section. First, we assume that $|\mathcal{B}|$, the total number of potential bad-events, is polynomial in $n$. This assumption can be relaxed if we have the proper kind of "separation oracle" for $\mathcal{B}$. Next, we assume that every element $B \in \mathcal{B}$ has size $|B| \leq M = \log^{O(1)} n$; this holds in many cases.

We describe the following Parallel Swapping Algorithm:

(1) In parallel, generate the permutations $\pi_1, \ldots, \pi_N$ uniformly at random.

(2) We proceed through a series of *rounds* while there is some true bad-event. In round $i$ ($i = 1, 2, \ldots,$) do the following:

(3) Let $\mathcal{V}_{i,1} \subseteq \mathcal{B}$ denote the set of bad-events which are currently true at the beginning of round $i$. We will attempt to fix the bad-events in $\mathcal{V}_{i,1}$ through a series of *sub-rounds*. This may introduce new bad-events, but we will not fix any newly created bad-events until round $i+1$.

We repeat the following for $j = 1, 2, \ldots$ as long as $\mathcal{V}_{i,j} \neq \emptyset$:

(4) Let $I_{i,j}$ be a maximal independent set (MIS) of bad-events in $\mathcal{V}_{i,j}$.

(5) For each true bad-event $B \in I_{i,j}$, choose the swaps corresponding to $B$. Namely, if we have some bad-event $B$ involving triples $(k_1, x_1, y_1), \ldots, (k_r, x_r, y_r)$, then we select each $z_l \in [n_{k_l}]$, which is the element to be swapped with $\pi_{k_l}(x_l)$ according to procedure Swap. *Do not perform the indicated swaps at this time though!* We refer to $(k_1, x_1), \ldots, (k_r, x_r)$ as the swap-sources of $B$ and the $(k_1, z_1), \ldots, (k_r, z_r)$ as the swap-mates of $B$.

(6) Select a random ordering $\rho_{i,j}$ of the elements of $I_{i,j}$. Consider the graph $G_{i,j}$ whose vertices correspond to elements of $I_{i,j}$: add an edge connecting $B$ with $B'$ if $\rho_{i,j}(B) < \rho_{i,j}(B')$ *and* one of the swap-mates of $B$ is a swap-source of $B'$. Generate $I'_{i,j} \subseteq I_{i,j}$ as the *lexicographically-first MIS* (LFMIS) of the resulting graph $G_{i,j}$, with respect to the vertex-ordering $\rho_{i,j}$.

(7) For each permutation $\pi_k$, enumerate all the transpositions $(x\ z)$ corresponding to elements of $I'_{i,j}$, arranged in order of $\rho_{i,j}$. Say these transpositions are, in order $(x_1, z_1), \ldots (x_l, z_l)$, where $l \leq n$. Compute, in parallel for all $\pi_k$, the composition $\pi'_k = \pi_k(x_l\ z_l) \ldots (x_1\ z_1)$.

(8) Update $\mathcal{V}_{i,j+1}$ from $\mathcal{V}_{i,j}$ by removing all elements which are either no longer true for the current permutation, *or* are connected via $\sim$ to some element of $I'_{i,j}$.

Most of the steps of this algorithm can be implemented using standard parallel algorithms. For example, step (1) can be performed simply by having each element

of $[n_k]$ choose a random real and then executing a parallel sort. The independent set $I_{i,j}$ can be found in time in polylogarithmic time using [**8, 80**].

The difficult step to parallelize is in selecting the LFMIS $I'_{i,j}$. In general, the problem of finding the LFMIS is P-complete [**29**], hence we do not expect a generic parallel algorithm for this. However, what saves us it that the ordering $\rho_{i,j}$ and the graph $G_{i,j}$ are constructed in a highly random fashion.

This allows us to use the following greedy algorithm to construct $I'_{i,j}$, the LFMIS of $G_{i,j}$:

(1) Let $H_1$ be the directed graph obtained by orienting all edges of $G_{i,j}$ in the direction of $\rho_{i,j}$. Repeat the following for $s = 1, 2, \ldots,$:
  (2) If $H_s = \emptyset$ terminate.
  (3) Find all source nodes of $H_s$. Add these to $I'_{i,j}$.
  (4) Construct $H'_{s+1}$ by removing all source nodes and all successors of source nodes from $H'_s$.

The output of this algorithm is the LFMIS $I'_{i,j}$. Each step can be implemented in parallel time $O(1)$. The number of iterations of this algorithm is the length of the longest directed path in $G'_{i,j}$. So it suffices it show that, whp, all directed paths in $G'_{i,j}$ have length at most polylogarithmic in $n$.

PROPOSITION 7.17. *Let $I \subseteq \mathcal{B}$ be an an arbitrary independent set of true bad-events, and suppose all elements of $\mathcal{B}$ have size $\leq M$. Let $G = G_{i,j}$ be the graph constructed in Step (6) of the Parallel Swapping Algorithm.*

*Then whp, every directed path in $G$ has length $O(M + \log n)$.*

PROOF. One of the main ideas below is to show that for the *typical* $B_1, \ldots, B_l \in I$, where $l = 5(M + \log n)$, the probability that $B_1, \ldots, B_l$ form a directed path is small. Suppose we select $B_1, \ldots, B_l \in I$ uniformly at random without replacement. Let us analyze how these could form a directed path in $G$. (We may assume $|I| > l$ or otherwise the result holds trivially.)

First, it must be the case that $\rho(B_1) < \rho(B_2) < \cdots < \rho(B_l)$. This occurs with probability $1/l!$.

Next, it must be that the swap-mates of $B_s$ overlap the swap-sources of $B_{s+1}$, for $s = 1, \ldots, l-1$. Now, $B_s$ has $O(M)$ swap-mates; each such swap-mate can overlap with at most one element of $I$, since $I$ is an independent set. Conditional on having chosen $B_1, \ldots, B_s$, there are a remaining $|I| - s$ choices for $B_{s+1}$. This gives that the probability of having $B_s$ with an edge to $B_{s+1}$, conditional on the previous events, is at most $\frac{M}{|I|-s}$. (The fact that swap-mates are chosen randomly does not give too much of an advantage here.)

Putting this all together, the total probability that there is a directed path on $B_1, \ldots, B_l$ is

$$P(\text{directed path } B_1, \ldots, B_l) \leq \frac{M^{l-1}(|I|-l)!}{(|I|-1)!l!}$$

Since the above was for a random $B_1, \ldots, B_l$, the probability that there is *some* such path (of length $l$) is at most

$$\begin{aligned} P(\text{some directed path}) &\leq \frac{|I|!}{(|I|-l)!} \times \frac{M^{l-1}(|I|-l)!}{(|I|-1)!l!} \\ &= |I| \times \frac{M^{l-1}}{l!} \leq n \times \frac{M^{l-1}}{(l/e)^l} \leq n^{-\Omega(1)}, \end{aligned}$$

since $l = 5(M + \log n)$. □

So far, we have shown that each sub-round of the Parallel Swapping Algorithm can be executed in parallel time $\log^{O(1)} n$. Next, we show that whp that number of sub-rounds corresponding to any round is bounded by $\log^{O(1)} n$.

PROPOSITION 7.18. *Suppose $|\mathcal{B}| = n^{O(1)}$ and all elements $B \in \mathcal{B}$ have size $|B| \leq M$. Then whp, we have $\mathcal{V}_{i,j} = \emptyset$ for some $j = O(M \log^2 n)$.*

PROOF. We will first show the following: Let $B \in I$, where $I$ is an arbitrary independent set of $\mathcal{B}$. Then with probability at least $1 - \frac{1}{2M \ln n}$ we have $B \in I'$ as well, where $I'$ is the LFMIS associated with $I$.

Observe that if there is no $B' \in I$ such that $\rho(B') < \rho(B)$ and such that a swap-mate of $B'$ overlaps with a swap-source of $B$, then $B \in I'$ (this is not a necessary condition). We will analyze the ordering $\rho$ using the standard trick, in which each element $B \in I$ chooses a rank $W(B) \sim \text{Uniform}[0,1]$, independently and identically. The ordering $\rho$ is then formed by sorting in increasing ordering of $W$. In this way, we are able to avoid the dependencies induced by the rankings. For the moment, let us suppose that the rank $W(B)$ is *fixed* at some real value $w$. We will then count how many $B' \in I$ satisfy $W(B') < w$ and a swap-mate of $B'$ overlaps a swap-source of $B$.

So, let us consider some swap-source $s$ of $B$ in permutation $k$, and consider some $B'_j \in I$ which has $r'_j$ other elements in permutation $k$. For $l = 1, \ldots, r'_j$, there are $n_k - l + 1$ possible choices for the $l$th swap-mate from $B'_j$, and hence the total expected number of swap-mates of $B'$ which overlap $s$ is at most

$$\begin{aligned}
\mathbf{E}[\text{ \# swap-mates of } B'_j \text{ overlapping } s] &\leq \sum_{l=1}^{r'_j} \frac{1}{n_k - l + 1} \\
&\leq \int_{l=1}^{r'_j+1} \frac{1}{n_k - l + 1} dl \\
&= \ln(\frac{n_k}{n_k - r'_j})
\end{aligned}$$

Next, sum over all $B'_j \in I$. Observe that since $I$ is an independent set, we must have $\sum r'_j \leq n_k - 1$. Thus,

$$\begin{aligned}
\mathbf{E}[\text{ \# swap-mates of some } B'_j \text{ overlapping } s] &\leq \sum_j \ln(\frac{n_k}{n_k - r'_j}) \\
&\leq \ln(\frac{n_k}{n_k - \sum_j r'_j}) \qquad \text{by concavity} \\
&\leq \ln n_k \leq \ln n
\end{aligned}$$

Thus, summing over all swap-sources of $B$, the total probability that there is some $B'$ with $\rho(B') \leq B$ and for which a swap-mate overlaps a swap-source of $B$, is at most

$w|B| \ln n \leq wM \ln n$. By Markov's inequality, we have

$$P(B' \in I' \mid W(B) = w) \geq 1 - wM \ln n$$

Integrating over $w$, we have that $B' \in I'$ with probability at least

$$P(B' \in I') \geq 1 - \frac{1}{2M \ln n}$$

Now, using this fact, we show that $\mathcal{V}_{i,j}$ is decreasing quickly in size. For, suppose $B \in \mathcal{V}_{i,j}$. So $B \sim B'$ for some $B' \in I_{i,j}$, as $I_{i,j}$ is a maximal independent set (possibly $B = B'$). We will remove $B$ from $\mathcal{V}_{i,j+1}$ if $B' \in I'_{i,j}$, which occurs with probability at least $1 - \frac{1}{2M \ln n}$. As $B$ was an arbitrary element of $\mathcal{V}_{i,j}$, this shows that $\mathbf{E}\big[|\mathcal{V}_{i,j+1}| \mid \mathcal{V}_{i,j}\big] \leq (1 - \frac{1}{2M \ln n})|\mathcal{V}_{i,j}|$.

For $j = \Omega(M \log^2 n)$, this implies that

$$\mathbf{E}\big[|\mathcal{V}_{i,j}|\big] \leq (1 - \frac{1}{2M \ln n})^{\Omega(M \log^2 n)} |\mathcal{V}_{i,1}| \leq n^{-\Omega(1)}$$

This in turn implies that $\mathcal{V}_{i,j} = \emptyset$ with high probability, for $j = \Omega(M \log^2 n)$. $\square$

To finish the proof, we must show that the number of rounds is itself bounded whp. We begin by showing that Witness Tree Lemma remains valid in the parallel setting.

PROPOSITION 7.19. *When we execute this parallel swapping algorithm, we may generate an "execution log" according to the following rule: suppose that we resample $B$ in round $i, j$ and $B'$ in round $i', j'$. Then we place $B$ before $B'$ iff:*

(1) *$i < i'$; OR*

(2) *$i = i'$ AND $j < j'$; OR*

(3) *$i = i'$ and $j = j'$ and $\rho_{i,j}(B) < \rho_{i',j'}(B')$*

*that is, we order the resampled bad-events lexicographically by round, sub-round, and then rank $\rho$.*

*Given such an execution log, we may also generate witness trees in the same manner as the sequential algorithm.*

*Now let $\tau$ be any tree-structure; we have*

$$P(\tau \text{ appears}) \leq \prod_{B \in \tau} P_\Omega(B)$$

PROOF. Observe that the choice of swaps for a bad-event $B$ at round $i$, subround $j$, and rank $\rho_{i,j}(B)$, is only affected by the events in earlier rounds / subrounds as well as other $B' \in I_{i,j}$ with $\rho_{i,j}(B') < \rho_{i,j}(B)$.

Thus, we can view this parallel algorithm as simulating the sequential algorithm, with a particular rule for selecting the bad-event to resample. Namely, we keep track of the sets $\mathcal{V}_i$ and $I_{i,j}$ as we do for the parallel algorithm, and within each sub-round we resample the bad-event in $I_{i,j}$ with the minimum value of $\rho_{i,j}(B)$.

This is why it is critical in step (6) that we select $I'_{i,j}$ to be the lexicographically-first MIS; this means that the presence of $B \in I'_{i,j}$ cannot be affected with $B'$ with $\rho(B') > \rho(B)$. □

PROPOSITION 7.20. *Let $B$ be any resampling performed at the $i$th round of the Parallel Swapping Algorithm (that is, $B \in I'_{i,j}$ for some integer $j > 0$.) Then the witness tree corresponding to the resampling of $B$ has height exactly $i$.*

PROOF. First, note that if we have $B \sim B'$ in the execution log, where $B$ occurs earlier in time, and the witness tree corresponding to $B$ has height $i$, then the witness tree corresponding to $B'$ must have height $i + 1$. So it will suffice to show that if $B \in I'_{i,j}$, then we must have $B \sim B'$ for some $B' \in I'_{i-1,j'}$.

At the beginning of round $i$, it must be the case that $\pi^i$ makes the bad-event $B$ true. By Proposition 7.2, either the bad-event $B$ was already true at the beginning of round $i - 1$, or some bad-event $B' \sim B$ was resampled at round $i - 1$. If it is the latter, we are done.

So suppose $B$ was true at the beginning of round $i-1$. So $B$ was an element of $\mathcal{V}_{i-1,1}$. In order for $B$ to have been removed from $\mathcal{V}_{i-1}$, then either we had $B \sim B' \in I'_{i-1,j'}$, in which case we are also done, or after some sub-round $j'$ the event $B$ was no longer true. But again by Proposition 7.2, in order for $B$ to become true again at the beginning of round $i$, there must have been some bad-event $B' \sim B$ encountered later in round $i-1$. □

This gives us the key bound on the running time of the Parallel Swapping Algorithm.

PROPOSITION 7.21. *Suppose that $\epsilon > 0$ and that there is some assignment of weights $\mu : \mathcal{B} \to [0, \infty)$ which satisfies, for every $B \in \mathcal{B}$, the condition*

$$\mu(B) \geq (1+\epsilon) P_\Omega(B) \theta(B)$$

*Then, whp, the Parallel Swapping Algorithm terminates after $\frac{\log n}{\epsilon}$ rounds.*

PROOF. Consider the event that for some $B \in \mathcal{B}$, that $B$ is resampled after $i$ rounds of the Parallel Swapping Algorithm. In this case, $\hat{\tau}$ has height $i$.

In Corollary 3.20, we showed that for the ordinary LLL then whp the witness trees had height $O(\frac{\log s}{\epsilon})$ whp, where $s$ is the number of variables. This proof depended on the fact that the dependency graph could be written as a union of $s$ cliques (one for eac variable). Here, too, the dependency graph can be written as the union of $2(n_1 + \cdots + n_N) = 2n$ cliques: one for each domain variable and one for each range variable. Thus, one can similarly show the witness trees for the parallel swapping algorithm have height $O(\frac{\log n}{\epsilon})$.

□

We can put this analysis all together to show:

THEOREM 7.22. *Suppose $|\mathcal{B}| = n^{O(1)}$ and that for all $B \in \mathcal{B}'$ we have $|B| \leq \log^{O(1)} n$. Suppose also that $\epsilon > 0$ and that there is some assignment of weights*

$\mu : \mathcal{B} \to [0, \infty)$ *which satisfies, for every* $B \in \mathcal{B}$*, the condition*

$$\mu(B) \geq (1 + \epsilon) P_\Omega(B) \theta(B)$$

*Then, whp, the Parallel Swapping Algorithm terminates after* $\frac{\log^{O(1)} n}{\epsilon}$ *time.*

PROOF. The number of rounds, the number of sub-rounds per round, and the running time of each sub-round, are all polylogarithmic in $n$ whp. □

## 7.7. Algorithmic Applications

The LLL for permutations plays a role in diverse combinatorial constructions. Using our algorithm, nearly all of these constructions become algorithmic. We examine a few selected applications now.

**7.7.1. Latin transversals.** Suppose we have an $n \times n$ matrix $A$. The entries of this matrix come from a set $C$ which are referred to as *colors*. A *Latin transversal* of this matrix is a permutation $\pi \in S_n$, such that no color appears twice among the entries $A(i, \pi(i))$; that is, there are no $i \neq j$ with $A(i, \pi(i)) = A(i', \pi(i'))$. A typical question in this area is the following: suppose each color $c$ appears at most $\Delta$ times in the matrix. How large can $\Delta$ be so as to guarantee the existence of a Latin transversal?

In [**35**], a proof using the probabilistic form of the Lovász Local Lemma for permutations was given, showing that $\Delta \leq n/(4e)$ suffices. This was the first application of the LLL to permutations. This bound was subsequently improved by [**17**] to the criterion $\Delta \leq (27/256)n$; this uses a variant of the probabilistic Local Lemma which is essentially equivalent to Pegden's variant on the constructive Local Lemma. Using our algorithmic LLL, we can almost immediately transform the existential proof of [**17**] into a constructive algorithm. To our knowledge, this is the first polynomial-time algorithm for constructing such a transversal.

THEOREM 7.23. *Suppose $\Delta \leq (27/256)n$. Then there is a Latin transversal of the matrix. Furthermore, the Swapping Algorithm selects such a transversal in polynomial time.*

PROOF. For any quadruples $i, j, i', j'$ with $A(i,j) = A(i',j')$, we have a bad-event $(i,j),(i',j')$. Such an event has probability $\frac{1}{n(n-1)}$. We give weight $\mu(B) = \alpha$ to every bad event $B$, where $\alpha$ is a scalar to be determined.

This bad-event can have up to four types of neighbors $(i_1, j_1, i_1', j_1')$, which overlap on one of the four coordinates $i, j, i', j'$; as discussed in [**17**], all the neighbors of any type are themselves neighbors in the dependency graph. Since these are all the same, we will analyze just the first type of neighbor, one which shares the same value of $i$, that is $i_1 = i$. We now may choose any value for $j_1$ ($n$ choices). At this point, the color $A(i_1, j_1)$ is determined, so there are $\Delta - 1$ remaining choices for $i_1', j_1'$.

By Lemma 7.1 and Pegden's criterion [**94**], a sufficient condition for the convergence of the Swapping Algorithm is that

$$\alpha \geq \frac{1}{n(n-1)}(1 + n(\Delta - 1)\alpha)^4$$

Routine algebra shows that this has a positive real root $\alpha$ when $\Delta \leq (27/256)n$.

□

We will encounter this problem again in Chapter 9, where we show that this Swapping Algorithm can be implemented in $O(n)$ time — this is significantly sublinear, note that the input matrix $A$ has size $n^2$.

In [**110**], Szabó considered a generalization of this question: suppose that we seek a transversal, such that no color appears more than $s$ times. When $s = 1$, this is asking for a Latin transversal. Szabó gave similar criteria "$\Delta \leq \gamma_s n$" for $s$ a small constant. Such bounds can be easily obtained constructively using the permutation LLL as well.

By combining the permutation LLL with the partial resampling approach of Chapter 5, we can provide asymptotically optimal bounds for large $s$:

THEOREM 7.24. *Suppose $\Delta \leq (s - c\sqrt{s})n$, where $c$ is a sufficiently large constant. Then there is a transversal of the matrix, in which each color appears no more than $s$ times. This transversal can be constructed in polynomial time.*

PROOF. For each set of $s$ appearances of any color, we have a bad event. We use the partial resampling framework, to associate the fractional hitting set which assigns weight $\binom{s}{r}^{-1}$ to any $r$ appearances of a color, where $r = \lceil \sqrt{s} \rceil$.

We first compute the probability of selecting a given $r$-set $X$. From the fractional hitting set, this has probability $\binom{s}{r}^{-1}$. In addition, the probability of selecting the indicated cells is $\frac{(n-r)!}{n!}$. So we have $p \leq \binom{s}{r}^{-1} \frac{(n-r)!}{n!}$.

Next, we compute the dependency of the set $X$. First, we may select another $X'$ which overlaps with $X$ in a row or column; the number of such sets is $2rn\binom{\Delta}{r-1}$. Next, we may select any other $r$-set with the same color as $X$ (this is the dependency due to $\bowtie$ in the partial resampling framework; see Chapter 5 for more details). The number of such sets is $\binom{\Delta}{r}$.

So the LLL criterion is satisfied if

$$e \times \binom{s}{r}^{-1} \frac{(n-r)!}{n!} \times \left(2rn\binom{\Delta}{r-1} + \binom{\Delta}{r}\right) \leq 1$$

Simple calculus now shows that this can be satisfied when $\Delta \leq (s - O(\sqrt{s}))n$. Also, it is easy to detect a true bad-event and resample it in polynomial time, so this gives a polynomial-time algorithm. □

Our result depends on the Swapping Algorithm in a fundamental way — it does not follow from the LLLL itself (which would roughly require $\Delta \leq (s/e)n$). Hence, prior to this work, we would not have been able to even show the existence of such transversals; here we provide an efficient algorithm as well. To see that our bound is asymptotically optimal, consider a matrix in which the first $s + 1$ rows all contain a

given color, a total multiplicity of $\Delta = (s+1)n$. Then the transversal must contain that color at least $s+1$ times.

**7.7.2. Rainbow Hamiltonian cycles and related structures.** The problem of finding Hamiltonian cycles in the complete graph $K_n$, with edges of distinct colors, was first studied in [**49**]. This problem is typically phrased in the language of graphs and edges, but we can rephrase it in the language of Latin transversals, with the additional property that the permutation $\pi$ has full cycle. How often can a color appear in the matrix $A$, for this to be possible? In [**3**], it was shown that such a transversal exists if each color appears at most $\Delta = n/16$ times.[2] This proof is based on applying the non-constructive Lovász Local Lemma to the probability space induced by a random choice of full-cycle permutation. This result was later generalized in [**41**], to show the following result: if each color appears at most $\Delta \leq c_0 n$ times for a certain constant $c_0 > 0$, then not only is there a full-cycle Latin transversal, but there are also cycles of each length $3 \leq k \leq n$. The constant $c_0$ was somewhat small, and this result was also non-constructive. Theorem 7.25 uses the Swapping Algorithm to construct Latin transversals with essentially arbitrary cycle structures; this generalizes [**41**] and [**3**] quite a bit.

THEOREM 7.25. *Suppose that each color appears at most $\Delta \leq 0.027n$ times in the matrix $A$, and $n$ is sufficiently large. Let $\tau$ be any permutation on $n$ letters, whose cycle structure contains no fixed points nor swaps (2-cycles). Then there is a Latin transversal $\pi$ which is conjugate to $\tau$ (i.e., has the same cycle structure); furthermore the Swapping Algorithm finds it in polynomial time. Also, the Parallel Swapping Algorithm finds it in time* $\log^{O(1)} n$.

[2]The terminology used for rainbow Hamilton cycles is slightly different from that of Latin transversals. In the context of Hamilton cycles, one often assumes that the matrix $A$ is symmetric. Furthermore, since $A(x,y)$ and $A(y,x)$ always have the same color, one only counts this as a single occurrence of that color. Thus, for example, in [**3**], the stated criterion is that the matrix $A$ is symmetric and a color appears at most $\Delta/32$ times.

PROOF. We cannot apply the Swapping Algorithm directly to the permutation $\pi$, because we will not be able to control its cycle structure. Rather, we will set $\pi = \sigma^{-1}\tau\sigma$, and apply the Swapping Algorithm to $\sigma$.

A bad-event is that $A(x, \pi(x)) = A(x', \pi(x'))$ for some $x \neq x'$. Using the fact that $\tau$ has no fixed points or 2-cycles, we can see that this is equivalent to one of the following two situations: (A) There are $i, i', x, y, x', y'$ such that $\sigma(x) = i, \sigma(y) = \tau(i), \sigma(x') = i', \sigma(y') = \tau(i')$, and $x, y, x', y'$ are distinct, and $i, i', \tau(i), \tau(i')$ are distinct, and $A(x, y) = A(x', y')$ or (B) There are $i, x, y, z$ with $\sigma(x) = i, \sigma(y) = \tau(i), \sigma(z) = \tau^2(i)$, and all of $x, y, z$ are distinct, and $A(x, y) = A(y, z)$. We will refer to the first type of bad-event as an event of type A led by $i$ (such an event is also led by $i'$); we will refer to the second type of bad-event as type B led by $i$.

Note that in an A-event, the color is repeated in distinct column and rows, and in a B-event the column of one coordinate is the row of another. So, to an extent, these events are mutually exclusive. Much of the complexity of the proof lies in balancing the two configurations. To a first approximation, the worst case occurs when A-events are maximized and B-events are impossible. This intuition should be kept in mind during the following proof.

We will define the function $\mu$ as follows. Each event of type A is assigned the same weight $\mu_A$, and each event of type B is assigned weight $\mu_B$. The event of type A has probability $(n-4)!/n!$ and each event of type B has probability $(n-3)!/n!$. In the following proof, we shall need to compare the relative magnitude of $\mu_A, \mu_B$. In order to make this concrete, we set

$$\mu_A = 2.83036n^{-4}, \mu_B = 1.96163n^{-3}$$

(In deriving this proof, we left these constant coefficients undetermined until the end of the computation, and we then verified that all desired inequalities held.)

Now, to apply Pegden's criterion [**94**] for the convergence of the Swapping Algorithm, we will need to analyze the independent sets of neighbors each bad-event

can have in the dependency graph. In order to keep track of all the neighborhood structure ssible neighbors, it will be convenient to define the following sums. We let $t$ denote the sum of $\mu(X)$ over all bad-events $X$ involving some fixed term $\sigma(x)$. Let $s$ denote the sum of $\mu(X)$ over all bad-events $X$ (of type either A or B) led by some fixed value $i$, and let $b$ denote the sum of $\mu(X)$ over B-events $X$ alone. Recall that each bad-event of type A is led by $i$ and also by $i'$.

We now examine how to compute the term $t$. Consider a fixed value $x$; we will enumerate all the bad-events that involve $\sigma(x)$. These correspond to color-repetitions involving either row or column $x$ in the matrix $A$. Let $c_i$ (respectively $r_i$) denote the number of occurrences of color $i$ in column (respectively row) $x$ of the matrix, excluding $A(x, y)$ itself.

We can have a color repetition of the form $A(y, x) = A(x, y')$ where $y \neq y'$; or we can have repetitions of the form $A(x, y) = A(x', y')$ or $A(y, x) = A(y', x')$, where $x \neq x', y \neq y'$ (but possibly $x' = y$). The total number of repetitions of the first type is $v_1 \leq \sum_i c_i r_i$. The total number of repetitions of the second type is at most $v_2 \leq \sum_i c_i(\Delta - c_i - r_i)$. The total number of repetitions of the third type is at most $v_3 \leq \sum_i r_i(\Delta - c_i - r_i)$.

For a repetition of the first type, this must correspond to an B-event, in which $\sigma(y) = i, \sigma(x) = \tau(i), \sigma(y') = \tau^2(i)$ for some $i$. For a repetition of the second type, if $x' \neq y$ this correspond to an A-event in which $\sigma(x) = i, \sigma(y) = \tau(i), \sigma(x') = i', \sigma(y') = \tau(i')$ for some $i, i'$ *or* alternatively if $x' = y$ it correspond to a B-event in which $\sigma(x) = i, \sigma(y) = \tau(i), \sigma(y') = \tau^2(i)$ for some $i$. A similar argument holds for the third type of repetition.

Summing all these cases, we have

$$\begin{aligned} t &\leq v_1 n\mu_B + v_2(\max(n^2\mu_A + n\mu_B)) + v_3(\max(n^3\mu_A + n\mu_B)) \\ &\leq v_1 n\mu_B + v_2 n^2\mu_A + v_3 n^2\mu_A \\ &\leq \sum_j (c_j r_j n\mu_B + c_j(\Delta - c_j - r_j)n^2\mu_A + r_j(\Delta - c_j - r_j)n^2\mu_A) \end{aligned}$$

Observe that the the RHS is maximized when there are $n$ distinct colors with $c_j = 1$ and $n$ distinct colors with $r_j = 1$. For, suppose that a color has (say) $c_j > 1$. If we decrement $c_j$ by 1 while adding a new color with $c_{j'} = 1$, this changes the RHS by $(-1 + 2(c_j + r_j) - \Delta)n^2\mu_A + (-1 + \Delta - r_j)n\mu_B \geq 0$.

This gives us

$$t \leq 2n^3\Delta\mu_A$$

Similarly, let us consider $s$. Given $i$, we choose some $y$ with $\sigma(y) = \tau(i)$. Now, we again list all color repetitions $A(x, y) = A(x', y')$ or $A(x, y) = A(y, z)$. The number of the former is at most $\sum_j c_j(\Delta - c_j - r_j)$ and the number of the latter is at most $\sum_j c_j r_j$. As before, this is maximized when each color appears once in the column, leading to

$$s \leq n^3\Delta\mu_A$$

For term $b$, the worst case is when each color appears $\Delta/2$ times in the row and column of $y$; this yields

$$b \leq n^2(\Delta/2)\mu_B$$

Now consider a fixed bad-event A, with parameters $i, i', x, y, x', y'$, and let us count the sum over all independent sets of neighbors, of $\mu$. This could have one or zero children involving $\sigma(x)$ and similarly for $\sigma(y), \sigma(x'), \sigma(y')$; this gives a total contribution of $(1 + t)^4$. The children could also overlap on $i$; the total set of possibilities is either zero children, a B-child led by $i - 2$, a B-child led by $i - 2$ and a child led by $i$, a child led by $i - 1$, a child led by $i - 1$ and a child led by $i + 1$, a child led by $i$, a child led by $i + 1$. There is an identical factor for the contributions of bad-events led by $i' - 2, \ldots, i' + 1$. In total, the criterion for A is that we must have

$$\mu_A \geq \frac{(n-4)!}{n!}(1 + t)^4(1 + b + sb + s + s^2 + s + s)^2$$

Applying the same type of analysis to an event of type B gives us the criterion:

$$\mu_B \geq \frac{(n-3)!}{n!}(1+t)^3(1+b+sb+sb+s+s^2+s^2+s+s^2+s)$$

Putting all these constraints together gives a complicated system of polynomial equations, which can be solved using a symbolic algebra package. Indeed, the stated values of $\mu_A, \mu_B$ satisfy these conditions when $\Delta \leq 0.027n$ and $n$ is sufficiently large.

Hence the Swapping Algorithm terminates, resulting in the desired permutation $\pi = \sigma^{-1}\tau\sigma$. It is easy to see that the Parallel Swapping Algorithm works as well. $\square$

We note that for certain cycle structures, namely the full cycle $\sigma = (123\ldots n{-}1\ n)$ and $n/2$ transpositions $\sigma = (12)(34)\ldots(n-1\ n)$, one can apply the LLLL directly to the permutation $\pi$. This gives a qualitatively similar condition, of the form $\Delta \leq cn$, but the constant term is slightly better than ours. For some of these settings, one can also apply a variant of the MT algorithm to find such permutations [**1**]. However, these results do not apply to general cycle structures, and they do not give parallel algorithms.

**7.7.3. Strong chromatic number of graphs.** Suppose we have a graph $G$, with a given partition of the vertices into $k$ blocks each of size $b$, i.e., $V = V_1 \sqcup \cdots \sqcup V_k$. We would like to $b$-color the vertices, such that every block has exactly $b$ colors, and such that no edge has both endpoints with the same color (i.e., it is a proper vertex-coloring). This is referred to as a *strong coloring* of the graph. If this is possible for *any* such partition of the vertices into blocks of size $b$, then we say that the graph $G$ has strong chromatic number $b$.

A series of papers [**7, 13, 39, 55**] have provided bounds on the strong chromatic number of graphs, typically in terms of their maximum degree $\Delta$. In [**56**], it is shown that when $b \geq (11/4)\Delta + \Omega(1)$, such a coloring exists; this is the best bound currently known. Furthermore, the constant $11/4$ cannot be improved to any number strictly

less than 2. The methods used in most of these papers are highly non-constructive, and do not provide algorithms for generating such colorings.

In this section, we use the permutation LLL to build the strong coloring when $b \geq 256/27\Delta$. This appears to give the first RNC algorithm with a reasonable bound on $b$. Later, in Section 8.3, we develop new bounds on the MT distribution and use these build the strong coloring using independent transversals. While the latter approach requires the weaker condition $b \geq 5\Delta$, it does not give a parallel algorithm.

THEOREM 7.26. *Suppose we have a given graph $G$ of maximum degree $\Delta$, whose vertices are partitioned into blocks of size $b$. Then if $b \geq \frac{256}{27}\Delta$, it is possible to strongly color graph $G$ in expected time $O(n\Delta)$. If $b \geq (\frac{256}{27} + \epsilon)\Delta$ for some constant $\epsilon > 0$, there is an RNC algorithm to construct such a strong coloring.*

PROOF. We will use the permutation LLL. For each block, we assume the vertices and colors are identified with the set $[b]$. Then any proper coloring of a block corresponds to a permutation of $S_b$. When we discuss the color of a vertex $v$, we refer to $\pi_k(v)$ where $k$ is the block containing vertex $v$.

For each edge $f = \langle u, v\rangle \in G$ and any color $c \in [1, \ldots b]$, we have a bad-event that both $u$ and $v$ have color $c$. (Note that we cannot specify simply that $u$ and $v$ have the *same color*; because we have restricted ourselves to *atomic* bad-events, we must list every possible color $c$ with a separate bad event.)

Each bad-event has probability $1/b^2$. We give weight $\mu(B) = \alpha$ to every bad event, where $\alpha$ is a scalar to be determined.

Now, each such event $(u, v, c)$ is dependent with four other types of bad-events:

(1) An event $u, v', c'$ where $v'$ is connected to vertex $u$;

(2) An event $u', v, c'$ where $u'$ is connected to vertex $v$;

(3) An event $u', v', c$ where $u'$ is in the block of $u$ and $v'$ is connected to $u'$;

(4) An event $u', v', c$ where $v'$ is in the block of $v$ and $u'$ is connected to $v'$

There are $b\Delta$ neighbors of each type. For any of these four types, all the neighbors are themselves connected to each other. Hence an *independent* set of neighbors of the bad-event $(u, v, c)$ can contain one or zero of each of the four types of bad-events.

Using Lemma 7.1 and Pegden's criterion [**94**], a sufficient condition for the convergence of the Swapping Algorithm is that

$$\alpha \geq (1/b^2) \cdot (1 + b\Delta\alpha)^4$$

When $b \geq \frac{256}{27}\Delta$, this has a real positive root $\alpha^*$ (which is a complicated algebraic expression). Furthermore, in this case the expected number of swaps of each permutation is $\leq b^2\Delta\alpha^* \leq \frac{256}{81}\Delta$. So the Swapping Algorithm terminates in expected time $O(n\Delta)$. A similar argument applies to the parallel Swapping Algorithm. □

**7.7.4. Hypergraph packing.** In [**77**], the following packing problem was considered. Suppose we are given two $r$-uniform hypergraphs $H_1, H_2$ and an integer $n$. Is it possible to find two injections $\phi_i : V(H_i) \to [n]$ with the property that $\phi_1(H_1)$ is edge-disjoint to $\phi_2(H_2)$? (That is, there are no edges $e_1 \in H_1, e_2 \in H_2$ with $\{\phi_1(v) \mid v \in e_1\} = \{\phi_2(v) \mid v \in e_2\}$. ). A sufficient condition on $H_1, H_2, n$ was given using the LLLL. We achieve this algorithmically as well:

THEOREM 7.27. *Suppose that $H_1, H_2$ have $m_1, m_2$ edges respectively. Suppose that each edge of $H_i$ intersects with at most $d_i$ other edges of $H_i$, and suppose that*

$$(d_1 + 1)m_2 + (d_2 + 1)m_1 < \frac{\binom{n}{r}}{e}$$

*Then the Swapping Algorithm finds injections $\phi_i : V(H_i) \to [n]$ such that $\phi_1(H_1)$ is edge-disjoint to $\phi_2(H_2)$.*

*Suppose further that $r \leq \log^{O(1)} n$ and*

$$(d_1 + 1)m_2 + (d_2 + 1)m_1 < \frac{(1 - \epsilon)\binom{n}{r}}{e}$$

*Then the Parallel Swapping Algorithm finds such injections with high probability in* $\frac{\log^{O(1)} n}{\epsilon}$ *time and using poly*$(m_1, m_2, n)$ *processors.*

PROOF. [**77**] proves this fact using the LLLL, and the proof immediately applies to the Swapping Algorithm as well. We review the proof briefly: we may assume without loss of generality that the vertex set of $H_1$ is $[n]$ and the vertex set of $H_2$ has cardinality $n$ and that $\phi_1$ is the identity permutation; then we only need to select the bijection $\phi_2 : H_2 \to [n]$. For each pair of edges $e_1 = \{u_1, \ldots, u_r\} \in H_1, e_2 = \{v_1, \ldots, v_r\} \in H_2$, and each ordering $\sigma \in S_r$, there is a separate bad-event $\phi_2(v_1) = u_{\sigma 1} \wedge \cdots \wedge \phi_2(v_r) = u_{\sigma r}$. Now observe that the LLL criterion is satisfied for these bad-events, under the stated hypothesis.

The proof for the Parallel Swapping Algorithm is almost immediate. There is one slight complication: the total number of atomic bad-events is $m_1 m_2 r!$, which could be super-polynomial for $r = \Theta(\log n)$. However, it is easy to see that the total number of bad-events *which are true at any one time* is at most $m_1 m_2$; namely, for each pair of edges $e_1, e_2$, there may be at most one $\sigma$ such that $\phi_2(v_1) = u_{\sigma 1} \wedge \cdots \wedge \phi_2(v_r) = u_{\sigma r}$. It is not hard to see that Theorem 7.22 still holds under this condition. □

## 7.A. Symmetry properties of the swapping subroutine

In the following series of propositions, we show a variety of symmetry properties of the swapping subroutine. This analysis will use simple results and notations of group theory. We let $S_l$ denote the symmetric group on $l$ letters, which we identify with the set of permutations of $[l]$. We let $(a\ b)$ denote the permutation (of whatever dimension is appropriate) that swaps $a/b$ and is the identity otherwise. We write multiplications on the right, so that $\sigma\tau$ denotes the permutation which maps $x$ to $\sigma(\tau(x))$. Finally, we will sometimes write $\sigma x$ instead of the more cumbersome $\sigma(x)$.

PROPOSITION 7.28. *The swapping subroutine is invariant under permutations of the domain or range, namely that for any permutations $\tau, \sigma$ we have*

$$P(Swap(\pi; x_1, \ldots, x_r) = \sigma) = P(Swap(\pi\tau; \tau^{-1}x_1, \ldots, \tau^{-1}x_r) = \sigma\tau)$$

*and*

$$P(Swap(\pi; x_1, \ldots, x_r) = \sigma) = P(Swap(\tau\pi; x_1, \ldots, x_r = \tau\sigma)$$

PROOF. We prove this by induction on $r$. The following equivalence will be useful. We can view a single call to Swap as follows: we select a random $x_1'$ and swap $x_1$ with $x_1'$; let $\pi' = \pi \cdot (x_1\ x_1')$ denote the permutation after this swap. Now consider the permutation on $n-1$ letters obtained by removing $x_1$ from the range and $\pi'(x_1)$ from the range of $\pi'$; we use the notation $\pi' - (x_1, *)$ to denote this restriction of range/domain. We then recursively call Swap$(\pi' - (x_1, *), x_2, \ldots, x_r)$.

Now, in order to have Swap$(\pi\tau; \tau^{-1}x_1, \ldots, \tau^{-1}x_r) = \sigma\tau$ we must first swap $\tau^{-1}x_1$ with $x_1' = \tau^{-1}\pi^{-1}\sigma\tau x_1$; this occurs with probability $1/n$. Then we would have

$$\begin{aligned}
&P(\text{Swap}(\pi\tau; \tau^{-1}x_1, \ldots, \tau^{-1}x_r) = \sigma\tau) \\
&\quad = \tfrac{1}{n} P(\text{Swap}(\pi\tau(\tau^{-1}x_1\ \ \tau^{-1}\pi^{-1}\sigma x_1) - (\tau^{-1}x_1, *); \tau^{-1}x_2, \ldots, \tau^{-1}x_r) = \sigma\tau - (\tau^{-1}x_1, *)) \\
&\quad = \tfrac{1}{n} P(\text{Swap}(\pi\tau(\tau^{-1}x_1\ \ \tau^{-1}\pi^{-1}\sigma x_1)\tau^{-1} - (x_1, *); \tau^{-1}\tau x_2, \ldots, \tau^{-1}\tau x_r) = \sigma\tau\tau^{-1} - (x_1, *)) \\
&\qquad \text{by inductive hypothesis} \\
&\quad = \tfrac{1}{n} P(\text{Swap}(\pi(x_1\ \pi^{-1}\sigma x_1)\tau^{-1} - (x_1, *); x_2, \ldots, x_r) = \sigma - (x_1, *)) \\
&\quad = P(\text{Swap}(\pi; x_1, x_2, \ldots, x_r) = \sigma)
\end{aligned}$$

A similar argument applies for permutation of the range (i.e., post-composition by $\tau$). □

Also, the order in which we perform the swaps is irrelevant:

PROPOSITION 7.29. *Let $\pi \in S_n$ be fixed, and let $x_1, \ldots, x_r \in [n]$ be fixed as well. Let $\rho : [r] \to [r]$ be a permutation on $r$ letters; then for any permutation $\sigma \in S_n$ we*

*have*

$$P(Swap(\pi; x_1, \ldots, x_r) = \sigma) = P(Swap(\pi; x_{\rho(1)}, \ldots, x_{\rho(r)} = \sigma)$$

PROOF. We will prove this by induction on $r$. We assume $\rho(1) \neq 1$ or else this follows immediately from induction.

Let $t$ denote the swap $(x_{\rho(1)} \;\; \pi^{-1}\sigma x_{\rho(1)})$, which is the first swap that must occur in $\text{Swap}(\pi; x_{\rho(1)}, \ldots, x_{\rho(r)})$. Then we have:

$$\begin{aligned}
&P(\text{Swap}(\pi; x_{\rho(1)}, \ldots, x_{\rho(r)}) = \sigma) \\
&= \tfrac{1}{n} P(\text{Swap}(\pi t; x_{\rho(2)}, \ldots, x_{\rho(r)}) = \sigma) \\
&= \tfrac{1}{n} P(\text{Swap}(\pi t; x_1, x_2, \ldots, x_{\rho(1)-1}, x_{\rho(1)+1}, \ldots, x_r) = \sigma) \qquad \text{by I.H.} \\
&= \tfrac{1}{n(n-1)} P(\text{Swap}(\pi t(x_1 \;\; (\pi t)^{-1}\sigma x_1); x_2, \ldots, x_{\rho(1)-1}, x_{\rho(1)+1}, \ldots, x_r) = \sigma) \\
&= \tfrac{1}{n(n-1)} P(\text{Swap}(\pi t(x_1 \;\; t\pi^{-1}\sigma x_1); x_2, \ldots, x_{\rho(1)-1}, x_{\rho(1)+1}, \ldots, x_r) = \sigma)
\end{aligned}$$

At this point, consider the following simple fact about permutations: for any $a_1, a_2, b_1, b_2 \in [l]$ with $a_1 \neq a_2, b_1 \neq b_2$, we have

$$(a_2 \; b_2)(a_1 \; (a_2 \; b_2)b_1) = (a_1 \; b_1)(a_2 \; (a_1 \; b_1)b_2)$$

This fact is simple to prove by case analysis considering which of the letters $a_1, a_2, b_1, b_2$ are equal to each other.

We now apply this fact using $a_1 = x_1, a_2 = x_{\rho(1)}, b_1 = \pi^{-1}\sigma x_1, b_2 = \pi^{-1}\sigma x_{\rho(1)}$; this gives us $t(x_1 \;\; t\pi^{-1}\sigma x_1) = (x_1 \;\; \pi^{-1}\sigma x_1)(x_{\rho(1)} \;\; (x_1 \; \pi^{-1}\sigma x_1)\pi^{-1}\sigma x_{\rho(1)})$, and so

$$\begin{aligned}
&P(\text{Swap}(\pi; x_{\rho(1)}, \ldots, x_{\rho(r)}) = \sigma) \\
&= \tfrac{1}{n(n-1)} P(\text{Swap}(\pi(x_1 \;\; \pi^{-1}\sigma x_1)(x_{\rho(1)} \;\; (x_1 \; \pi^{-1}\sigma x_1)\pi^{-1}\sigma x_{\rho(1)}); x_2, \ldots, x_{\rho(1)-1}, x_{\rho(1)+1}, \ldots, x_r) = \sigma) \\
&= \tfrac{1}{n} P(\text{Swap}(\pi(x_1 \; \pi^{-1}\sigma x_1); x_{\rho(1)}, x_2, \ldots, x_{\rho(1)-1}, x_{\rho(1)+1}, \ldots, x_r) = \sigma) \\
&= \tfrac{1}{n} P(\text{Swap}(\pi(x_1 \; \pi^{-1}\sigma x_1); x_2, \ldots, x_r) = \sigma) \qquad \text{by I.H.}
\end{aligned}$$

$= P(\text{Swap}(\pi; x_1, \ldots, x_r) = \sigma)$

□

In our analysis and algorithm, we will seek to maintain the symmetry between the "domain" and "range" of the permutation. The swapping subroutine seems to break this symmetry, inasmuch as the swaps are all based on the *domain* of the permutation. However, this symmetry-breaking is only superficial as shown in Proposition 7.30.

PROPOSITION 7.30. *Define the alternate swapping subroutine, which we denote Swap2*$(\pi; y_1, \ldots, y_r)$ *as follows:*

(1) *Suppose* $\pi$ *is a permutation of* $[n]$*. Repeat the following for* $i = 1, \ldots, r$*:*
(2) *Select* $y_i'$ *uniformly at random among* $[n] - \{y_1, \ldots, y_{i-1}\}$*.*
(3) *Swap entries* $\pi^{-1}(y_i)$ *and* $\pi^{-1}(y_i')$ *of the permutation* $\pi$*.*

*More compactly:*

$$\textit{Swap2}(\pi; y_1, \ldots, y_r) = \textit{Swap}(\pi^{-1}, y_1, \ldots, y_r)^{-1}$$

*Then the algorithms Swap and Swap2 induce the same distribution, namely that if* $\pi(x_1) = y_1, \ldots, \pi(x_r) = y_r$*, then for any permutation* $\sigma$ *we have*

$$P(\textit{Swap}(\pi; x_1, \ldots, x_r) = \sigma) = P(\textit{Swap2}(\pi; y_1, \ldots, y_r) = \sigma)$$

PROOF. A similar recursive definition applies to Swap2 as for Swap: we select $x_1'$ uniformly at random, swap $x_1/x_1'$, and then call Swap2$(\pi(x_1\ x_1') - (*, y_1); y_2, \ldots, y_r)$. The main difference is that we remove the image point $(*, y_1)$ instead of the domain point $(x_1, *)$.

Now, in order to have Swap2$(\pi; y_1, \ldots, y_r) = \sigma$ we must first swap $x_1$ with $x_1' = \pi^{-1}\sigma x_1$; this occurs with probability $1/n$. Next, we recursively call Swap2 on the

permutation $\pi(x_1 x_1') - (*, y_1)$ yielding:

$$P(\text{Swap2}(\pi; y_1, \ldots, y_r) = \sigma)$$

$$= \tfrac{1}{n} P(\text{Swap2}(\pi(x_1 x_1') - (*, y_1); y_2, \ldots, y_r) = \sigma - (*, y_1))$$

$$= \tfrac{1}{n} P(\text{Swap}(\pi(x_1\ x_1') - (*, y_1); (x_1\ x_1')\pi^{-1} y_2, \ldots, (x_1\ x_1')\pi^{-1} y_r) = \sigma - (*, y_1))$$

by inductive hypothesis

$$= \tfrac{1}{n} P(\text{Swap}(\pi - (x_1, y_1); x_2, \ldots, x_r) = \sigma(x_1\ x_1') - (x_1, y_1))$$

by Proposition 7.28, when we pre-compose with $(x_1\ x_1')$

$$= \tfrac{1}{n} P(\text{Swap}((\sigma x_1\ \sigma x_1')\pi - (x_1, *); x_2, \ldots, x_r) = (\sigma x_1\ \sigma x_1')\sigma(x_1\ x_1') - (x_1, *)$$

by Proposition 7.28; when we post-compose with $(\sigma x_1\ \sigma x_1')$

$$= \tfrac{1}{n} P(\text{Swap}(\pi(x_1\ x_1') - (x_1, *); x_2, \ldots, x_r) = \sigma - (x_1, *)$$

$$= P(\text{Swap}(\pi; x_1, \ldots, x_r) = \sigma)$$

□

CHAPTER 8

# Algorithmic and enumerative aspects of the MT distribution

Recall that our definition of the MT distribution as the distribution on variables at the termination of the MT algorithm. A key randomness property of this distribution has been demonstrated in [**48**], namely the critical Theorem 2.16 which we have discussed briefly in Chapter 2. We develop this study further here, showing that the MT distribution has a number of other nice features.

In Section 8.1, we examine the MT distribution for events determined by a single variable, that is, events of the form $X_i \in J$. For such events, one can give better bounds than Theorem 2.16; one can even show *lower-bounds* on the probabilities of such events. We apply this in Section 8.2, to show upper and lower bounds on the weights of independent transversals.

In Section 8.4, we address the problem of partially avoiding bad events, in cases where the LLL criterion is not satisfied. We tighten the bounds of [**48**], giving a symmetric criterion in the case when $epd = \alpha$, for $\alpha \in [1, e]$, as well as, for the first time, an asymmetric criterion. Furthermore, we give a faster parallel algorithm in this case; while applying the parallel MT algorithm directly would give a running time of $O(\frac{\log^3 m}{(1-\alpha)^2})$, we improve this to $O(\frac{\log^2 m}{1-\alpha})$.

Finally in Section 8.5 we estimate the entropy of the MT-distribution, and show that it is close to the original distribution. This automatically implies that there are many more solutions than known before for various problems such as $k$-SAT, non-repetitive coloring, and independent transversals – and especially the maximum-satisfiability variants of these problems.

THEOREM 8.1. *Suppose we are given a vector $\lambda$ satisfying Theorem 1.16. Then, for any atomic event $E \equiv X_{i_1} = j_1 \wedge X_{i_k} = j_k$ we have*

$$P(E \text{ is true in output of MT algorithm}) \leq \prod_{s=1}^{k} \lambda_{i_s,j_s}$$

PROOF. Each independent set of neighbors of $E$ contains one or zero bad-events involving each variable $i_s$, so:

$$\begin{aligned} P_\Omega(E)\theta(E) &\leq P_\Omega(E) \prod_s (1 + \sum_{B': i_s \in S_{B'}} \mu(B')) \\ &\leq \prod_s \frac{\lambda_{i_s,j_s}}{\lambda_{i_s}}(\lambda_{i_s}) \qquad \text{by (37)} \\ &= \prod_s \lambda_{i_s,j_s} \end{aligned}$$

□

### 8.1. The MT distribution for individual variables

Theorem 2.16 only provides an *upper bound* on the probability that $E$ occurs in the output of MT. In general, it is not possible to show a lower bound on the probability of an arbitrary event $E$ — if $E \in \mathcal{B}$, then necessarily $P(E) = 0$. There is one limited circumstance for which one can show a useful lower bound — if the event $E$ has the form $X_i = j$. That is, we can show a useful lower bound on the probability that a variable takes on a particular value. We can also show upper bounds for such events, which are stronger than Theorem 2.16.

In developing such a bound, it is very useful to use the accounting method of Section 1.6, instead of trying to specify a weighting function $\mu : \mathcal{B} \to [0, \infty)$. So, we suppose we are given some vector $\lambda$ which satisfies Theorem 1.16. Namely, there is a vector $\lambda$ such that for each $i \in [n]$ we satisfy

$$\lambda_i - \prod_{B: i \in S_{B'}} \prod_{i',j') \in B} \lambda_{i',j'} \geq 1 \tag{37}$$

where $\lambda_i = \sum_j \lambda_{i,j}$.

For any variable $i$ and value $j$, we define

$$H_{i,j} = \sum_{B \ni (i,j)} \prod_{(i',j') \in B} \lambda_{i',j'}$$

Observe that, with this definition, we can rewrite (37) as:

$$\lambda_i - \sum_j H_{i,j} \geq 1 \tag{38}$$

THEOREM 8.2. *Suppose that $\lambda$ satisfies (37). Let $J$ be a set of assignments to variable $i$. The probability that the MT algorithm ever selects $(i,j)$ for $j \in J$ is bounded by*

$$P(MT \text{ selects } X_i \in J) \leq \frac{\sum_{j \in J} \lambda_{i,j}}{\lambda_i - \sum_{j \notin J} H_{i,j}}$$

PROOF. We consider the first time in which $X_i \in J$ during the execution of the MT algorithm. There are two cases for this occurrence; we may either select $X_i \in J$ initially, or we resample some atomic bad-event $B$ which includes $(i,k)$ amongs its conditions, where $k \in \bar{J}$. Abusing notation somewhat, we write $(i, \bar{J}) \in B$ to denote this condition. We may build a witness tree for this event (although it is not itself a bad-event). This tree contains a root node labeled $X_i = k$. It has either no children, or it has a single child node labeled by such $B$. Below the root node, this tree cannot contain any instances of $X_i = j$ for $j \in J$.

Let $R$ denote the total weight of all tree-structures rooted in some $B \ni (i, \bar{J})$, below which never occurs any instances of $X_i \in J$. To enumerate such trees: root node may have children corresponding to any other variables involved in $B$ (other than $i$ itself); or it may have another child also rooted in such a $B' \ni (i, \bar{J})$. The total weight of all tree-structures rooted in any $B'$ is at most $\mu(B') = \prod_{(i,j) \in B} \lambda_{i,j}$. Thus, a simple induction on tree-height shows that if $r > 0$ satisfies the condition

$$r \geq \sum_{B \ni (i, \bar{J})} P_\Omega(B)(1+r) \prod_{i' \in S_B - \{i\}} (1 + \sum_{B' : i' \in S_{B'}} \mu(B')) \tag{39}$$

then it must be that $R \leq r$.

For any $B \ni (i,j)$, the main term of (39) can be rewritten as

$$\begin{aligned}
&\sum_{B \ni (i,\bar{J})} P_\Omega(B)(1+r) \prod_{i' \in S_B - \{i\}} (1 + \sum_{B' : i' \in S_{B'}} \mu(B')) \\
&\quad = \sum_{B \ni (i,\bar{J})} (1+r) \prod_{(i',j') \in B} \frac{\lambda_{i',j'}}{\lambda_{i'}} \times \prod_{i' \in S_B - \{i\}} (1 + \sum_{B' : i' \in S_{B'}} \mu(B')) \\
&\quad = \sum_{B \ni (i,\bar{J})} \frac{1+r}{\lambda_i} \prod_{(i',j') \in B} \lambda_{i',j'} \\
&\quad = \sum_{B \ni (i,\bar{J})} \frac{1+r}{\lambda_i} \prod_{(i',j') \in B} \lambda_{i',j'} \\
&\quad = \frac{1+r}{\lambda_i} \sum_{j \notin J} H_{i,j}
\end{aligned}$$

Thus, a sufficient condition to satisfy (39) is given by

$$r \geq \frac{1+r}{\lambda_i} \sum_{j \notin J} H_{i,j}$$

which implies that $R \leq \frac{\sum_{j \notin J} H_{i,j}}{\lambda_i - \sum_{j \notin J} H_{i,j}}$. Then:

$$\begin{aligned}
P(\text{MT selects } X_i \in J) &\leq P_\Omega(X_i \in J)(1+R) \\
&= \sum_{j \in J} \frac{\lambda_{i,j}}{\lambda_i} (1 + \frac{\sum_{j \notin J} H_{i,j}}{\lambda_i - \sum_{j \notin J} H_{i,j}}) \\
&= \frac{\sum_{j \in J} \lambda_{i,j}}{\lambda_i - \sum_{j \notin J} H_{i,j}}
\end{aligned}$$

□

A simple corollary of Theorem 8.2 shows a *lower bound* on the probability that the MT terminates with $X_i \in J$:

COROLLARY 8.3. *Suppose that $\lambda$ satisfies (37). Let $J$ be a set of possible assignments to variable $i$. Then:*

$$P(MT \text{ terminates with } X_i \in J) \geq \frac{\sum_{j\in J} \lambda_{i,j} - \sum_{j\in J} H_{i,j}}{\lambda_i - \sum_{j\in J} H_{i,j}}$$

PROOF. Apply Theorem 8.2 to bound from above the probability of ever selecting $X_i \in \bar{J}$:

$$\begin{aligned} 1 - P(X_i \in \bar{J}) &\geq 1 - \frac{\sum_{j\in \bar{J}} \lambda_{i,j}}{\lambda_i - \sum_{j\notin \bar{J}} H_{i,j}} \\ &= 1 - \frac{\sum_{j\notin J} \lambda_{i,j}}{\lambda_i - \sum_{j\in J} H_{i,j}} \\ &= \frac{\sum_{j\in J} \lambda_{i,j} - \sum_{j\in J} H_{i,j}}{\lambda_i - \sum_{j\in J} H_{i,j}} \end{aligned}$$

□

## 8.2. Weighted independent transversals

As an illustration of the Theorem 8.2 and Corollary 8.3, we next study *weighted* transversals, as considered by [**2**]. We give lower- and upper- bounds on the weights of independent transversals, which is not possible using [**54**] or [**94**].

Suppose that we are given weights $w(v) \geq 0$ for each vertex of $G$. There is a simple argument that $G = (V, E)$ has an independent set of weight at least $\frac{w(V)}{\Delta+1}$ and that $G$ has a transversal (not necessarily independent) of weight at least $\frac{w(V)}{b}$. Likewise, $G$ has independent sets or transversals with weight at most $w(V)/(\Delta + 1)$ or $w(V)/b$, respectively. Note also that we cannot always expect independent transversals of weight more (or less) than $w(V)/b$: e.g., consider the case of all weights being equal.

THEOREM 8.4. *Suppose $4\Delta \leq b \leq 4.5\Delta$. Then there is an independent transversal $I \subseteq V$ with weight*

$$w(I) \geq w(V)\Big(\frac{\sqrt{b} + \sqrt{b - 4\Delta}}{\sqrt{b}(2b-1) + \sqrt{b-4\Delta}}\Big) \geq \frac{w(V)}{8\Delta - 1}.$$

*Suppose $b \geq 4.5\Delta$. Then there is an independent transversal $I \subseteq V$ with weight*

$$w(I) \geq w(V) \cdot \min(\frac{1}{b}, \frac{4}{27\Delta - 2}).$$

*Furthermore, independent transversals with weight at least $(1-n^{-\Theta(1)})$ times these lower-bounds, can be found in polynomial time with high probability.*

Proof. The first result follows in a straightforward manner from Theorem 8.2 when we set each entry of $\lambda$ to the scalar constant $\alpha = \frac{b-\sqrt{b}\sqrt{b-4\Delta}}{2b\Delta}$. Now for each vertex $v \in V_i$ we have $H_{i,v} \leq \alpha^2\Delta$; so the probability of selecting this vertex in the LLL distribution is

$$P(\text{select } v) \geq \frac{\lambda_{i,j} - H_{i,v}}{\lambda_i - H_{i,v}} \geq \frac{\alpha - \alpha^2\Delta}{b\alpha - \alpha^2\Delta} = \frac{\sqrt{b} + \sqrt{b-4\Delta}}{\sqrt{b}(2b-1) + \sqrt{b-4\Delta}}$$

To obtain the second result, in each block $V_i$, we discard all but the $b' = \lfloor 9/2\Delta \rfloor$ highest-weight vertices. To simplify the proof, consider only the case when $\Delta$ is even (the odd $\Delta$ is similar).

In this case, we assign $\lambda = \alpha = \frac{1}{3\Delta}$ for each of the $b'$ highest-weight vertices, and $\lambda = 0$ for the remaining. Let us fix a block $V_i$, consisting of vertices $v_1, \ldots, v_b$ sorted by weight so that $w(v_1) \geq w(v_2) \geq \cdots \geq w(v_b)$. By Theorem 8.3, each of the high-weight vertices is selected with probability $\geq \frac{\alpha - \Delta\alpha^2}{b'\alpha - \Delta\alpha^2} = \frac{4}{27\Delta - 2}$. Hence the expected weight of the independent transversal selected is at least $(w(v_1) + \cdots + w(v_{b'}))\frac{4}{27\Delta-2}$. By concavity, subject to a fixed value of $w(V_i)$, the choices of weights $w(v_1), \ldots, w(v_b)$ which minimizes this assigns constant weight $x$ to all vertices, except for an additional vertex which receives an additional weight of $w(V_i) - xb$. The expected weight of this block then becomes

$$\mathbf{E}[w(V_i \cap I)] = x + (w(V_i) - bx)\frac{4}{27\Delta - 2}$$

This achieves its minimum at either $x = 0$ or $x = 1/b$, yielding

$$\mathbf{E}[w(I)] \geq w(V) \cdot \min(\frac{1}{b}, \frac{4}{27\Delta - 2}).$$

Finally, the high-probability bound follows by repetition of the MT algorithm. □

We show a matching upper bound on weights:

THEOREM 8.5. *Suppose* $4\Delta \le b \le 8\Delta$. *Then there is an independent transversal* $I \subseteq V$ *with weight*

$$w(I) \le w(V)\frac{2}{4\sqrt{\Delta}\sqrt{b-4\Delta}+b}$$

*Suppose* $b \ge 8\Delta$. *Then there is an independent transversal* $I \subseteq V$ *with weight*

$$w(I) \le \frac{w(V)}{b}$$

*Furthermore, independent transversals with weight at most* $(1+n^{-\Theta(1)})$ *times these upper-bounds, can be found in polynomial time with high probability.*

PROOF. Suppose we discard all but the lowest-ranking $b'$ vertices in each block, where $b' = 4\Delta$. For these vertices $v$, we set $\lambda_v = \alpha = \frac{1}{2\Delta}$, and we set $\lambda_v = 0$ for the remaining vertices. This satisfies Theorem 1.16.

Fix a block $V_i$, in which the vertices are sorted in increasing order of their weight $w(v_1) \le w(v_2) \le \cdots \le w(v_b)$. Then we can write the expected weight of the resulting block as

$$\begin{aligned}\mathbf{E}[w(V_i \cap I)] &= w(v_1) + (w(v_2)-w(v_1))(1-P(v_1 \text{ selected})) \\ &\quad + (w(v_3)-w(v_2))(1-P(v_1 \text{ or } v_2 \text{ selected})) \\ &\quad + \cdots + (w(v_{b'})-w(v_{b'-1})P(v_{b'} \text{ selected}) \\ &\le w(v_1) + (w(v_2)-w(v_1))(1-\frac{\alpha-\Delta\alpha^2}{b'\alpha-\Delta\alpha^2}) \\ &\quad + (w(v_3)-w(v_2))(1-\frac{2(\alpha-\Delta\alpha^2)}{b'\alpha-2\Delta\alpha^2}) + \dots\end{aligned}$$

Subject to the constraints that $w(v_1) \le w(v_2) \le \dots$ and $w(v_1)+\cdots+w(v_b) = w(V_i)$, the choice of $w(v_1),\dots,w(v_b)$ which maximizes this is the following: for some $2 \le k \le b'$, all the vertices $v_k,\dots,v_b$ have weight $x$, while vertex $v_{k-1}$ has weight

$y \leq x$, and $(b-k+1)x + y = w(V_i)$. In this case, we have

$$\begin{aligned}\mathbf{E}[w(V_i \cap I)] &\leq \max_{x \geq y \in \mathbf{R}, k \in \{2,\ldots,b'\}} yP(v_{k-1}, \ldots, v_{b'} \text{ selected}) + (x-y)P(v_k, \ldots v_{b'} \text{ selected}) \\ &\leq \max_{x \geq y \in \mathbf{R}, k \in \{2,\ldots,b'\}} y\frac{(b'-k+2)\alpha}{b'\alpha - (k-2)\alpha\Delta^2} + (x-y)\frac{(b'-k+1)\alpha}{b'\alpha - (k-1)\alpha\Delta^2}\end{aligned}$$

We now relax the restriction that $k$ is an integer in the range $\{2, \ldots, b'\}$ to allow $k$ to be a real number in the interval $[1, b']$. When $k$ is relaxed in this way, the maximum of the above expression occurs at $y = 0$ and $x = \frac{w(V_i)}{b-k+1}$; we thus have

$$\mathbf{E}[w(V_i \cap I)] \leq \max_{k \in [1,b']} \frac{w(V_i)}{b-k+1} \frac{(b'-k+1)\alpha}{b'\alpha - (k-1)\alpha\Delta^2}$$

When $b \geq 8\Delta$, this is decreasing function of $k$ on the interval $[1, b']$, hence achieves its maximum value at $k = 1$, yielding $\mathbf{E}[w(V_i \cap I)] \leq \frac{1}{b}$. When $4\Delta < b \leq 8\Delta$, this achieves its maximum at the critical point $k = b' + 1 - \frac{\left(\sqrt{b'-4\Delta}+\sqrt{b'}\right)\sqrt{b'(b-b')}}{2\sqrt{\Delta}}$; this yields $\mathbf{E}[w(V_i \cap I)] \leq \frac{2}{4\sqrt{\Delta}\sqrt{b-4\Delta}+b}$. Finally, at $b = 4\Delta$, then we again restrict $k$ to range over the integers; in this case it achieves a maximum value at $k = b$ yielding $\mathbf{E}[w(V_i \cap I)] \leq \frac{2}{1+4\Delta}$.

Putting all these cases together gives us the claimed result. □

Note that such bound cannot be specified in terms of the average degree, because we might add vertices of small degree and weight.

### 8.3. Strong chromatic number revisited

We have seen in Section 7.7.3 how to construct strong colorings using the Swapping Algorithm. In this section, we take an alternate route based on independent transversals. This proof is almost identical to the proof of Theorem 5.3 of [**2**]; the main difference is that we have replaced their nonconstructive application of the LLL with the MT algorithm. This translation requires a non-tivial bound on the MT distribution.

THEOREM 8.6. *Suppose $b \geq 5\Delta$. Then $G$ has a strong coloring, which can be found in expected time $O(n^2\Delta^2)$.*

PROOF. We maintain a *partial coloring* of the graph $G$, in which some vertices are colored with $\{1, \ldots, b\}$ and some vertices are uncolored. Initially all vertices are uncolored. We require that in a block, no vertices have the same color, and no adjacent vertices have the same color.

Now, suppose some color is partially missing from the strong coloring; say without loss of generality there is a vertex $w$ missing color 1. In each block $i = 1, \ldots, k$, we will select some vertex $v_i$ to have color 1. If the block does not have such a vertex already, we will simply assign $v_i$ to have color 1. If the block $i$ *already* had some vertex $u_i$ with color 1, we will swap the colors of $v_i$ and $u_i$ (if $v_i$ was previously uncolored, then $u_i$ will become uncolored).

We need to ensure three things. First, the vertices $v_1, \ldots, v_k$ must form an independent transversal of $G$. Second, if we select vertex $v_i$ and swap its color with $u_i$, this cannot cause $u_i$ to have any conflicts with its neighbors. Third, we insist of selecting $w$ itself for the independent traversal.

A vertex $u_i$ will have conflicts with its neighbors if $v_i$ currently has the same color as one of the neighbors of $u_i$. In each block, there are at least $b - \Delta$ possible choices of $v_i$ that avoid that; we must select an independent transversal among these vertices, which also includes the designated vertex $w$.

Now suppose we define a weighting function which assigns weight 1 to vertex $w$ and weight 0 to the remaining vertices. By Theorem 8.4, the MT algorithm produces an independent transversals of expected weight $\Omega(1/b)$. In particular, one can select an independent transversal containing $w$ by repeating the MT algorithm for $O(b)$ expected times. The total expected cost is $O(n^2\Delta)$.

Whenever we select the independent transversal $v_1, \ldots, v_k$, the total number of colored vertices increases by at least one: for, the vertex $w$ becomes colored while it was not initially, and in every other block the number of colored vertices does not

decrease. So, after $n$ iterations, the entire graph has a strong coloring; the total time is $O(n^2\Delta^2)$. □

### 8.4. Partially avoiding bad events

When the LLL condition is satisfied, then it is possible to select the variables so that *no* bad events occur. Alternatively, if one simply selects the underlying variables from $\Omega$ directly, then each bad event $B$ occurs with probability $P_\Omega(B)$. However, there can be a middle ground. As described in [**48**] even when the LLL condition is violated, one can use the MT-distribution to select the variables so that many fewer bad events occur than one would expect from $\Omega$. For example, if we have $epd = \alpha$, for $\alpha \in [1, e]$, then one can show that it is possible to cause at most $(1 + o(1)) \times mpe \ln(\alpha)/\alpha$ events to occur; here $o(1)$ is parameter which decreases with the dependency $d$ [**48**].

The result of [**48**] is based on the following idea: select each event to be a "core event" independently with probability $q$. These core events will not be allowed to occur; the non-core events are ignored. Each core event has on average $dq$ core neighbors. For $d$ sufficiently large, one can apply Chernoff bounds and the LLL, and show that the number of core neighbors is bounded close to $dq$ in actuality. Now, apply the LLL a second time to avoid the core events, and apply the MT-distribution to show that the non-core events are mostly avoided simply by chance.

While the method of [**48**] is intriguing, it suffers from a few shortcomings. First, the result is asymptotic; there is a second-order term, which is difficult to compute explicitly, and only goes away as $d \to \infty$. Second, this algorithm may be computationally expensive; the first application of the LLL, in particular, may dominate the second, "real" application, and may even be exponential time. Third, one obtains only gross bounds on the total number of true bad events; one cannot easily get more detailed information on the average behavior of a particular bad event.

In this section, we give new bounds and algorithms for partially avoiding bad events, which avoid these problems. In many cases, these algorithms are faster than

the MT algorithm itself. The basic idea parallels [**48**], in that we mark each bad event $B$ as core with probability $q(B)$. However, instead of using two separate LLL phases, we combine them into a single one.

THEOREM 8.7. *Suppose we are given a mapping $\mu : \mathcal{B} \to [0, \infty)$. Then there is an algorithm, which we refer to as the* Truncated MT Algorithm*, which produces a distribution $\Omega'$ on the underlying variables $X_1, \ldots, X_n$, with the property*

$$\forall B \in \mathcal{B}, P_{\Omega'}(B) \leq \max(0, -\mu(B) + P_{\Omega}(B)\theta(B)) \tag{40}$$

*This algorithm has the same running-time behavior as other Moser-Tardos applications. In particular, the expected number of resamplings of a bad event is $\mu(B)$. (Note that the LLL criterion is simply that the RHS of (40) is equal to zero.)*

PROOF. Given our original set of bad events $\mathcal{B}$, we define a new binary variable $Y(B)$ for each bad event, which is Bernoulli-$q(B)$ and which represents that $B$ is "core". We introduce a new set of bad events $\mathcal{B}'$, defined as follows: for each bad event $B \in \mathcal{B}$, we define $B' \in \mathcal{B}'$ to be the event that $B$ is true and $Y(B) = 1$, where we define $q(B) = \min(1, \frac{\mu(B)}{P_{\Omega}(B)\theta(B)})$. The truncated MT algorithm for $\mathcal{B}$ is then defined by running the MT algorithm for $\mathcal{B}'$.

It is not hard to see that the set of bad events $\mathcal{B}'$ satisfies the asymmetric LLL criterion with the weighting function $\mu$.

Now, consider a bad event $B$. In order for $B$ to occur in the output, it must be the case that $Y(B) = 0$. Hence, the probability that $B$ occurs is the probability of the event $(Y(B) = 0) \wedge B$; namely we have

$$P_{\Omega'}(B) = P_{\Omega'}(B \wedge (Y(B) = 0)) \leq (1 - q(B))P_{\Omega}(B)\theta(B) = -\mu(B) + P_{\Omega}(B)\theta(B)$$

as desired. □

This specializes easily to the symmetric setting by setting $\mu(B) = (e/\alpha)^{1/d} - 1$ for all $B$:

COROLLARY 8.8. *Suppose each bad event $B$ has $P_\Omega(B) \leq p, |N(B)| \leq d$; and suppose that $epd = \alpha$ for $\alpha \in [1, e]$. Then one can efficiently find values for the underlying variables so that each bad event $B$ occurs with probability $\leq pe\ln(\alpha)/\alpha$. The total number of expected resamplings is $O(m/d)$.*

As an example of the asymmetric form of this criterion, consider $k$-SAT, in which each variable may appear in up to $L$ clauses in total (positively or negatively). Applying the Lopsided LLL, it is shown in [**43**] that $L \leq \frac{2^{k+1}}{e(k+1)}$ implies that the instance is satisfiable. We prove that this can be relaxed so that the instance is partially satisfiable:

THEOREM 8.9. *Suppose we have a $k$-SAT instance with $m$ clauses, in which each variable appears in up to $L \leq \frac{\alpha 2^{k+1}}{ek} - 2/k$ clauses (in total, either positively or negatively), for $\alpha \in [1, e]$. Then we can construct in expected time $m\log^{O(1)} m$ a truth assignment whose expected number of satisfied clauses is at least $m(1-2^{-k}e\ln(\alpha)/\alpha)$.*

PROOF. We assume that $m \geq 2^{k-1}$ as otherwise a randomly chosen solution will satisfy all the clauses.

Suppose a variable $x_i$ appears in $l_i$ clauses; of these occurrences, it appears $\delta_i l_i$ positively and $(1-\delta_i)l_i$ negatively. Then, as described in [**43**], we set variable $i$ to be T with probability $1/2 - x(\delta_i - 1/2)$, where $x \in [0, 1]$ is a well-chosen parameter. This is quite counter-intuitive. One would think that if a variable occurs positively in many clauses, then one should set the variable to be T with high probability; in fact we do the opposite.

Now, set $\mu(B) = z$ for all bad events $B$, where $z$ is a parameter to be chosen. In this case, it suffices to show that

$$\forall B \in \mathcal{B}, -z + P_\Omega(B)\exp(\sum_{B' \sim B} z) \leq 2^{-k}e\ln\alpha/\alpha \tag{41}$$

It is not hard to show, following [**43**], that for $x = Lz/2$ the LHS here is maximized when variables corresponding to the bad event $B$ each occur in exactly $L/2$ clauses

positively or negatively; and that in this case, we have $P_\Omega(B) = 2^{-k}$, and there are $1 + Lk/2$ neighbors of $B$ in the dependency graph. (The factor of $L/2$ here comes from the Lopsided LLL; namely, clauses that intersect on a variable and agree on it, are not counted as dependent for the purposes of the Lopsided LLL.)

Thus, we set $z = \frac{2\ln\left(\frac{2^{k+1}}{2+kL}\right)}{2+kL}$ and then we have the bound

$$\begin{aligned} -z + P_\Omega(B)\exp(\sum_{B' \sim B} z) &\leq -z + 2^{-k}\exp(z(1 + Lk/2)) \\ &= \frac{2\ln(1 + kL/2) + 2 - k\ln 4}{2 + kL} \\ &= 2^{-k} e \ln(\alpha)/\alpha \end{aligned}$$

Now, the expected number of resamplings is at most $mz \leq m \log^{O(1)} m/L$. For each resampling, we must scan all the affected clauses to see if they have become falsified, which takes time $k^{O(1)}L \leq L \log^{O(1)} m$. Hence the total expected runtime is $m \log^{O(1)} m$. □

We can also apply this criterion for *partial* Latin transversals.[1]

DEFINITION 8.10. *Given an $n \times n$ matrix $A$, a* partial Latin transversal *is a selection of $k \leq n$ cells, at most one in each row and column, with the property that there are no two selected cells with the same color.*

Partial Latin transversals have been most studied in the case when $A$ is a Latin square. In [**109**], Stein analyzes the case of partial Latin transversals for arbitrary matrices. Using techniques from that paper, one can show the existence of partial Latin transversals, whose length is a function of $\Delta$, the maximum number of occurrences of any color. This generalizes [**35**], which showed that if $\Delta$ is sufficiently small, then a full Latin transversal exists.

[1]Note that the Witness Tree Lemma holds for the Swapping Algorithm, and so that algorithm produces distribution which is essentially the same as for the ordinary MT algorithm. All our results work equally for the Swapping Algorithm.

THEOREM 8.11. *Suppose each color appears at most $\Delta = \beta n$ times in the matrix $A$ for $\beta \in [0,1]$. Then one can construct a partial Latin transversal of length at least $n \times \frac{1-e^{-\beta}}{\beta}$.*

PROOF. Suppose that we select a random permutation $\pi$; whenever a color appears more than once in $\pi$, we will remove all but one of those cells from $\pi$ to turn it into a partial Latin transversal.

Suppose that a color appears $d \leq n$ times in the matrix. As shown in [**109**], the probability that $\pi$ meets the color at least once is minimized when all $d$ occurrences of the color are in distinct rows and columns; in this case the probability is (by negative correlation) at least $1 - (1 - 1/n)^d$.

Thus, summing over all colors $i$, the total expected number of colors appearing in $\pi$ is at least

$$\mathbf{E}[\#\text{ distinct colors appearing in } \pi] \geq \sum_i 1 - (1 - 1/n)^{d_i} \tag{42}$$

The expression $1 - (1/n)^d$ is increasing and concave-down as a function of $d$, thus $1 - (1 - 1/n)^d \geq \frac{d}{\Delta}(1 - (1/n)^\Delta$, and so

$$\begin{aligned}\mathbf{E}[\#\text{ distinct colors appearing in } \pi] &\geq \sum_i \frac{d_i}{\Delta}(1 - (1 - 1/n)^\Delta) \\ &= \frac{n^2}{\beta n}(1 - (1 - 1/n)^{\beta n}) \\ &\geq \frac{n^2}{\beta n}(1 - e^{-\beta})\end{aligned}$$

Thus, the resulting partial Latin transversal has an expected length of at least $n(\frac{1-e^{-\beta}}{\beta})$ as we claimed. □

We can improve on Theorem 8.11 for $\beta \leq 0.19$ by using the MT-distribution. (Note that for $\beta \leq 0.105$, the LLL constructs a full Latin transversal.)

THEOREM 8.12. *Suppose each color appears at most $\Delta = \beta n$ times in the matrix $A$, for $\beta \in [0, 1/4]$. Then the truncated MT algorithm runs in expected time $O(n)$ and*

*produces a partial Latin transversal whose expected length is at least*

$$n \cdot \min\Big(1, \frac{1}{2} + \sqrt[3]{\frac{27}{2048\beta}}\Big)$$

PROOF. For every pair of cells $(i,j),(i',j')$ such that $A(i,j) = A(i',j')$, we have a bad event $\pi(i) = j \wedge \pi(i') = j'$.

We apply the criterion of Theorem 8.7, setting $\mu(B) = \alpha$ for each such bad-event, where $\alpha \geq 0$ is a scalar to be chosen. This gives us a probability space $\Omega'$ with the property that, for each $B$, we have

$$\begin{aligned} P_{\Omega'}(B) &\leq \max(0, -\mu(B) + P_\Omega(B)\theta(B)) \\ &\leq \max(0, -\alpha + \frac{1}{n(n-1)}(1 + n(\Delta-1)\alpha)^4 \end{aligned}$$

Now set

$$\alpha = \frac{1}{n(\Delta-1)}\Big(\sqrt[3]{\frac{n-1}{4(\Delta-1)}} - 1\Big)$$

Note that the condition $\Delta \leq n/4$ implies that $\alpha \geq 0$. We then have

$$P_{\Omega'}(B) \leq \max\Big(0, \frac{1 - \frac{3\sqrt[3]{2}\sqrt[3]{n-1}}{8(\Delta-1)^{1/3}}}{n(\Delta-1)}\Big)$$

Now consider the following experiment: we draw the permutation $\pi$ from the space $\Omega'$. For each bad-event that occurs, we de-activate one of the two cells (chosen arbitrarily). Let $Q$ denote the number of active cells at the end of this process; then

$$\mathbf{E}_{\Omega'}[Q] \geq n - \sum_B P_{\Omega'}(B)$$

The total number of bad-events can be computed as follows. First, there are $n^2$ choices for $i,j$. Next, there are $\Delta - 1$ choices for $i',j'$. This double-counts the number of bad-events, so we have

$$\#\text{bad-events} \leq n^2(\Delta-1)/2$$

Thus

$$\begin{aligned}\mathbf{E}_{\Omega'}[Q] &\geq n - \frac{n^2(\Delta-1)}{2} \times \max\Big(0, \frac{1 - \frac{3\sqrt[3]{2}\sqrt[3]{n-1}}{8(\Delta-1)^{1/3}}}{n(\Delta-1)}\Big) \\ &= n \min\Big(1, \frac{1}{2} + \sqrt[3]{\frac{27(n-1)}{2048(\Delta-1)}}\Big) \\ &\geq n \min\Big(1, \frac{1}{2} + \sqrt[3]{\frac{27}{2048\beta}}\Big)\end{aligned}$$

For the analysis of the run-time, see Theorem 9.7 (which describes some optimizations to the Swapping Algorithm for Latin transversals). □

**8.4.1. A faster parallel (RNC) algorithm.** Suppose we wish to use the parallel MT algorithm to draw from the sample space $\Omega'$ such that:

$$\forall B \in \mathcal{B}, P_{\Omega'}(B) \leq \max(0, -\mu(B) + P_{\Omega}(B)\theta(B))$$

In the symmetric setting (with $epd = \alpha$), and using the choice of $\mu$ from Corollary 8.8, one can easily verify that the parallel MT algorithm, as described in [**87**], will terminate with high probability after $O(\frac{\log m}{(\alpha-1)^2})$ rounds. (The approach of [**48**], based on two applications of LLL, will give the same result.) The running time of the parallel MT algorithm is dominated by selecting a maximal independent set (MIS) of true bad events (in this case, with the additional property that $Y(B) = 1$). As finding an MIS requires requires $O(\log^2 m)$ parallel time (using Luby's MIS algorithm), the total runtime of parallel MT would be $O(\frac{\log^3 m}{(\alpha-1)^2})$.

We can improve this running time by only running the parallel MT algorithm for a *constant* number of rounds, using a slightly higher resampling probability than indicated in Theorem 8.7. Unfortunately, we are not able to show a simple condition analogous to the asymmetric LLL for this algorithm to work. Unlike the MT algorithm, which "converges" to a good solution, we give an algorithm which "over-converges" to the desired solution. It reaches a good distribution faster than the MT algorithm, but then it moves away from the good distribution. This algorithm seems

to require a "uniformity" among the bad events, which is by definition true for the Symmetric LLL but seems harder to formalize in general.

We may now define a parallel algorithm corresponding to the Truncated MT Algorithm. It differs from the usual parallel MT algorithm in two key ways. First, we maintain for each bad event $B$ a resampling variable $Y(B)$ which is Bernoulli-$q(B)$, and we only resample bad events (including $Y(B)$ itself) when $Y(B) = 1$. Second, instead of running the algorithm until there are no more true bad events, we run it for some fixed number $t$ of iterations.

THEOREM 8.13. *Suppose we are given a family of weighting functions $\sigma_i : \mathcal{B} \to [0, \infty)$ for $i = 1, \ldots, t+1$ as well as probabilities $q : \mathcal{B} \to [0, 1]$, satisfying the recurrence:*

$$\sigma_1(B) \geq q(B)P_\Omega(B)$$

$$\sigma_{i+1}(B) \geq \sigma_i(B) + q(B)P_\Omega(B) \sum_{\substack{I \subseteq N(B) \\ I \text{ independent}}} \Big[\prod_{B' \in I} \sigma_i(B') - \prod_{B' \in I} \sigma_{i-1}(B')\Big] \qquad \text{for } i = 1, \ldots, t$$

*Then, if the Parallel Truncated MT Algorithm is terminated after $t$ iterations, then each $B$ is true at that point with probability*

$$P(B \text{ true after } t \text{ iterations}) \leq \frac{\sigma_{t+1}(B)}{q(B)} - \sigma_t(B)$$

PROOF. For notational convenience, define $\sigma_0(B) = 0$ for each $B \in \mathcal{B}$.

For each tree-structure $\tau$ whose nodes are labeled $B_1, \ldots, B_s$, define the weight

$$w(\tau) = \prod_{i=1}^{s} q(B_i)P_\Omega(B_i)$$

Let $T_i(B)$ denote the total weight of all tree-structures of height $i$ rooted in $B$, and let $T_{\leq i}(B) = \sum_{j \leq i} T_j(B)$. We claim that

$$T_i(B) \leq \sigma_i(B) - \sigma_{i-1}(B)$$

for $i = 1, \ldots, t$. To show this, we develop a recursive formula for $T_i(B)$, based on decomposing the children into two categories, those of height $i-1$ exactly and those of height $\leq i-2$:

$$\begin{aligned}
T_i(B) &\leq q(B)P_\Omega(B) \sum_{\substack{\mathcal{A}_1,\mathcal{A}_2 \subseteq N(B) \\ \mathcal{A}_1 \neq \emptyset, \mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset \\ \mathcal{A}_1 \cup \mathcal{A}_2 \text{ independent}}} \prod_{B_1 \in \mathcal{A}_1} T_{i-1}(B_1) \prod_{B_2 \in \mathcal{A}_2} T_{\leq i-2}(B_2) \\
&\leq q(B)P_\Omega(B) \sum_{\substack{\mathcal{A}_1,\mathcal{A}_2 \subseteq N(B) \\ \mathcal{A}_1 \neq \emptyset, \mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset \\ \mathcal{A}_1 \cup \mathcal{A}_2 \text{ independent}}} \prod_{B_1 \in \mathcal{A}_1} (\sigma_{i-1}(B_1) - \sigma_{i-2}(B_1)) \prod_{B_2 \in \mathcal{A}_2} \sigma_{i-2}(B_2) \qquad \text{by induction} \\
&= q(B)P_\Omega(B) \sum_{\substack{I \subseteq N(B) \\ I \text{ independent}}} (\prod_{B' \in I} \sigma_{i-1}(B')) - (\prod_{B' \in I} \sigma_{i-2}(B')) \\
&\leq \sigma_i(B) - \sigma_{i-1}(B)
\end{aligned}$$

Now consider the event that bad event $B$ is true after $t$ rounds of the parallel algorithm. We may construct a witness tree for this event; it has height $\leq t+1$. If $Y(B) = 1$ after $t$ rounds, then it must be the case that this tree has height *exactly* $t+1$; for, either $B$ or a neighbor would have been resampled at round $t$. Hence the probability that $B$ remains true after $t$ rounds can be described by either a witness tree of height $t+1$, rooted in $B$; or a witness tree of height $\leq t$, rooted in $(Y(B) = 0) \wedge B$. Furthermore, for every event in the witness tree, other than the root node $B$, we require that $Y(B') = 1$ at the appropriate time. Thus, in total, we have

$$\begin{aligned}
P(B \text{ true after } t \text{ rounds}) &\leq \frac{T_{t+1}(B) + T_{\leq t}(B)(1 - q(B))}{q(B)} \\
&\leq \frac{\sigma_{t+1}(B) - \sigma_t(B) + \sigma_t(B)(1 - q(B))}{q(B)} \\
&= \frac{\sigma_{t+1}(B)}{q(B)} - \sigma_t(B)
\end{aligned}$$

as desired. □

And this specializes to the symmetric setting:

THEOREM 8.14. *Suppose $epd = \alpha$. Then let $\Omega'$ be the distribution induced on the variables after running the Parallel Truncated MT Algorithm for $t = O((\alpha - 1)^{-1})$ steps. In the space $\Omega'$, bad events have probability*

$$P_{\Omega'}(B) \leq pe \ln(\alpha)/\alpha.$$

*The total running time for this procedure is $O(\frac{\log^2 m}{\alpha - 1})$.*

PROOF. We will first show how to select $q$. For all $B \in \mathcal{B}$, define $q(B) = \beta$, for some parameter $\beta$ to be chosen. Define $\sigma_i(B) = \gamma_i$ where $\gamma_i$ is defined recursively as follows:

$$\gamma_0 = 0$$

$$\gamma_{i+1} = p\beta(1 + \gamma_i)^d$$

A simple induction shows that $\gamma_i$ is increasing in $i$:

$$\gamma_{i+1} = p\beta(1 + \gamma_i)^d \geq p\beta(1 + \gamma_{i-1})^d = \gamma_i$$

We claim that this definition of $q, \sigma$ satisfies the conditions of Theorem 8.13. For, we have:

$$\begin{aligned}
&\sigma_i(B) + q(B)P_\Omega(B) \sum_{\substack{I \subseteq N(B) \\ I \text{ independent}}} \prod_{B' \in I} \sigma_i(B') - \prod_{B' \in I} \sigma_{i-1}(B') \\
&\quad \leq \gamma_i + p\beta\Big((1 + \gamma_i)^d - (1 + \gamma_{i-1})^d\Big) \qquad \text{as } |N(B)| \leq d \text{ and } \gamma_{i+1} \geq \gamma_i \\
&\quad = \gamma_{i+1} = \sigma_{i+1}(B)
\end{aligned}$$

Next we claim that for $t$ sufficiently large, there is some $\beta \in [0, 1]$ with

$$\gamma_t = (e/\alpha)^{1/d} - 1 \tag{43}$$

We will show this by continuity. When $\beta = 0$, then the LHS is equal to zero for all $t \geq 1$. As $\beta$ increases, $\gamma_t$ increases for all $t$. Furthermore, when $\beta = 1$, an induction on $t$ shows that

$$\gamma_t \geq p((\alpha/e)(1+1/d)^d)^{t-1}$$

Hence, if $p((\alpha/e)(1+1/d)^d)^{t-1} \geq (e/\alpha)^{1/d} - 1$, then it follows that there is indeed some $\beta \in [0,1]$ satisfying (43). By simple algebraic manipulations, one can verify that this is satisfiable by

$$\begin{aligned} t &= \left\lceil \frac{\ln((e/\alpha)^{1/d} - 1) - \ln\alpha + \ln d + 1}{\ln\alpha + d\ln(1+1/d) - 1} \right\rceil \\ &\leq 1 + \frac{1 - \ln\alpha + \ln(1 - \ln(\alpha))}{\ln\alpha} \leq O(\frac{1}{\alpha - 1}) \end{aligned}$$

Now, the conclusion of Theorem 8.13 applies:

$$\begin{aligned} P(B \text{ true after } t \text{ rounds}) &\leq \frac{\sigma_{t+1}(B)}{q(B)} - \sigma_t(B) \\ &= p(1+\gamma_t)^d - \gamma_t \\ &= p(1 - \alpha^{-1/d}e^{1/d} + \frac{ep}{\alpha}) \\ &\leq pe\ln(\alpha)/\alpha \qquad \text{taking the limit as } p \to 0 \end{aligned}$$

So far, we have shown by continuity that there is *some* choice of $\beta$, for which the parallel MT algorithm would induce $P_{\Omega'}(B) \leq pe\ln(\alpha)/\alpha$. To give a full algorithm, we need to show that it is possible to determine such $\beta$ efficiently.

Recall that $\beta$ is the root of $\gamma_t - (e/\alpha)^{1/d} + 1$ in the range $\beta \in [0,1]$. We can determine this root via numerical bisection: for a putative $\hat{\beta}$, one can recursively compute $\gamma_t$ in time $O(t)$. As each bisection iteration gives one additional bit of precision, we can determine $\beta$ accurately in expected time $O(t\log m) = O(\frac{\log m}{1-\alpha})$. $\square$

### 8.5. Entropy of the MT-distribution

One of the main themes of this paper has been that the MT-distribution has a high degree of randomness, comparable to the randomness of the original distribution $\Omega$. One more quantitative measure of this is the *Rényi entropy* of the MT-distribution.

DEFINITION 8.15 ([**25**]). *Let $\mathcal{V}$ be a distribution on a finite set $S$. We define the* Rényi entropy *with parameter $\rho$ of $\mathcal{V}$ to be*

$$H_\rho(\mathcal{V}) = \frac{1}{1-\rho} \ln \sum_{v \in S} P_{\mathcal{V}}(v)^\rho$$

The entropy of any distribution is at most $\ln |S|$, which is achieved by the uniform distribution, and so $H_\rho$ measures how close a distribution is to uniform. The min-entropy $H_\infty$ is a special case, see, e.g., [**28, 24, 91, 114**] for the centrality of this notion.

It is possible to use the LLL directly for combinatorial enumeration: if one can show that, when drawing from $\Omega$, there is a certain small probability $p$ of avoiding all the bad-events, then it follows that the number of solutions is at least $p|S|$. This principle was used in [**78**], which counted certain types of permutations and matchings in this way. The entropy can also be used as a tool for enumerative combinatorics; namely, if $\Omega'$ is the distribution at the end of the MT algorithm, we know that the total number of solutions (i.e. combinatorial structures avoiding the bad-events) is at least $\exp(H_\rho(\Omega'))$ (for any choice of $\rho$). By analyzing the entropy of the MT-distribution, we achieve "constructively" the "non-constructive" enumerative bounds of the LLL (since each promised solution is output with nonzero probability by MT), and improve upon these bounds as well.

Our main result on the entropy of the MT-distribution is given by:

THEOREM 8.16. *Let $\Omega'$ be the MT-distribution; then for $\rho > 1$ we have*

$$H_\rho(\Omega') \geq H_\rho(\Omega) - \frac{\rho}{\rho - 1} \ln \sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B)$$

PROOF. Consider some event $E$ defined by $X_1 = v_1 \wedge \cdots \wedge X_n = v_n$. By Theorem 2.16, the probability that $E$ occurs at the end of MT is at most $P_\Omega(E)\theta(E)$. Now observe that $\theta(E) \leq \sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B)$.

Letting $x = \sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B)$, we have: Thus, we have

$$\begin{aligned} H_\rho(\mathcal{V}) &= \frac{1}{1-\rho} \ln \sum_v P_{\Omega'}(v)^\rho \\ &\geq \frac{1}{1-\rho} \ln \sum_v (x P_\Omega(v))^\rho \\ &\geq \frac{\rho}{1-\rho} \ln x + \frac{1}{1-\rho} \sum_v P_\Omega(v)^\rho \end{aligned}$$

□

We can think of the term $\sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B)$ as a distortion factor between $\Omega$ and $\Omega'$. The following is a crude but simple estimate of this factor:

PROPOSITION 8.17. *We have*

$$\ln \sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B) \leq \sum_{B \in \mathcal{B}} \mu(B)$$

PROOF. We have

$$\sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B) \leq \sum_{I \subseteq \mathcal{B}} \prod_{B \in I} \mu(B) = \prod_{B \in \mathcal{B}} (1 + \mu(B)) \leq \exp(\sum_{B \in \mathcal{B}} \mu(B))$$

and the claim follows. □

In most applications of the LLL, we keep track of independent sets of bad-events in terms of their variables: namely, for each variable $i$, $I$ can contain at most one

bad-event including $i$. The following result shows how this accounting method yields a better estimate for the entropy:

THEOREM 8.18. *For any bad-event $B$, define*

$$y(B) = (1 + \mu(B))^{\frac{1}{|var(B)|}} - 1$$

*Then we have*

$$\sum_{\substack{I \subseteq \mathcal{B} \\ I \text{ independent}}} \prod_{B \in I} \mu(B) \leq \prod_{i \in [n]} \Big(1 + \sum_{\substack{B \in \mathcal{B} \\ B \text{ involves variable } i}} y(B)\Big)$$

PROOF. We can expand the RHS as a polynomial in the values $y$. Given an independent set $I$, we say that a monomial in the terms $y$ is *supported* on $I$ if, for each $B$, the exponent of $y(B)$ is positive iff $B \in I$. For any independent set $I$, define $q(I)$ to be the sum of all monomials supported on $I$. Thus, for example if $I = \{B\}$ then $q(I)$ is the sum over all terms in RHS of the form $y(B)^j$, for $j > 1$.

The RHS is a positive sum over sets $J \subseteq \mathcal{B}$ of $q(J)$. ($J$ here is not necessarily an independent set.) So, it suffices to show that for any independent $I$ we have

$$\prod_{B \in I} \mu(B) \leq q(I)$$

Now, observe that if $I, I'$ are mutually independent sets of bad-events, then $q(I \cup I') = q(I)q(I')$. The reason for this is any monomial in $q(I)$ corresponds to selecting events $B_{i_1}, \ldots, B_{i_k}$ for variables $i_1, \ldots, i_k$ and a monomial in $q(I')$ corresponds to events $B_{i'_1}, \ldots, B_{i'_{k'}}$, then it must be that $i, i'$ are all mutually distinct (as otherwise $I, I'$ would involve a shared variable), and so the term $i_1, \ldots, i_k, i'_1, \ldots, i'_{k'}$ gives the corresponding term in $q(I)q(I')$. Note that this equality does *not* hold unless $I, I', I \cup I'$ are all independent.

Thus, for any independent set $I$, we have $q(I) = \prod_{B \in I} q(\{B\})$.

So, let us consider some bad-event $B$. To form a monomial involving the term $y(B)$, we must select $B$ as the summand for at least one of the variables involved in

$B$. Each time we select $B$ as a summand, we contribute a factor of $y(B)$, otherwise we contribute a factor of 1. Thus, the total contribution of all such terms is $(1 + y(B))^{|\mathrm{var}(B)|} - 1 = \mu(B)$. (The factor of $-1$ comes from enforcing that we select $B$ at least one time.)

So, we have shown that

$$q(I) = \prod_{B \in I} q(\{B\}) = \prod_{B \in I} \mu(B)$$

and the claim follows.

□

We give an example for independent transversals:

PROPOSITION 8.19. *Suppose we have a graph $G$ of maximum degree $\Delta$, with its vertex set partitioned into $k$ blocks containing $b$ vertices, such that $b \geq 4\Delta$. Suppose we run the MT algorithm to find an independent transversal, using the natural probability distribution (selecting one vertex independently from each block). Then the MT algorithm terminates and the resulting probability space has min-entropy at least*

$$H_\infty(\Omega') \geq k \ln \frac{4b}{2 + b/\Delta - \sqrt{b^2/\Delta^2 - 4b/\Delta}}$$

PROOF. The min-entropy of $\Omega$ is $-\ln b^{-k} = k \ln b$.

It is any easy exercise to see that the asymmetric LLL criterion is satisfied by setting $\mu(B) = \alpha = \frac{(b - \sqrt{b^2 - 4b\Delta})^2}{4b^2\Delta^2}$ for all $B \in \mathcal{B}$. Thus, we have $y(B) = (1 + \alpha)^{1/2} - 1$. So the contribution for each variable $i$ is given by

$$1 + \sum_{B \text{ involves variable } i} y(B) \leq b\Delta \left( \sqrt{\frac{2}{b^{3/2}\sqrt{b - 4\Delta} + b^2 - 2b\Delta} + 1} - 1 \right) + 1$$

Simple analysis shows that this expression is at most

$$\frac{2 + b/\Delta - \sqrt{b^2/\Delta^2 - 4b/\Delta}}{4}$$

Thus, we have

$$H_\infty(\Omega') \geq k \ln b - k \ln\Big(\frac{2 + b/\Delta - \sqrt{b^2/\Delta^2 - 4b/\Delta}}{4}\Big)$$
$$= k \ln \frac{4b}{2 + b/\Delta - \sqrt{b^2/\Delta^2 - 4b/\Delta}}$$

□

We see that the distortion of $\Omega'$ is relatively mild. When $b = 4\Delta$, then the min-entropy is $\leq k(\ln b - \ln 3/2)$. When $b \gg \Delta$, the min-entropy is (up to first order) $k(\ln b - \frac{\Delta}{2b} - \frac{7\Delta^2}{8b^2} - O((\Delta/b)^{5/2})$. By comparison, the cruder Proposition 8.17 would give estimates in these two regimes of, respectively, $k(\ln b - 1/2)$ and $k(\ln b - \frac{\Delta}{2b} - \frac{\Delta^2}{b^2} - O((\Delta/b)^3)$.

Finally, we give an example for partially satisfying $k$-SAT. This is, to our knowledge, the first result to show that not only is the $k$-SAT problem partially satisfiable, but that it has many partial solutions (indeed, exponentially many solutions).

Proposition 8.20. *Suppose we have a $k$-SAT instance with $m$ clauses, in which each variable participates in up to $L \leq \frac{\alpha 2^{k+1}}{ek} - 2/k$ clauses (either positively or negatively), for $\alpha \in [1, e]$. Then there are at least*

$$\frac{2^n}{\exp(\frac{9n}{2k^2})poly(m)}$$

*assignments which satisfy at least $m(1 - 2^{-k} e \ln(\alpha)/\alpha) - 1$ clauses.*

Proof. Suppose we run the MT algorithm as in Theorem 8.9, however using a parameter $\alpha' < \alpha$ instead of $\alpha$, and then compute $H_\rho$ of the resulting distribution; we will now use notation from the proof of Theorem 8.9. We have $\sum_{B \in \mathcal{B}} \mu(B) \leq mz$. Observe that, by double-counting $m \leq nL/k$ and so we have $\sum_{B \in \mathcal{B}} \mu(B) \leq 2n/k^2$.

Next, we compute $H_\rho$ of the original distribution. Each variable is Bernoulli with probability $1/2 + x(1/2 - \delta) \leq \frac{k+1}{2k}$, so we have

$$H_\rho(\Omega) \geq n\Big(\frac{\rho \ln 2 - \log\left(\left(1 + \frac{1}{k}\right)^\rho + \left(1 - \frac{1}{k}\right)^\rho\right)}{\rho - 1}\Big)$$

Thus, we have

$$H_\rho(\Omega') \geq n\Big(\frac{\rho \ln 2 - \ln\left(\left(1+\frac{1}{k}\right)^\rho + \left(1-\frac{1}{k}\right)^\rho\right)}{\rho - 1}\Big) - \frac{\rho}{\rho-1}\frac{2n}{k^2}$$

Now, setting $\rho = 3$, we obtain an entropy of $H_\rho(\Omega') \geq n(\ln 2 - \frac{9}{2k^2}) \geq n(\ln 2 - \frac{9}{2k^2})$.

In the resulting probability distribution, the expected number of failed constraints is $m2^{-k}e\ln(\alpha)/\alpha$. Hence, by Markov's inequality we fail at most $\leq m2^{-k}e\ln(\alpha)/\alpha + 1$ constraints with probability at least $\text{poly}(1/m)$. Thus, the entropy of $\Omega'$ conditioned on this event is at least $n(\ln 2 - \frac{9}{2k^2}) - O(\log m)$. The result follows.

□

CHAPTER 9

# Using the MT distribution to accelerate the search for bad-events

Recall that our definition of the MT distribution as the distribution on variables at the *termination* of the MT algorithm. A key randomness property of this distribution has been demonstrated in [**48**], which we have discussed in Chapter 2. We develop this study further here, showing that the intermediate structures arising in the execution of MT have some very useful "random-like" properties. This internal distribution is quite close to the original sampling distribution $\Omega$, which is just a product distribution.

We will use this observation to speed up one of the key bottlenecks in implementing the MT algorithm. This bottleneck is that, in each iteration of the MT algorithm, one must find some bad event which is currently true (or else certify that there are none.) Implemented directly, this step can be fairly slow, although it is usually polynomial time. We show that this kep step is quite similar to a search problem over a random configuration. Random configurations are often easy to search: for example, while deciding $k$-colorability is NP-hard in general, a simple algorithm of [**71**] solves it for Erdős-Rényi random graphs in expected polynomial time. Thus, he key step of the MT algorithm thus often boils down to detecting a type of configuration in a (nearly) random configuration. This can often be accomplished by *branching algorithms*, in which one gradually builds up a putative true bad event by "guessing" successively more of its state. At every step, one can check whether the partial bad event is extendable to a full bad event, and abort the search if not. Using the randomness of the configuration, one can show that there is a good probability of aborting early.

Sections 9.1 and 9.2 describe the basic algorithms and data structures to implement this idea. Two examples are given, for Ramsey numbers and for hypergraph

2-coloring. They are good representatives of "typical" applications in combinatorics and algorithms. These examples show how these techniques can lead to faster algorithms for nearly all existing applications of the LLL, even those which already have good polynomial-time algorithms. Section 9.3 analyzes how to apply this idea to the permutation LLL setting, and shows that one can obtain the first sub-linear (square-root of input size) algorithms for Latin transversals, a problem of fundamental combinatorial interest. Section 9.4 addresses non-repetitive vertex coloring – one of the few remaining cases where polynomial-time versions of the LLL were not known – and develops such polynomial-time versions.

### 9.1. Fast search for bad events

To implement the MT algorithm, we must search for any bad-events which are currently true (or certify there are none). The simplest way to do this would be to check the entire set $\mathcal{B}$ in each iteration. This will cost $\Omega(m)$ time per iteration (at least). If the bad-events are provided to us an arbitrary list, this is optimal. However, most applications of the LLL have more bad events than variables, and these bad events are much more structured.

Consider the very first step of the MT algorithm, searching for currently-true bad-events. In this case, the variables $X$ are distributed according to $\Omega$, a product distribution. For many problems, one can search random configurations faster (in expectation) than arbitrary configurations. Thus, one should be able to perform the first search step much faster than $O(m)$ time. As the MT algorithm proceeds, the distribution becomes distorted. However, we prove that it does not stray too far from its original distribution. Thus, one can still hope to find bad-events significantly faster on these intermediate distributions than on arbitrary distributions.

**A key ingredient: a search algorithm $S$.** One main ingredient of our algorithms is a problem-specific search algorithm $S$ which given a configuration $X$, determines all the bad-events currently true on $X$. This search procedure may be randomized,

consuming a random source $R$ (which is independent of the random source used to drive the MT algorithm itself). We refer to this as $S(X, R)$.

In many settings, finding a search algorithm which gives good worst-case bounds can be difficult or impossible. However, we will seek to parametrize the run-time of $S$ so that we can analyze its behavior on distributions drawn from the intermediate stages of MT. We thus define an *event-decomposition* for $S$ to be a set of events $A_i$ (not necessarily bad events) and constant terms $c_i$, where $i$ ranges over the integers, with the property that

$$\mathbf{E}_R[\text{Time}(S(X, R))] \leq \sum_i c_i[A_i(X)]. \tag{44}$$

It is important to note in this definition that the expectation is taken only over the random source $R$ consumed by $S$, *not* on the randomness of the MT process itself.

We can now measure the running time of MT as follows:

THEOREM 9.1. *Given an event-decomposition for $S$ as in (44), define*

$$T = \sum_i c_i P_\Omega(A_i)\theta(A_i).$$

*Then,* $\mathbf{E}[\textit{run-time of MT}] \leq (1 + \sum_{B \in \mathcal{B}} \mu(B))T$.

*(Recall the definition of $\theta(E)$ for an event $E$ from Section 2.2.)*

PROOF. Consider a set $A_i$; we wish to estimate the total contribution of the term $c_i[A_i(X)]$ over the total execution of the MT algorithm. This term contributes $c_i$ for each configuration $X$ we encounter in which $A_i(X)$ is true. Every configuration we encounter either corresponds to the original configuration, or the configuration produced after resampling some bad-event $B$.

We can construct a type of witness tree for the latter occurrences. We place a node labeled by $A_i$ at the root and place a child node labeled by $B$ below it. (Note that we do not necessarily have $A_i \sim B$, and so the $B$ would not necessarily have been placed as a child of $A_i$ in the standard method for generating witness trees). We then

go backward in time through the execution log of the MT, placing any resampled bad events in the tree (as children of $A_i$ or $B$ or lower nodes). We may similarly construct a witness tree for the initial configuration; this is a singleton tree containing only the root node labeled $A_i$.

Observe that each resampling produces a distinct witness tree. Hence the total contribution of $A_i$ can be upper bounded by summing over all such tree-structures. To enumerate such trees, one may join any tree-structure rooted in $B$ with any tree-structure rooted in any $A_i$. (This over-counts the weight of all such trees.) By Proposition 2.12, the contribution from $B$ is at most $\mu(B)$ and the contribution from $A_i$ is at most $P_\Omega(A_i)\theta(A_i)$. So the total weight of all such tree-structures is at most $\mu(B)P_\Omega(A_i)\theta(A_i)$. Similarly, the total contribution from the trees corresponding to original configurations is $P_\Omega(A_i) \leq P_\Omega(A_i)\theta(A_i)$. Summing over all $(B, i)$ yields the stated bound. □

**9.1.1. Example: Faster algorithms to construct Ramsey graphs.** A classical result in combinatorics is a lower bound on the diagonal Ramsey number $R(k,k) > \frac{\sqrt{2}}{e}k2^{k/2}$ via the LLL [**11**]. This can be viewed also as an algorithmic challenge: given $k$, two-color the edges of the complete graph $K_n$ for $n = \lceil \frac{\sqrt{2}}{e}k2^{k/2} \rceil$, such that no $k$-clique has all $\binom{k}{2}$ edges of the same color.

PROPOSITION 9.2 (Follows easily from MT). *For $n = \lceil \frac{\sqrt{2}}{e}k2^{k/2} \rceil$, there is an algorithm to construct a two-coloring of $K_n$ avoiding monochromatic $k$-cliques, in expected $2^{k^2/2+o(k^2)}$ time.*

PROOF. For each $k$-clique, there is a bad-event that it is monochromatic; this has probability $p = 2^{1-\binom{k}{2}}$. There are $m = \binom{n}{k} \leq n^k/k!$ cliques, and so the expected number of resamplings is at most $mep$. For each resampling, we check each $k$-clique taking $km$ time. Thus, the total expected time in $O(epkm^2) \leq 2^{k^2/2+o(k^2)}$. □

Although there are exponentially many bad-events in this case, they have a combinatorial structure and it is not necessary to search each bad-event individually.

Rather, we can use a branching to enumerate the cliques, as we did in Section 4.3.5. However note that in Section 4.3.5 it was only necessary to analyze the *initial* configuration, not the intermediate ones.

PROPOSITION 9.3. *There is a deterministic search algorithm $S$ for monochromatic $k$-cliques with an event decomposition*

$$Time(S(X)) = n^{O(1)} \sum_{i=2}^{k} \sum_{i\text{-cliques } I} [I \text{ is monochromatic on } X]$$

PROOF. We recursively enumerate all $i$-cliques, for $i = 2, \ldots, k$. Initially, every edge is a monochromatic 2-clique. Next, for each monochromatic $i$-clique $I$, we test all possible vertices $v$ and check if $I \cup \{v\}$ is also monochromatic. The term $n^{O(1)}$ here accounts for searching over the vertex $v$ as well as any operations involving testing whether $I \cup \{v\}$ is monochromatic. □

PROPOSITION 9.4. *For $n = \lceil \frac{\sqrt{2}}{e} k 2^{k/2} \rceil$, there is an algorithm to construct a two-coloring of $K_n$ avoiding monochromatic $k$-cliques, in expected $2^{k^2/8+o(k^2)}$ time.*

PROOF. We apply Theorem 9.1 to the event-decomposition of Proposition 9.3. We have:

$$\begin{aligned}
T &\leq n^{O(1)} \sum_{i=2}^{k} \sum_{i\text{-cliques } I} P_\Omega(I \text{ is monochromatic on } X)\theta(I \text{ is monochromatic on } X) \\
&\leq n^{O(1)} \sum_{i=2}^{k} \sum_{i\text{-cliques } I} 2^{1-\binom{i}{2}} \exp(epN(I)) \\
&\leq n^{O(1)} \sum_{i=2}^{k} n^i 2^{-\binom{i}{2}} \exp(epi^2 n^{k-2}/(k-2)!) \\
&\leq n^{O(1)} \max_{i\in[2,k]} n^i 2^{-\binom{i}{2}} \exp(epi^2 n^{k-2}/(k-2)!) \\
&\leq 2^{k^2/8+o(k^2)}
\end{aligned}$$

Now, $\sum_B \mu(B) \leq mep = 2^{O(k)}$. Hence by Proposition 9.1 the overall run-time of MT is $2^{k^2/8+o(k^2)}$. □

This is a polynomial improvement over Proposition 9.2, roughly reducing the time to the fourth root.

## 9.2. Depth-first-search Moser-Tardos

As we have seen, the main cost in the MT algorithm is to search for any bad-events which are currently true (or certify there are none). The simple way to do this, as we have discussed in Section 9.1, is to check the entire set $\mathcal{B}$ in each iteration. This is rather wasteful; a small optimization, suggested by [**105**], is to maintain a stack which records all the currently-true bad-events. At the very beginning of the MT algorithm, we scan the entire set $\mathcal{B}$ to find all the true bad-events. Whenever we resample a bad-event $B$, we only need to check its neighbors to determine whether they became true (and if so, we add them to the stack); we do not need to search the entire space. This can potentially improve the runtime of MT by a factor of $n$. We refer to this as a "depth-first-search" MT. Using this method, we must expend $O(d)$ work after each each resampling (assuming that it requires unit time to check a bad-event). As the expected number of resamplings overall is $O(m/d)$, this gives a total expected running time $O(m)$. If the bad-events are simply provided to us as an arbitrary list, this is already optimal.

For applications with complex bad-events, we can speed up the depth-first search strategy by taking advantage of the random nature of the MT-distribution. We can hope to design a search algorithm which takes as input a configuration of variables, *and a bad-event* $B$ , and lists all of the bad events $B' \sim B$ which hold in it.

**A key ingredient: data structure** $D$**.** One main ingredient of our algorithms is a problem-specific data-structure $D$ which, given a bad event $B$ and configurations $X', X$ before and after a resampling of $B$, can determine all the bad events $B' \sim B$ which are true in $X$; we refer to this as $D(B, X', X)$. This data-structure also requires an initialization step, in which given a configuration $X$ we find *all* bad events currently true in it; we refer to this as $D(\emptyset, X)$. (Initialization is typically much cheaper and

simpler than the updating step, and is only performed once, so we mostly ignore it in our analyses.)

Informally speaking, we would like define an *event-decomposition* for $D$ to be a set of events $A_{B,i}$ (not necessarily bad events) and constant terms $c_{B,i}$, where $B \in \mathcal{B}$ and $i$ ranges over the integers, with the property that

$$\text{Time}(D(B, X', X)) \leq \sum_i c_{B,i}[A_{B,i}(X)].$$

However, we will want to allow amortized run-time bounds and randomized data structures. We suppose that $D$ uses a random source $R$ (which is independent of the randomness used to drive the MT algorithm itself). We also suppose that we have an amortized expected-run-time guarantee. This leads to the following (somewhat intimidating looking) Theorem 9.5:

THEOREM 9.5. *Suppose that we are given an event-decomposition* $\{c_{B,i}, A_{B,i} \mid B \in \mathcal{B}\}$*, which satisfies the following condition: "for all sequences of configurations* $X_1, \ldots, X_{t+1}$*, and any resampled bad events* $B_1, \ldots, B_t$ *which are true in configurations* $X_1, \ldots, X_t$ *respectively, we have*

$$\mathbf{E}_R\Big[\sum_{j=1}^{t} Time(D(B_j, X_j, X_{j+1}))\Big] \leq \sum_{j=1}^{t} \sum_i c_{B_j,i}([A_{B_j,i}(X_j)] + [A_{B_j,i}(X_{j+1})])."$$

*For each event* $B$*, define* $T_B = \sum_i c_{B,i} P_\Omega(A_{B,i})\theta(A_{B,i})$.

*Then, the expected run-time of the MT algorithm,* exclusive of the initialization of the data-structure $D$ itself*, is at most* $\sum_{B \in \mathcal{B}} \mu(B) T_B$.

PROOF. This is almost identical to Theorem 9.1 and is omitted. □

**9.2.1. Example: hypergraph two-coloring.** We consider a more technically involved example. Suppose we are given a $k$-uniform hypergraph with $m$ hyper-edges, and we wish to find a two-coloring of the vertices so that no edge is monochromatic. Suppose each edge intersects with $\leq L$ others. When $L \leq 0.17\sqrt{\frac{k}{\ln k}}2^k$, MT can be applied to the approach of [**95**] to find a good coloring. Straightforward analysis of

this algorithm would indicate a running time $mL \cdot \text{poly}(k)$; potentially, a quadratic-time algorithm. We reduce this to $m \log^{O(1)} m$ time:

THEOREM 9.6. *Suppose, in a $k$-uniform hypergraph, there are $m$ edges and each edge intersects at most $L \leq 0.17\sqrt{\frac{k}{\ln k}}2^k$ other edges, and $k$ is sufficiently large. Then a non-monochromatic two-coloring can be found in expected time $m \log^{O(1)} m$.*

PROOF. We present a version of the algorithm of [**95**] to find such a coloring: first, each vertex chooses a color at random. Next, we choose a random ordering of the vertices (equivalently, each vertex independently chooses a random rank $\rho_v \in [0, 1]$). For each vertex $v$ in this order, we look for any monochromatic edges of which $v$ is the lowest-ranking vertex. If we find any such edge, we flip the color of $v$.

It is easy to implement this procedure in time $O(m)$, but the probability it succeeds can be very low when $m \gg L$. We we will assume that $m \geq \Omega(\sqrt{\frac{k}{\log k}}2^k)$; otherwise, as shown in [**95**], then this algorithm produces a good coloring with probability $\Omega(1)$.

This procedure fails to produce a valid coloring only if the following occurs. There is some edge $f$, originally colored blue (w.l.o.g.), and vertex $v \in f$ is the lowest-ranking vertex of $f$. There is another edge $f'$, which intersects $f$ in exactly $v$, with the property that all other vertices in $f'$ are either red or have rank lower than $v$. In that case, it is possible that all the originally blue vertices in $f'$ are flipped, becoming red. This type of monochromatic edge will remain in the final coloring.

Each vertex has two variables associated with it: its (original) color and its rank $\rho_v$. We use the MT algorithm to select both values.

We will translate this into the LLL framework in a somewhat unusual way. We define a bad event $B^{\text{blue}}(f, f')$ to mean that the above event occurred *and the minimum-ranking vertex in $f$ had rank $\leq R$*, where $R = \frac{\ln k}{2k}$. We define a bad event $B^{\text{blue}}(f)$ to mean that edge $f$ was originally blue and *all vertices in it had rank $> R$*. We similarly define $B^{\text{red}}(f)$ and $B^{\text{red}}(f, f')$. Note that the algorithm fails iff one of the four types of bad events occurs. The reason we are distinguishing the two cases of

the minimum-ranking vertex in $f$, is that when this rank is large, then fixing $f$ will typically break many $f'$; so it is not beneficial to take a union-bound over all such $f'$.

We now use the asymmetric LLL. For an event $B(f)$, we assign $\mu(B(f)) = \sqrt{e}p_1$ and for an event $B(f, f')$ we assign $\mu(B(f, f')) = ep_2$, where $p_1 = P_\Omega(B(f)), p_2 = P_\Omega(B(f, f'))$.

Let us first compute $p_1$. For an event $B^{\text{blue}}(f)$, it must occur that all the vertices in $f$ are blue and have rank $> R$; this occurs with probability $p_1 = 2^{-k}(1-R)^k$.

Next, let us compute $p_2$. Suppose $f, f'$ intersect in $v$. For an event $B^{\text{blue}}(f, f')$, it must occur that all vertices in $f$ are blue; this occurs with probability $2^{-k}$. All the vertices in $f$, other than $v$, must have rank exceeding that of $v$; this occurs with probability $(1-\rho_v)^{k-1}$. All the vertices in $f'$, other than $v$, must be either red or have rank less than $v$; this occurs with probability $(1/2 + 1/2\rho_v)^{k-1}$. Hence, integrating over $\rho_v \in [0, R]$, we have

$$\begin{aligned} p_2 &\leq \int_{\rho_v=0}^{R} d\rho_v \; 2^{-k}(1-\rho_v)^{k-1}(1/2+1/2\rho_v)^{k-1} \\ &= 2^{1-2k}\int_{\rho_v=0}^{R} d\rho_v \; (1-\rho_v)^{k-1}(1+\rho_v)^{k-1} \\ &\leq 2^{1-2k}R \end{aligned}$$

Finally, we need to analyze the dependency. Consider an edge $f$; we want to compute $t = \prod_B(1+\mu(B))$ where $B$ ranges over all bad events touching $f$. One can verify there are at most $2L$ events of type $B(f')$ (one for each color) and at most $4L^2$ events of $B(f', f'')$ (either $f'$ or $f''$ could touch $f$, and there are two possible colors). Hence we have

$$t \leq (1+\sqrt{e}p_1)^{2L}(1+ep_2)^{4L^2} \leq \exp(2L\sqrt{e}p_1 + 4L^2ep_2)$$

The LLL criterion is now

$$p_1\sqrt{e} \geq p_1 t \qquad p_2 e \geq p_2 t^2$$

which can be seen to be satisfied for $L \leq 0.17\sqrt{\frac{k}{\ln k}}2^k$ and $k$ sufficiently large. Furthermore, for any edge $f$, we have $t \leq O(1)$.

We now discuss how to *find* bad events efficiently. For this, we will need the data-structure $D$ to track the following information: for each vertex $v$, we list all *monochromatic* edges which contain $v$. When an edge was originally monochromatic and was resampled, we delete it from the $k$ corresponding lists; that takes time $O(k)$. When an edge becomes monochromatic, we add it to the $k$ corresponding lists, again in time $O(k)$.

Now supposing we have this list of monochromatic edges; we show how to find the bad events when we resample some edge $f$. To find an event of type $B(f')$ affected by $f$, we simply loop over all the monochromatic edges $f'$ intersecting $f$, and check if they also satisfy the property that $\rho(w) \geq R$ for all $w \in f'$; this takes time $\sum_{f' \text{ touches } f} k^{O(1)}$.

Next, we search for events $B(f', f'')$, where $f'$ touches $f$: we begin by looping over all edges $f'$ touching $f$. For each monochromatic $f'$, we then loop over all edges $f''$ touching $f'$. The total work for this is

$$\sum_{f' \text{ touches } f} k^{O(1)} + \sum_{f' \text{ touches } f} [f' \text{ is monochromatic}] \sum_{f'' \text{ touches } f'} k^{O(1)}$$

Finally, consider how to find an event $B(f', f'')$, where now $f''$ touches $f$. We begin by looping over edges $f''$. Next, we use our data-structure to *efficiently* list all monochromatic $f'$ touching $f''$. The work for this is

$$k^{O(1)} + \sum_{f'' \text{ touching } f} [f'' \text{ is monochromatic}] \sum_{f' \text{ touches } f''} k^{O(1)}$$

Putting all these terms together, we can write the work factor for the data-structure in terms of an event decomposition. Namely, if we resample a bad event of

type $B(f)$, then the work expended searching for neighbors of $f$ is

$$\begin{aligned}\text{Time} &\le k^{O(1)}\Big(\sum_{f' \text{ touches } f} 1 + \sum_{f' \text{ touches } f} [f' \text{ is monochromatic}] \sum_{f'' \text{ touches } f'} 1\Big) \\ &\le Lk^{O(1)}\Big(1 + \sum_{f' \text{ touches } f} [f' \text{ is monochromatic}]\Big)\end{aligned}$$

This is precisely the form required for the event-decomposition in Theorem 9.5. Thus, we have

$$\begin{aligned}T_{B(f)} &\le Lk^{O(1)}\Big(1 + \sum_{f' \text{ touches } f} P_\Omega(f' \text{ is monochromatic})\theta(\{B' \mid B' \text{ touches } f\})\Big) \\ &\le Lk^{O(1)}\Big(1 + \sum_{f' \text{ touches } f} 2^{-k} \prod_{B' \text{ touches } f} (1+\mu(B'))\Big) \\ &\le Lk^{O(1)}(1 + L2^{-k}t) \le Lk^{O(1)}\end{aligned}$$

and hence the total expected work for this bad event $B(f)$, over the entire execution of MT, is at most $\mu(B(f))T_B \le p_1\sqrt{e}Lk^{O(1)} \le k^{O(1)}$; summing over all edges $f$ gives a total time of $mk^{O(1)}$.

Similarly, one can easily see that the work to find bad events $B' \sim B(f, f')$, is almost $T_{B(f,f')} \le Lk^{O(1)}$. Hence the total expected work for such events, over the execution of MT, is at most

$$\begin{aligned}\text{Work for events } B(f,f') &\le \sum_{f,f'} \mu(B(f,f'))T_{B(f,f')} \\ &\le \sum_{f \text{ touches } f'} ep_2 Lk^{O(1)} \\ &\le mLep_2 \times Lk^{O(1)} \le mk^{O(1)}\end{aligned}$$

It is easy to see that one can initialize in the same time, namely $mk^{O(1)}$. Recalling that $k = \log^{O(1)} m$, this proves the theorem. □

### 9.3. Latin transversals revisited

Recall the problem of finding a Latin transversal of an $n \times n$ matrix $A$. We discussed in Section 7.7.1 how the Swapping Algorithm could be applied as long as each color appears at most $\Delta \leq (27/256)n$ times in the matrix. We can apply our analysis of the MT distribution in order to accelerate that algorithm, reducing the running time to $O(n)$.[1]

THEOREM 9.7. *Suppose each color appears at most $\Delta \leq (27/256)n$ times in the matrix $A$. Then there is an algorithm to find a Latin transversal in expected time $O(n)$ assuming that we have fast read access to the matrix, namely:*

(A1) *The entries of $A$ allow random-access reads.*

(A2) *The colors of $A$ can be represented as bit-strings of length $O(\log n)$.*

(A3) *Our algorithm can perform elementary arithmetic operations on words of size $O(\log n)$ in time $O(1)$.*

*Note that the input size to the problem is $\Theta(n^2)$.*

PROOF. Each bad-event $B$ has probability $p = \frac{1}{n(n-1)}$. We have seen in Section 7.7.1 that the asymmetric LLL criterion holds with these parameters with $\mu(B) = O(p)$. For any $x, y \in [n]$ define

$$\theta(x,y) = \theta\Big(\{B \in \mathcal{B} | B \text{ involves } x \text{ or } y\}\Big)$$

Thus $\theta(x,y) = \prod_B (1+\mu(B))$, where the product is taken over all bad events involving $x$ or $y$. There are $O(n^2)$ such bad events, and for each such bad event $B$ we have $\mu(B) = O(n^{-2})$, so in total $\theta(x,y) = O(1)$.

Now consider the following data-structure $D$. We first randomly choose some 2-independent hash function $H$, uniformly mapping the labels of colors to the set

[1]We note that Sections 9.1 and 9.2 were stated in terms of the ordinary MT algorithm. However, it is not hard to see that all this analysis dependend solely on the Witness Tree Lemma, which we have seen holds for the Swapping Algorithm as well. Thus, all the results of those two sections hold equally for the Swapping Algorithm.

$[n]$. We will maintain a list, for each $t \in [n]$, of all pairs $(x, y)$ with $\pi(x) = y$ and $H(A(x, y)) = t$. These can be maintained with a doubly-linked-list for each element $t \in [n]$ in the range of $H$. We will update this structure during the execution of the MT algorithm; for example, if $\pi(x) = y$ and we resample so that $\pi(x) = y'$, we would remove the pair $(x, y)$ from the list corresponding to $H(A(x, y))$ and add the pair $(x, y')$ to the list corresponding to $H(A(x, y'))$. It is not hard to see how to add and remove pairs from their appropriate list in constant time.

Now consider the work required in a single step of $D(B, X', X)$. The operation of adding and removing pairs from their corresponding linked-lists takes $O(1)$ time. The costly operation is that, for each affected entry $x$ in the permutation, we must loop over all pairs $x, x'$ with $H(A(x, \pi(x))) = H(A(x', \pi(x')))$ and test whether $A(x, \pi(x)) = A(x', \pi(x'))$. If the latter holds, then we have detected a new bad event.

Thus, suppose we resample $B = (\pi(x_1) = y_1) \wedge (\pi(x_2) = y_2)$, obtaining the new permutation $\pi'$. There are four positions in the permutation $\pi'$ that differ from $\pi$, and we must test each of these to see if there are new bad events. We thus have:

Time for $D$ updating $B =$

$$\sum_{y_1' \in [n]} \sum_{\substack{x_3 \neq x_1 \\ y_3 \neq y_1'}} \Big[\pi'(x_1) = y_1' \wedge \pi'(x_3) = y_3 \wedge H(A(x_1, y_1')) = H(A(x_3, y_3))\Big] + \dots$$

(Here, we have only written one of the four summands, corresponding to new bad events involving the $\pi(x_1) = y_1'$. The other three summands are analogous, and will have the same cost.)

By 2-independence of $H$, we have:

$$\text{Expected Time for } D(B, X', X) =$$

$$1/n \sum_{y_1' \in [n]} \sum_{\substack{x_3 \neq x_1 \\ y_3 \neq y_1'}} \Big[\pi'(x_1) = y_1' \wedge \pi'(x_3) = y_3\Big]$$

$$+ \sum_{y_1' \in [n]} \sum_{\substack{x_3 \neq x_1 \\ y_3 \neq y_1'}} \Big[\pi'(x_1) = y_1 \wedge \pi'(x_3) = y_3 \wedge A(x_1, y_1') = A(x_3, y_3)\Big] + \ldots$$

(This expectation is taken over the hash function $H$, *not* on any of the random choices during the MT algorithm. Thus, the permutations $\pi, \pi'$, should be viewed as fixed values and not random variables.)

We can now apply Theorem 9.5 to calculate:

$$T_B = 1/n \sum_{y_1', x_3 \neq x_1, y_3 \neq y_1'} P_\Omega[\pi'(x_1) = y_1' \wedge \pi'(x_3) = y_3]\theta(x_1, y_1')\theta(x_3, y_3)$$

$$+ \sum_{\substack{y_1', x_3 \neq x_1, y_3 \neq y_1' \\ A(x_1, y_1') = A(x_3, y_3)}} P_\Omega[\pi'(x_1) = y_1' \wedge \pi'(x_3) = y_3]\theta(x_1, y_1')\theta(x_3, y_3) + \ldots$$

$$= n^3 \times 1/n \times \frac{1}{n(n-1)} \times O(1) + n^2 \times \frac{1}{n(n-1)} \times O(1) + \ldots$$

$$= O(1)$$

We can perform a similar calculation that shows $O(n)$ time to initialize $D$. So, by Theorem 9.5, the expected running time of MT is

$$O(n) + \sum_B \mu(B) T_B \leq O(n) + O(1) \sum_{\substack{x, y, x', y' \\ A(x,y) = A(x', y')}} \mu(x, y, x', y') = O(n).$$

as desired. □

### 9.4. Non-repetitive vertex coloring: from exponential to polynomial

So far, we have examined problems in which good data structures can lead to polynomial improvements in the MT runtime. However, Theorems 9.1, 9.5 can be much more powerful, and can indeed transform exponential-time algorithms to polynomial-time ones. We will consider a series of related problems based on *non-repetitive vertex coloring* of graphs. These represent one of the few remaining cases in which the Lovász Local Lemma provides a proof of existence without a corresponding polynomial-time algorithm. We will remedy this for these problems.

Given a graph $G$, we seek to color its vertices so that no color sequence appears repeated in any vertex-simple path; i.e., there is no simple path colored $xx$, where $x$ can denote any nonempty sequence of colors. How many colors are needed in order to ensure such a coloring exists? This is known as the *Thue number* $\pi(G)$ of $G$, motivated by Thue's classical result that $\pi$ is at most 3 for paths of any length [**113**].[2]

The problem of determining non-repetitive colorings and Thue numbers have been studied extensively in a variety of contexts. In [**10**], it was shown via the LLL that for any graph $G$ with maximum degree $\Delta$, $\pi(G) = O(\Delta^2)$. The original constant term in that paper was not tight; a variety of further papers such as [**46, 47, 50**] have brought it down further. The best currently-known bound is that $\pi(G) \leq (1 + o(1))\Delta^2$ [**32**]. The analysis of [**32**] does not use the LLL; it uses a *non-constructive* Kolmogorov-complexity argument which is somewhat complicated and specialized to the graph-coloring problem.

While the MT resampling framework applies to this problem, the key bottleneck is to either find a bad event (a path with repeated colors), or to certify that none such exists. In this case, the number of bad events is exponentially large; more seriously,

[2]There are a few variants on this definition such as whether the edges or vertices are colored, and whether each has its own palette of colors or whether there is a common palette. For concreteness, we color vertices from a common palette; all of our bounds would apply to the other scenarios as well. We assume that the graph $G$ is simple with $2 \leq \Delta \leq n - 1$.

it is NP-hard to even detect whether a given coloring has a repeated color sequence [**82**]. So, in fact this is a situation in which it is *intractable* to find a data-structure with good *worst-case* run-time bounds.

In [**48**], a constructive algorithm was introduced using $C = \Delta^{2+\epsilon}$ colors (i.e., if a slack $\Delta^{\epsilon}$ is allowed). The basic idea of [**48**] is to apply the MT algorithm, but to ignore the long paths. This algorithm succeeds in finding a good coloring with high probability, and the running time is $n^{O(1/\epsilon)}$ – polynomial time for fixed $\epsilon$. This cannot be amplified to succeed with probability 1, as it is not clear how to test whether the output of the algorithm is valid. Thus, it is a Monte Carlo, but not a Las Vegas, algorithm.

**9.4.1. New results.** We present the first polynomial-time coloring that shows $\pi(G) \leq (1 + o(1))\Delta^2$; furthermore, our algorithm is Las Vegas. Until this work, no Las Vegas algorithms were known for this problem where the number of colors $C$ is *any* function of $\Delta$, and no Monte Carlo algorithms were known where $C = \phi\Delta^2$ for $\phi$ any fixed constant. We also develop the first-known $ZNC$ (parallel Las Vegas) versions of such results.

As another application, Section 9.4.4 considers a generalization of non-repetitive colorings, introduced in [**9**], to avoid *k-repetitions*. That is, given an integer parameter $k \geq 2$, we aim to color the vertices to avoid the event that a sequence of colors $xx \ldots x$ appears on a vertex-simple path, with the string $x$ occurring $k$ times. (Standard non-repetitive coloring corresponds to $k = 2$.) The best type of result achievable in polynomial time using [**48**] is a coloring using $O(\Delta^{1+\epsilon})$ colors, for any desired *constant* $\epsilon > 0$. Theorem 9.11 shows that a coloring using $\Delta^{1+\frac{1+\epsilon}{k-1}} + O(\Delta^{2/3+\frac{1+\epsilon}{k-1}})$ colors and which avoids any $k$-repetitions, can be found in $n^{O(1/\epsilon)}$ (i.e., polynomial) time.

A second type of generalization of non-repetive colorings comes from work of [**70**], which considered when it is possible to avoid nearly-repeated color sequences; that is, a sequence of colors $xy$ where the Hamming distance of $x$ and $y$ is small. The work of [**70**] considered the problem for color sequences alone, while we extend this to graph

coloring. This presents new algorithm challenges as well. We develop new bounds and algorithms in Section 9.4.5.

### 9.4.2. Non-repetitive vertex coloring.

THEOREM 9.8. *There is some constant $\phi > 0$, such that any graph $G$ of maximum degree $\Delta$ can be $C$-colored to avoid repetitive vertex-colorings as long as $C \geq \Delta^2 + \phi\Delta^{5/3}$. Furthermore, such a coloring can be found in expected time $O(n^2\Delta^{-2/3})$.*

PROOF. A bad-event in this context is some vertex-simple path with a repeated color sequence, of length $2l$. We define $\mu(B) = \alpha^{2l}$ for all such events, where $\alpha$ is a parameter to be determined. Our convention is that each color sequence gives rise to a distinct bad-event; thus, all bad-events are *atomic* and have probability $C^{-2l}$.

Now consider a fixed vertex $v$, and let us enumerate all bad events affecting it. These have the following form: There is a path of length $2l$, of which $v$ is the $t^{\text{th}}$ vertex for some $t = 0, \ldots, l-1$ (by reversing the path, one can assume without loss of generality $v$ comes in the initial half); the first $l$ vertices have some pattern of colors, and the final $l$ vertices have also this pattern. Summing over all possible values of $t, l$, all $\Delta^{2l-1}$ paths, and all possible $C^l$ color patterns, we have that the total contribution of all bad events involving the vertex $v$ is at most

$$\sum_{l=1}^{\infty}\sum_{r=1}^{l} C^l\Delta^{2l-1}\alpha^{2l} = \frac{\alpha^2 C\Delta}{(1-\alpha^2 C\Delta^2)^2} \qquad \text{for } \alpha^2 C\Delta^2 < 1.$$

To show that the asymmetric LLL criterion holds, consider some bad-event $B$ of length $2l$. Its probability is $C^{-2l}$. Its independent sets of neighbors can be determined by, for each of the $2l$ vertices, selecting zero or one bad-events involving that vertex. Thus, the total weight of all independent sets of neighbors is at most $(1+\frac{\alpha^2 C\Delta}{(1-\alpha^2 C\Delta^2)^2})^{2l}$. Thus, the LLL criterion becomes

$$\alpha^{2l} \geq C^{-2l}(1 + \frac{\alpha^2 C\Delta}{(1-\alpha^2 C\Delta^2)^2})^{2l}$$

which is satisfied for all $l \geq 1$ iff

$$\alpha C \geq 1 + \frac{\alpha^2 C \Delta}{(1 - \alpha^2 C \Delta^2)^2} \tag{45}$$

Set $\alpha = (\sqrt{C}(\Delta + \Delta^{2/3}))^{-1}$; routine algebra shows that (45) holds for $\phi$ sufficiently large.

The key bottleneck is to search for some true bad event. Suppose we are given a configuration and a fixed vertex $v$, and we wish to determine if $v$ participates in any paths with repeated colors. As we have said before, this is difficult because there are exponentially many potential bad events. However, we will take advantage of the random nature of the typical coloring; we will be able to rule out certain bad events prematurely, pruning our search in those cases.

Say that $v$ participates in a repeated path $v_0, \ldots, v_{2l-1}$ of length $2l$, and occurs in position $t < l$. For the moment, let us suppose that $t = 0$ and $l$ is fixed. To emphasize the position of $v$ in the list, we write $v_t = v = v_0$.

We begin by looping over the vertex in position $l$. If this vertex $v_l$ has the same color as $v_0$, we continue the search, otherwise we abort. Next, we loop over all neighbors $v_1, v_{l+1}$ of $v_0, v_l$ respectively. Again, if they have the same color (and also $v_1 \neq v_{l+1}$), then we continue the search otherwise we abort. We continue this process, looping over pairs of vertices $v_2, \ldots, v_{l-1}, v_{l+2}, \ldots, v_{2l-1}$. At each stage of this branching process, we insist that the colors in the path are repeated up to that point, and all vertices are distinct. At the end, we examine if the resulting path corresponds to a bad event. We can do a similar procedure if $t \neq 0$; we begin by guessing vertices $v_{t+1}, \ldots, v_{l-1}, v_{t+l}, \ldots, v_{2l-1}$ and then branch backward on $v_{t-1}, \ldots, v_0, v_{l+t-1}, \ldots, v_l$.

Let us examine how to write the work factor of this type of branching process. Suppose again that $v_t$ is a fixed vertex with a color $c$. For position $v_{t+l}$, we can loop over any vertex $v'$ with the same color as $v_t$. The total work for this looping is the number of vertices with color $c$; i.e. it is $\sum_{v'}[\chi(v') = c]$, where $\chi$ denotes the coloring map. (Our data structure $D$ will maintain the vertices sorted by color; this is easy

to do in linear time.) This term has the form required for Theorem 9.5; i.e., it is the sum of indicator functions of events.

The next stage of the branching process requires extending the stories by enumerating $v_{t+1}, v_{t+l+1}$ with the same color, distinct and neighbors of $v_t, v_{t+l}$ respectively. By sorting the neighborhoods of these vertices, one can enumerate all such pairs in time $\Delta$ plus the number of such pairs. That is, the next stage of the branching process has cost

$$\sum_{v'}[\chi(v') = c]\Delta + \sum_{\substack{v',c \\ w \in N(v), w' \in N'(v)}} [\chi(v') = c \wedge \chi(w) = c' \wedge \chi(w') = c'],$$

where here the terms $w, w'$ indicate potential candidates for $v_1, v_{t+1}$. Again, this has the form required for Theorem 9.5.

When we are applying Theorem 9.5 to compute $T_B$, for any boolean predicate $X(E)$, a term of the form $[X(E)]$ will contribute $P_\Omega(E)\theta(E)$. If the event $E$ is defined by $E = (\chi(u_1) = c_1) \wedge \cdots \wedge (\chi(u_k) = c_k)$ for $u_1, \ldots, u_k$ distinct, then a simple calculation shows that $P_\Omega(E)\theta(E) \leq \alpha^k$. Hence, by Theorem 9.5, the total cost of the branching process to find $v_{t+l}$ is $n\alpha$ and the total cost to enumerate $v_{t+1}, v_{t+l+1}$ is $n\alpha\Delta + n\alpha^3 C\Delta^2$. For the remainder of this analysis, we will not specifically write the running time of our data-structure in term of Theorem 9.5, but we will go immediately to writing down the final contribution in terms of polynomials of $n, \alpha, \Delta$.

Assuming that $t, l$ are fixed and known, the *total* contribution of the branching process, across all $l$ stages, is at most $\sum_{r=0}^{l} nC^r(\alpha\Delta)^{2r+1}$. With a little thought, one can see that it is not necessary to specify a fixed value of $l, t$ for this branching. Once one specifies the initial vertex $v_t$ (without necessarily knowing $t$) and the corresponding vertex $v_{t+l}$ (again, without necessarily knowing $l$), one merely has to decide how many steps to branch forward/backward from these two vertices. If at some point during this branching process one detects a repeated color sequence, one can then infer the corresponding $t, l$.

If one branches $r_1$ forward steps and $r_2$ backward steps, then again the work expended is $n(\alpha\Delta)^{2(r_1+r_2)+1}$. Summing over $r_1, r_2$, one has the total work of branching at most

$$n\sum_{r_1=0}^{\infty}\sum_{r_2=0}^{\infty} C^r(\alpha\Delta)^{2(r_1+r_2)+1} \leq n\sum_{r=0}^{\infty}(r+1)C^r(\alpha\Delta)^{2r+1} \leq O(n\Delta^{-1/3}).$$

So far, we have computed the total work for finding bad events $B'$ which contain a given vertex. Now, suppose we have a bad event $B$, we need to find bad events $B' \sim B$ which are created after resampling $B$. If the bad event $B$ has length $2l$, then by the same token, the work expended will be $T_B \leq 2l \times O(n\Delta^{-3})$. Summing over all such bad events, we have

$$\sum_{B}\mu(B)T_B \leq \sum_{l=1}^{\infty} n\Delta^{2l-1}c^l\alpha^{2l} \times 2l \times O(n\Delta^{-1/3}) \leq O(n^2\Delta^{-2/3})$$

We want to emphasize the intuition here, which is that searching for a repetitive coloring in the intermediate configurations of the MT algorithm is very similar for searching for a repetitive coloring in a completely random configuration. For, suppose we were simply choosing a random vertex coloring and checking if there was a repetitive coloring. In this case, one could similarly compute the expected running time of the branching algorithm. One would obtain identical formulas, with the only difference being that all instances of $\alpha$ in the above proof would be replaced by the slightly smaller value $C^{-1}$, the probability that a given vertex has a given color. $\square$

**9.4.3. Parallel algorithm for the Thue number.** When we seek to turn the above into a parallel (say, $ZNC$) algorithm, there are three aspects of the MT algorithm which can give us trouble. The first two are standard issues with this algorithm: First, we need the MT algorithm to terminate after a polylogarithmic number of iterations. Second, we need to find a maximal independent set of bad events in

polylogarithmic time. The third roadblock is the most subtle — it can be very difficult to deal with "large" bad events — paths consisting of more than polylogarithmic vertices.

PROPOSITION 9.9. *There is a constant $\phi > 0$ such that any graph $G$ of maximum degree $\Delta$ can be $C$-colored to avoid repetitive vertex-colorings as long as $C \geq \Delta^2 + \phi\Delta^2/\log\Delta$. Furthermore, such a coloring can be found in ZNC (Las Vegas NC): the algorithm terminates successfully with probability 1 after expected time $O(\log^6 n)$ using $n^{O(1)}$ processors.*

PROOF. In order to get a Las Vegas NC algorithm, we will only need to bound the *expected* size of the branching process involved. We will have a polynomial number of processors, sufficiently large to handle (say) $n$ times the expected size of the total branching work. If the branching work ever exceeds this bound, we abort the entire algorithm and start from scratch. This will only multiply the total expected time by a constant factor.

The main difficulty in turning our sequential algorithm into a parallel one is that our branching process seems to require $l$ steps to check a path of length $l$ — extending partial paths vertex by vertex. This means that we will not be able to check long paths in polylogarithmic time.

To handle this, we require a stronger condition on the coloring. Let $T = x\log^3 n$, for $x$ a sufficiently large constant (to be specified later). We disallow any repeated path of length $2l$, for $l \leq T$; this is the usual condition for non-repetitive coloring. In addition, we require that there are no vertex-simple paths $L_1, L_2$ of length $\geq T$, which share no vertices, which have the same color sequence. Although this condition only involves $O(\log^3 n)$ vertices (and hence gives rise to "small" bad events), it is not hard to see that if this condition if satisfied there can be no repeated paths of length $\geq T$. Thus, these two conditions jointly guarantee that we have a non-repetitive coloring.

In addition to the repeated color paths, each vertex now participates in $nT\Delta^{2T}$ of this second type of bad event. Along the same lines as Theorem 9.8, a sufficient

condition for the parallel MT algorithm with $\epsilon$ slack is

$$C\alpha - \frac{\alpha^2 C\Delta}{(1-\alpha^2 C\Delta^2)^2} - nTC^T\Delta^{2T}\alpha^{2T} \geq 1+\epsilon \tag{46}$$

and this is satisfied for $\alpha = (\Delta^2 + \frac{\phi\Delta^2}{2\log\Delta})^{-1}$.

For $\phi, x$ sufficiently large, the LHS of (46) is a decreasing function of $\Delta$, hence reaches its minimum value at $\Delta = n$. At this point, one can observe that (46) is satisfied for $\epsilon = \Omega(1/\log n)$. This implies that MT terminates after $O(\log^2 n)$ iterations, with high probability.

We next claim that the bad events can be enumerated in polylogarithmic time in an expected polynomial number of processors. This means we can use Luby's algorithm to choose an MIS in time $O(\log^2 n)$. To prove this claim, it suffices to employ a branching process, which proceeds through $O(\log n)$ stages, and show that the expected stack size remains polynomial. This is very similar to Theorem 9.8, and is omitted. Note that to find bad events of the second type, we will employ the exact same kind of branching we use to find the short repeated color sequences.

So the MT algorithm takes $O(\log^2 n)$ iterations to converge. Each iteration can be executed in parallel time $\log^4 n$ (with a polynomial number of processors), giving a total time of $O(\log^6 n)$. □

**9.4.4. Higher-order Thue numbers.** Recall the notion of *k-repetitions* introduced in [**9**]. That is, given a parameter $k$, we want to avoid the event that a sequence of colors $xx\ldots x$ appears on a vertex-simple path, with the string $x$ occurring $k$ times.

It is not hard to extend the analysis of Theorem 9.8 to obtain an algorithm for $k$-Thue number as follows:

THEOREM 9.10. *For some constant $\phi > 0$, there is an algorithm which takes as input a graph $G$ and parameter $k$, and produces a vertex coloring with $C = \Delta^{1+\frac{1}{k-1}} + \phi\Delta^{2/3+\frac{1}{k-1}}$ colors which avoids $k$-repetitions. This algorithm runs in time $n^{k+O(1)}$.*

For any fixed value of $k$, this is a polynomial-time algorithm. But developing an algorithm whose running time scales with $k$, presents new algorithmic challenges. Note that the approach of [**48**], which is based on finding a "core" set of bad events which can be checked quickly, will not work here — for, the work required to check even the color sequences of length 1 (the simplest class of bad event), is already $n\Delta^k$, which can be super-polynomial time.

Our main result here is:

THEOREM 9.11. *For some constant $\phi > 0$, there is an algorithm with the following properties. It takes as input a graph $G$, a parameter $k$, and a parameter $\epsilon$. It runs in expected time $n^{O(1/\epsilon)}$, and whp produces a vertex coloring with $C = \Delta^{1+\frac{1+\epsilon}{k-1}} + \phi\Delta^{2/3+\frac{1+\epsilon}{k-1}}$ colors, which avoids any $k$-repetitions. That is, there is no vertex-simple path in which a color sequence is repeated $k$ times. Note that this is* not *a Las-Vegas algorithm.*

PROOF. Suppose we are given a fixed $\epsilon > 0$. As in Theorem 9.8, for any bad-event $B$ of length $kl$, we set $\mu(B) = \alpha^{kl}$, where

$$\alpha = \left(\Delta^{1+\frac{1+\epsilon}{k-1}} + \frac{\phi}{2}\Delta^{2/3+\frac{1+\epsilon}{k-1}}\right)^{-1}$$

Now observe that for $\phi > 0$, we have $\alpha^k C\Delta^k < 1$, so the LLL criterion reduces to

$$C\alpha \geq 1 + \frac{k\alpha^k C\Delta^{k-1}}{(1-\alpha^k C\Delta^k)^2} \tag{47}$$

The LHS of (47) can be written as a function of $\Delta, k, \phi$, and a parameter $v = \Delta^{\epsilon/(k-1)}$. By routine calculus, we see that this is indeed satisfied, for all $k, \Delta$, for $\phi$ sufficiently large. (The worse case comes when $k$ is small, $v = 1$, and $\Delta \to \infty$.)

The remaining task is to find any bad events which are true in a current configuration. To begin, we will simply ignore any color-sequences whose length $l$ is greater than $\Omega(\frac{\log n}{\epsilon \log \Delta})$. We claim that, even though we do not check these events explicitly, the probability that any such bad event ever becomes true, is negligible. For, by the

union-bound, the probability of such a long path is

$$P[\text{some bad event occurs with } l \geq x\frac{\log n}{\epsilon \log \Delta}] \leq n \sum_{l=x\frac{\log n}{\log \Delta}}^{\infty} C^l \Delta^{kl} \alpha^{kl}$$

To show that this is negligible, for $x$ a sufficiently large constant, it suffices to show that

$$(C\Delta^k \alpha^k)^{\epsilon^{-1} \log^{-1} \Delta}$$

is bounded below 1. Again, using routine analysis, we have that $(C\Delta^k \alpha^k)^{\epsilon^{-1} \log^{-1} \Delta} \leq 1/e$.

Hence, by the union bound, with high probability no bad events with color sequences longer than $\Omega(\frac{\log n}{\epsilon \log \Delta})$ will ever become true. So we only need to check the shorter sequences.

Now, suppose we wish to check for a $k$-repetition involving a color sequence of length $l$. As we are not attempting to determine exactly the exponent of $n$, we will simplify our task by simply searching for any bad event at all. We will also simply enumerate over the exact value of the length $l$ of the path, rather than attempting to handle all values of $l$ simultaneously. This is wasting work but only by a factor of $n^{O(1)}$.

We first guess the initial vertex $v$ and the color sequence. We next start exploring vertices, starting from $v$, which match the given color sequence. We have a stack of partial paths, beginning at the vertex $v$, which match the color sequence for $t \leq l$ edges. This is a similar type of branching process to the one we considered in Theorem 9.8.

The stack of the branching process begins with size $nC^l$. At each stage, the expected size of the stack is multiplied by a factor of $\Delta\alpha$; the term $\Delta$ represents that we can extend each partial path in $\Delta$ ways; the term $\alpha$ represents that any given vertex matches the color sequence with probability at most $\alpha$. So the size of the stack

after $j$ steps is $nC^L(\Delta\alpha)^j$. Some simple analysis shows that $\Delta\alpha \leq 1 - \Omega(\Delta^{-1/3})$, so the expected work for this branching process is $n^{O(1)}C^l$.

As we are only examining color sequences of length $l \leq O(\frac{\log n}{\epsilon \log \Delta})$, the expected work is at most $n^{O(1/\epsilon)}$.

It is notable in this proof that we need to combine the method of [**48**], which is based on identifying a core subset of bad events, with the fast-search method of Theorem 9.1. In this application, the large bad events cannot be searched efficiently; searching the small "easy" bad events efficiently takes exponential time in general but is polynomial time on the random configurations presented during the MT algorithm.

□

**9.4.5. Approximately-repeated color sequences.** In [**70**], the idea of non-repeated color sequences was generalized to avoiding $\rho$-similar color sequences, for some parameter $0 < \rho \leq 1$. If $x, y$ are two color-sequences of length $l$, we say that $x, y$ are $\rho$-similar if $x, y$ agree in at least $\lceil \rho l \rceil$ positions. When $\rho = 1$, of course, this simply means that $x = y$. Hence the problem of coloring the graph to avoid $\rho$-similar color sequences generalizes the problem of non-repetitive coloring. Although the work of [**70**] considered the problem for color sequences alone, this generalization has not been studied in the context of graph coloring. It presents new algorithmic challenges as well. We present the following result:

THEOREM 9.12. *There is some constant $\phi > 0$ with the following property. For all $\rho \in (0, 1]$ and any graph $G$ with maximum degree $\Delta$, there is a coloring that avoids $\rho$-similar sequences, with*

$$C = \rho^{-1}(1-\rho)^{1-1/\rho}(\Delta^2 + \phi\Delta^{11/6})^{1/\rho}$$

*colors. Furthermore, such a coloring can be found in expected time $n^{O(1)}$.*

PROOF. Define the usual entropy function $h = -(1-\rho)\ln(1-\rho) - \rho \ln \rho$.

We can enumerate the bad events as follows. If we have an sequence $s$ of $2l$ vertices, and a $l$-dimensional binary vector $w$ which has Hamming weight $H(w) = \lceil \rho l \rceil$, we define the bad event $B_{w,s}$ which is that vertices $s_i, s_{i+l}$ have the same color for all indices $i$ which $w_i = 1$. It is not hard to see that there is an $\rho$-similar vertex sequence iff there is some $w, s$ where the bad event $B_{w,s}$ occurs. (We can further insist that the vector $w$ has $w_1 = 1$; this gives slightly better bounds but does not change the asymptotics).

Set $\mu(B) = \alpha^{2l}$ for a bad-event of length $2l$, where $\alpha = e^{-h/\rho}(\Delta^2 + \phi/2\Delta^{11/6})^{-1/\rho}$

Let us count the bad events involving an vertex $v$. We enumerate this as follows. There are $(2l)\Delta^{2l-1}$ paths involving vertex $v$. We must check a vector $w \in \{0,1\}^l$ which has a 1 in the position corresponding to vertex $f$; this gives us $\binom{l-1}{\lceil \rho l \rceil - 1}$ further choices. Then there are $C^{\lceil \rho l \rceil}$ choices for the color sequence shared by $x, y$. Any such event has probability $\alpha^{2\lceil \rho l \rceil}$. Summing over all $l$ gives us a total contribution of

$$\begin{aligned}
\sum_{B \text{ involving } v} \mu(B) &\leq \sum_{l=1}^{\infty} (2l)\Delta^{2l-1} \binom{l-1}{\lceil \rho l \rceil - 1} \alpha^{2\lceil \rho l \rceil} C^{\lceil \rho l \rceil} \\
&= \sum_{k=1}^{\infty} (\alpha^2 C)^k \sum_{l=\lceil k/\rho \rceil}^{\lceil (k+1)/\rho \rceil - 1} (2l)\Delta^{2l-1} \binom{l-1}{k-1} \\
&\leq \sum_{k=1}^{\infty} (\alpha^2 C)^k \sum_{q=0}^{\infty} (2(k+1)/\rho)\Delta^{2l-q-1} \binom{(k+1)/\rho - 1}{k-1} \\
&\leq \sum_{l=1}^{\infty} (2l)\Delta^{2l-1} e^{hl} C^{\rho l} \alpha^{2\rho l} \\
&\leq \frac{2\alpha^{2\rho}\Delta e^h}{(1 - \alpha^{2\rho} C^\rho \Delta^2 e^h)^2}
\end{aligned}$$

Hence the asymmetric LLL criterion for avoiding such $\rho$-similar edge colors reduces to

$$C\alpha \geq 1 + \frac{2\alpha^{2\rho} C^\rho \Delta e^h}{(1 - \alpha^{2\rho} C^\rho \Delta^2 e^h)^2}$$

Routine calculus shows that the LHS is decreasing in $\rho$. So the worst case is when $\rho = 1$; then simple calculus shows that this is satisfied for $\phi$ sufficiently large.

We now come to the main algorithmic challenge: finding a bad event (if any are currently true). One might naively expect to apply the branching process of Theorem 9.8: first choose the first and middle vertex in the path. Then branch on the vertices, aborting the search early if the color sequence so far has too many disagreements. To see why this naive branching process does not give a polynomial-time algorithm, observe that we will not be able to remove *any* stories in the early stages of the branching, because we might have a color sequence $xy$ in which the agreeing positions all come at the *end*. Thus, the stack will increase exponentially before collapsing exponentially. Although the final stack size is relatively small, the maximum stack size can become large. We want the agreeing positions to come fast enough so that the stack size is controlled.

We will branch on the color sequence starting not from the vertices at positions $0, l$ (the first and middle vertex in the path), but rather starting at positions $i, l+i$ for some well-chosen $i = 0, \ldots, l-1$. At the $t^{\text{th}}$ stage of the branching process, we will branch on the vertices at positions $i+t, l+i+t$ modulo $2l$. Here, $t = 0$ corresponds to the initial choice of vertices, and $t = 1$ corresponds to choosing the first edge emanating from them. At stage $t$ of the branching, we insist that the number of agreeing positions seen so far, is at least $\lceil t\rho \rceil$; otherwise we remove that possibility from the branching process. (We refer to an element of this branching process as a *story*; it records information about a subset of the vertices)

To summarize, we use the following algorithm to find bad color sequenes of length $2l$:

1. For $a = 0, \ldots, l-1$ repeat the following:
    2. Initialize the stack with a single, null story.
    3. For $t = 0, \ldots, l-1$ do the following:
        4. For each story in the stack, count the number of positions at which the color sequences agree so far. If this number is smaller than $\lceil \rho t \rceil$, remove the story from the stack.

5. For each story remaining in the stack, choose the vertex at positions $(a+t)$ modulo $2l$ and $(l+a+t)$ modulo $2l$. Typically there are $\Delta^2$ possibilities for these two vertices; except when $t=0$ there are $n^2$ possibilities and at $t = l - a$ there are $n\Delta$ possibilities. Extend each story in the stack in all valid ways.

We will first show that the running time for this algorithm is polynomially bounded. Let us fix some value of $a, t$, and consider the expected stack size. This must correspond to a color sequences $x, y$ of length $t$ which agree on at least $\lceil \rho t \rceil$ positions, and there are $\Delta^{2t} n^{O(1)}$ choices for the vertices. For a fixed sequence of vertices, we can bound the probability that they agree on $\lceil \rho t \rceil$ positions as:

$$\begin{aligned} P(\text{vertex sequence agrees on} \geq \lceil \rho t \rceil \text{ positions}) &\leq \binom{t}{\lceil \rho t \rceil} c^{\lceil \rho t \rceil} \alpha^{2\lceil \rho t \rceil} \\ &\leq n^{O(1)} e^{ht} c^{\rho t} \alpha^{2\rho t} \\ &\leq n^{O(1)} \Big( \frac{\Delta^2 + \phi \Delta^{11/6}}{(\Delta^2 + \phi/2\Delta^{11/6})^2} \Big)^t \end{aligned}$$

Hence, the total expected stack size is at most

$$\mathbf{E}[\text{Stack size at } a, t] \leq n^{O(1)} \Delta^{2t-1} \Big( \frac{\Delta^2 + \phi \Delta^{11/6}}{(\Delta^2 + \phi/2\Delta^{11/6})^2} \Big)^t \leq n^{O(1)}$$

Next, we must show that any bad event will indeed be discovered by this branching process. For, suppose $x, y$ are color sequence of length $l$ which agree on $\rho' l \geq \lceil \rho l \rceil$ positions. For $i = 1, \ldots, l$ define $s_i$ to be the total number of agreements in positions $1, .., i$; for $i$ outside this range, define $s_i := s_{i \bmod l}$. We also define the parameter $r_i = s_i - \rho' i$. Because $x, y$ agree on exactly $\rho' l$ positions, the sequence $r$ is periodic with period $l$.

We claim that for the value of $a$ in the range $1, \ldots, l$ which minimizes $r_a$, then the color sequence $xy$ will survive the corresponding branching process. For, suppose at stage $t$, we lose $xy$. This implies that the total number of agreements between stages

$a, a+t$ is strictly less than $\lceil \rho t \rceil \leq \rho' t$. This implies that $s_{t+a} < s_a + \rho' t$ and hence $r_{t+a} < r_t$, contradicting minimality of $a$. $\square$

# Bibliography


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

[2] Aharoni, R., Berger, E., Ziv, R.: Independent systems of representatives in weighted graphs. Combinatorica 27, pp. 253-267 (2007)

[3] Albert, M., Frieze, A., Reed, B.: Multicoloured Hamilton Cycles. The Electronic Journal of Combinatorics 2-1, R10. (1995)

[4] Alon, N., Kriz, I., Nešetřil, J.: How to color shift hypergraphs. Studia Scientiarum Mathematicarum Hungarica 30, pp. 1-12 (1995)

[5] Alon, N. The linear arboricity of graphs. Israel Journal of Mathematics 62, pp. 311-325 (1988)

[6] Alon, N.: Probabilistic proofs of existence of rare events. Springer Lecture Notes in Mathematics No. 1376 (J. Lindenstrauss and V. D. Milman, Eds.), Springer-Verlag, pp. 186-201 (1988)

[7] Alon, N. The strong chromatic number of a graph. Random Structures and Algorithms 3, pp. 1-7 (1992)

[8] Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms 7(4): 567-583 (1986)

[9] Alon, N., Grytczuk, J. "Breaking the rhythm on graphs." Discrete Math. 308, pp. 1375-1380 (2008)

[10] Alon, N., Grytczuk, J., Haluszczak, M., Riordan, O.: Nonrepetitive colorings of graphs. Random Structures and Algorithms, 21(3-4), pp. 336346 (2002)

[11] Alon, N., Spencer, J. H.: *The Probabilistic Method, Third Edition*. John Wiley & Sons, Inc. (2008)

[12] Alon, N., Spencer, J., Tetali, P.: Covering with Latin transversals. Discrete Applied Math. 57, pp. 1-10 (1995)

[13] Axenovich, M., Martin, R.: On the strong chromatic number of graphs. SIAM Journal of Discrete Math 20-3, pp. 741-747 (2006)

[14] Azar, Y., Epstein, A.: Convex programming for scheduling unrelated parallel machines. Symposium on Theory of Computing (2005)

[15] Bansal, N., Korula, N., Nagarajan, V., Srinivasan, A.: Solving packing integer programs via randomized rounding with alterations. Theory of Computing 8-1, pp. 533-565 (2012)

[16] Beck, J., Tibor, F.: "Integer-making" theorems. Discrete Applied Mathematics 3-1, pp. 1-8 (1981)

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

[18] Bollobás, B., Erdős, P, Szemerédi, E.: On complete subgraphs of $r$-chromatic graphs. Discrete Math 1, pp. 97-107 (1975)

[19] Boppana, R. B., Spencer, J.H.: A useful elementary correlation inequality. Journal of Combinatorial Theory Series A 50, pp. 305-307 (1989)

[20] Bottcher, J., Kohayakawa, Y., Procacci, A.: Properly coloured copies and rainbow copies of large graphs with small maximum degree. Random Structures and Algorithms 40-4, pp. 425-436 (2012)

[21] Brualdi, R. A., Ryser, H. J.: Combinatorial Matrix Theory (Encyclopedia of Mathematics and its Applications). Cambridge University Press (1991)

[22] Chattopadhyay, E., Zuckerman, D.: Explicit two-source extractors and resilient functions. Electronic Colloquium on Computational Complexity (ECCC) 22, p. 19. (2015)

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

[24] Chattopadhyay, E., Zuckerman, D. "Explicit two-source extractors and resilient functions." Electronic Colloquium on Computational Complexity (ECCC) 22, p. 119. (2015)

[25] Chor, B., Goldreich, O.: Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. SIAM Journal on Computing 17, pp. 230-261 (1988)

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

[27] Chvátal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4-3, pp. 233-235 (1979)

[28] Cohen, G.: Two-Source Dispersers for Polylogarithmic Entropy and Improved Ramsey Graphs. arXiv, http://arxiv.org/abs/1506.04428 (2015)

[29] Cook, S.: A Taxonomy of problems with fast parallel algorithms. Information and Control 64, pp. 2-22 (1985)

[30] Dénes, J., Keedwell, A. D.: Latin squares and their applications. Akadémiai Kiadó, Budapest & English Universities Press (1974)

[31] Dobson, G.: Worst-case analysis of greedy heuristics for integer programming with nonnegative data. Mathematics of Operations Research 7-4, pp. 515-531 (1982)

[32] Dujmovic, V., Joret, G., Kozik, J., Wood, D.R. Nonrepetitive colouring via entropy compression. Combinatorica (2013)

[33] Erdős, P., Hickerson, D. R., Norton, D. A., Stein, S. K.: Has every Latin square of order n a partial Latin transversal of size n-1? Amer. Math. Monthly 95, pp. 428430 (1988)

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

[35] Erdős, P., Spencer, J.: Lopsided Lovász Local Lemma and Latin transversals. Discrete Applied Math 30, pp. 151-154 (1990)

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

[37] Feige, U.: A threshold of $\ln n$ for approximating set cover. Journal of the ACM 45-4, pp. 634-652 (1998)

[38] Feige, U., Halldorsson, M., Kortsarz, G., Srinivasan, A.: Approximating the domatic number. SIAM Journal of Computing 32, pp. 172-195 (2002)

[39] Fellows, M.: Transversals of vertex partitions in graphs. SIAM Journal on Discrete Math 3-2, pp. 206-215 (1990)

[40] Fisher, M. L., Wolsey, L. A.: On the greedy heuristic for continuous covering and packing problems. SIAM Journal on Algebraic Discrete Methods 3-4, pp. 584-591 (1982)

[41] Freize, A., Krivelevich, M.: On rainbow trees and cycles. The Electronic Journal of Combinatorics 15-1, R. 59 (2008)

[42] Fortuin, C. M., Ginibre, J., Kasteleyn, P. N.: Correlational inequalities for partiall ordered sets. Communications of Mathematical Physics 22, pp. 89-103 (1971)

[43] Gebauer, H., Szabó, T., Tardos, G.: The local lemma is tight for SAT. Symposium on Discrete Algorithms (SODA) (2011)

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

[45] Ghandehari, M., Hatami, H.: A note on independent dominating sets and second Hamiltonian cycles. Submitted for publication.

[46] Grytczuk, J. Nonrepetitive graph coloring. Graph Theory in Paris, Trends in Mathematics, pp. 209218 (2007)

[47] Grytczuk, J. Nonrepetitive colorings of graphs – a survey. International Journal of Mathematics and Mathematical Sciences 74639 (2007)

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

[49] Hahn, G., Thomassen, C.: Path and cycle sub-Ramsey numbers and an edge-colouring conjecture. Discrete Mathematics 62-1, pp. 29-33 (1986)

[50] Haranta, J., Jendrol, S.: Nonrepetitive vertex colorings of graphs. Discrete Mathematics 312-2, doi: 10.1016/j.disc.2011.09.027, pp. 374380 (2012)

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

[52] Harvey, N.: A note of the discrepancy of matrices with bounded row and column sums. Arxiv 1307.2159 (2013)

[53] Hatami, P., Shor, P. W.: A lower bound for the length of a partial transversal in a Latin square. Journal of Combinatorial Theory Series A 115, pp. 1103-1113 (2008)

[54] Haxell, P. E.: A note on vertex list colouring. Combinatorics, Probability, and Computing 10, pp. 345-348 (2001)

[55] Haxell, P.: On the strong chromatic number. Combinatorics, Probability, and Computing 13-6, pp. 857-865 (2004)

[56] Haxell, P.: An improved bound for the strong chromatic number. Journal of Graph Theory 58-2, pp. 148-158 (2008)

[57] Haxell, P., Szabó, T.: Odd Independent Transversals are Odd. Combinatorics, Probability, and Computing 15, pp. 193-211 (2006)

[58] Haxell, P., Seamonez, B., Verstraete, J.: Independent dominating sets and hamiltonian cycles. Journal of Graph Theory 54-3, pp. 233-244 (2007)

[59] Haxell, P. E., Szabó T, Tardos, G.: Bounded size components – Partitions and transversals. Journal of Combinatorial Theory Series B 88, pp. 281-297 (2003)

[60] Jin, G.: Complete subgraphs of $r$-partite graphs. Combinatorics, Probability, and Computing 1, pp. 241-250 (1992)

[61] , Karp, R.: Reducibility among combinatorial problems. Springer (1972).

[62] Karp, R., Leighton, F. T, Rivest, R. L., Thompson, C. D, Vazirani, U. V, Vazirani, V. V.: Global wire routing in two-dimensional arrays. Algorithmica 2 1-4, pp. 113-129 (1987)

[63] Karp, R. M., Leighton, F. T., Rivest, R. L., Thompson, C. D., Vazirani, U. V., Vazirani, V. V.: Global wire routing in two-dimensional arrays. Algorithmica 2, pp. 113-129 (1987)

[64] Keevash, P., Ku, C.: A random construction for permutation codes and the covering radius. Designs, Codes and Cryptography 41-1, pp. 79-86 (2006)

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

[66] Kolipaka, K., Szegedy, M., Xu, Y.: A sharper local lemma with improved applications. Approximation, Randomization, and Combinatorial Optimizationn. Algorithms and Techniques LNCS 7408, pp. 603-614 (2012)

[67] Kolliopoulos, S., Young, N.: Approximation algorithms for covering/packing integer programs. Journal of Computer and System Sciences 71, pp. 495-505 (2005)

[68] Kolmogorov, V.: Commutativity in the random walk formulation of the Lovász Local Lemma. Arxiv 1506.08547 (2015)

[69] Kratochvíl, J., Savický, P., Tuza, Z.: One more occurrence of variables makes satisfiability jump from trivial to NP-complete. SIAM Journal of computing 22-1, pp. 203-210 (1993)

[70] Krieger, D., Ochem, P., Rampersad, N., Shallit, J.: Avoiding approximate squares. Developments in Language Theory. LNCS 4588, pp. 278-289 (2007)

[71] Krivelevich, M.: Deciding $k$-colorability in expected polynomial time. Information Processing Letters 81-1, pp. 1-6 (2002)

[72] Leighton, F. T., Lu, C.-J., Rao, S. B., Srinivasan, A.: New algorithmic aspects of the Local Lemma with applications to routing and partitioning. SIAM Journal on Computing 31, pp. 626-641 (2001)

[73] Leigton, F. T., Maggs, B., Rao, S.: Packet routing and jobshop scheduling in $O$(congestion + dilation) steps. Combinatorica 14, pp. 167-186 (1994)

[74] Lenstra, J. K., Shmoys, D. B., Tardos, É: Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming 46, pp. 259-271 (1990)

[75] Loh, P. S., Sudakov, B. Independent transversals in locally sparse graphs. Journal of Combinatorial Theory Series B 97, pp. 904-918 (2007)

[76] Lu, L., Mohr, A., Székely, L.: Quest for negative dependency graphs. Recent Advances in Harmonic Analysis and Applications. Springer New York, 2013. 243-258.

[77] Lu, L., Székély, L.: Using Lovász Local Lemma in the space of random injections. The Electronic Journal of Combinatorics 13-R63 (2007)

[78] Lu, L., Székély, L.: A new asymptotic enumeration technique: the Lovász local lemma. arXiv:0905.3983v3 (2011)

[79] Luby, M., Nisan, N.: A parallel approximation algorithm for positive linear programming. Symposium on Theory of Computing, pp. 448-457 (1993)

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

[81] Mahler, K.: An inequality for the discriminant of a polynomial. The Michigan Mathematical Journal 11.3, pp. 257-262 (1964)

[82] Marx, D., Schaefer, M.: The complexity of nonrepetitive coloring. Discrete Applied Mathematics 157.1, pp. 13-18 (2009)

[83] McDiarmid, C.: Hypergraph coloring and the Lovász Local Lemma. Journal of Discrete Mathematics 167/168, pp. 481-486 (1995)

[84] Mohr, A.: Applications of the Lopsided Lovász Local Lemma regarding hypergraphs. PhD Thesis, University of South Carolina (2013)

[85] Molloy, M., Reed, B.: Graph Colouring and the Probabilistic Method. Springer-Verlag (2001)

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

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

[88] Moshkovitz, D.: The projection games conjecture and the NP-hardness of $\ln n$-approximating set-cover. Approximation, Randomization, and Combinatorial Optimization Algorithms and Techniques, pp. 276-287 (2012)

[89] Motwani, R., Raghavan, P.: Randomized algorithms. Chapman & Hall/CRC (2010)

[90] Raghavan, P., Thompson, C. D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7-4, pp. 365-374 (1987)

[91] Nisan, N., Zuckerman, D.: Randomness is linear in space. Journal of Computer and System Sciences 52, pp. 43-52 (1996)

[92] Panconesi, A., Srinivasan, A.: Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. SIAM Journal of Computing 26-2, pp. 350-368 (1997)

[93] Peis, B., Wiese, A.: Universal packet routing with arbitrary bandwidth and transit times. IPCO 2011, pp. 362-375 (2011)

[94] Pegden, W.: An extension of the Moser-Tardos algorithmic local lemma. SIAM Journal of Discrete Math 28, pp. 911-917 (2014)

[95] Radhakrishnan, J., Srinivasan, A., "Improved bounds and algorithms for hypergraph two-coloring." Random Structures and Algorithms 16, pp. 4-32 (2000)

[96] Raghavan, P., Thompson, C. D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7, pp. 365-374 (1987)

[97] Rothvoss, T.: A simpler proof for $O$(congestion+dilation) packet routing. Arxv 1206.3718 (2012)

[98] Ryser, H. J.: Neuere Probleme der Kombinatorik. In Vortrage uber Kombinatorik, pp. 69-91, Oberwolfach, Matematisches Forschungsinstitute Oberwolfach (1967)

[99] Scheideler, C.: Universal routing strategies for interconnection networks. Lectures Notes in Computes Science 1390. (1998)

[100] Schmidt, J. P, Siegel, A., Srinivasan, A.: Chernoff-Hoeffding bounds for applications with limited independence. SIAM Journal of Discrete Mathematics 8, pp. 223-250 (1995)

[101] Scott, A., Sokal, A.: The repulsive lattice gas, the independent-set polynomial, and the Lovász Local Lemma. Journal of Statistical Physics 118, No. 5-6, pp. 11511261 (2005)

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

[103] Shor, P. W.: A lower bound for the length of a partial transversal in a Latin square. Journal of Combinatorial Theory Series A 33, pp. 1-8 (1982)

[104] Spencer, J.: Asymptotic lower bounds for Ramsey functions. Discrete Mathematics 20, pp. 69-76 (1977)

[105] Spencer, J. H. Lecture notes (2013)

[106] Singh, M.: Iterative methods in combinatorial optimization. PhD thesis from Tepper School of Business, Carnegie-Mellon University (2008)

[107] Slavik, P.: A tight analysis of the greedy algorithm for set cover. Journal of Algorithms 25-2, pp. 237-254 (1997)

[108] Srinivasan, A.: An extension of the Lovász Local Lemma, and its applications to integer programming. SIAM Journal on Computing 36-3, pp. 609-634 (2006)

[109] Stein, S. K.: Transversals of Latin squares and their generalizations. Pacific Journal of Mathematics 59, pp. 567-575 (1975)

[110] Szabó, S.: Transversals of rectangular arrays. Acta Math. Univ. Comenianae, Vol. 37, pp. 279-284 (2008)

[111] Szabó, T., Tardos, G.: Extremal problems for transversals in graphs with bounded degree. Combinatorica 26, pp. 333-351 (2006)

[112] Thomassen, C.: Independent dominating sets and a second Hamiltonian cycle in regular graphs. Journal of Combinatorial Theory Series B 72, pp. 104-109 (1998)

[113] Thue, A.: Uber Unendliche Zeichenreihen. Norske Vid Selsk. Skr. I. Mat. Nat. Kl. Christiana 7, pp. 1-22 (1906)

[114] Vadhan, S.: Pseudorandomness. Foundations and Trends in Theoretical Computer Science 7, pp. 1-336, NOW Publishers (2012)

[115] Vazirani, V. V.: Approximation algorithms. Springer (2001)

[116] Yuster, R.: Independent transversals in $r$-partite graphs. Discrete Math 176, pp. 255-261 (1997)