Parallel algorithms for the Lopsided Lovász Local Lemma

David G. Harris

Abstract

The Lovász Local Lemma (LLL) is a probabilistic tool which shows that, if a collection of “bad” events $B$ in a probability space are not too likely and not too interdependent, then there is a positive probability that no bad-events in $B$ occur. Moser & Tardos (2010) gave sequential and parallel algorithms which transformed most applications of the variable-assignment LLL into efficient algorithms.

There has been limited success in developing parallel algorithms for more generalized forms of the LLL. Harris (2016) developed RNC algorithms for the variable-assignment Lopsided Lovász Local Lemma (LLLL) and Harris & Srinivasan (2014) developed an algorithm for the permutation LLL. These algorithms are cumbersome and limited in the types of bad-events they can handle (limitations not shared for the standard LLL). Kolmogorov (2016) developed a framework which partially parallelizes LLL settings such as random matchings of $K_n$, although key algorithm components remained missing.

We give new parallel algorithms for most forms of LLLL, which are simpler, faster, and more general than the algorithms of Harris and Harris & Srinivasan. This also includes probability spaces for which no previous RNC algorithm was known, including matchings and hamiltonian cycles of $K_n$. We achieve this by providing a unified algebraic framework for applications of the LLLL to permutations.

The parallel LLLL algorithm is based on a new primitive for parallel computing, which we refer to as the lexicographically-first maximal-independent-set of a directed graph. We give an efficient algorithm for constructing this and show that it is precisely what is needed for our LLLL algorithms. This generalizes an algorithm given by Blelloch, Fineman, Shun (2012) for undirected graphs.

*Department of Computer Science, University of Maryland, College Park, MD 20742. Email: davidgharris29@gmail.com
1 The Lovász Local Lemma and its algorithms

The Lovász Local Lemma (LLL) is a probabilistic tool, first introduced in [9], which shows that for a probability space \( \Omega \) with a finite set \( B \) of \( m \) “bad” events, then as long as the bad-events are not too interdependent (in a certain technical sense) and are not too likely, then there is a positive probability no events in \( B \) occur. This principle has become a cornerstone of the probabilistic method of combinatorics, as this establishes that a configuration avoiding \( B \) exists. This does not lead to efficient algorithms, as the probability of avoiding all the bad-events is typically exponentially small.

The simplest formulation of the LLL, known as the symmetric LLL, can be stated as follows: suppose that each bad-event \( B \) has probability \( P_\Omega(B) \leq p \); and suppose that each bad-event \( B \) is dependent with at most \( d \) others; and suppose that \( ep(d + 1) \leq 1 \). Then there is a positive probability that none of the bad-events occur. Other forms of the LLL, such as the asymmetric LLL, or the cluster-expansion criterion [3] allow more complex information about the probabilities and interdependency structure of the bad-events; Shearer’s criterion [32] gives the most powerful criterion of this type, although it is technically difficult to analyze.

Most applications of the LLL in combinatorics make use of a relatively simple probability space and dependency structure, which we refer to as the variable-assignment LLL. In the setting, the probability space \( \Omega \) selects \( n \) independent variables \( X_1, \ldots, X_n \), where each \( X_i \) has from a finite probability distribution \( p_i \). Each bad-event is a boolean function of a subset of these variables, which we denote \( \text{Var}(B) \), and bad-events \( B, B' \) are dependent (written \( B \sim B' \)) iff \( \text{Var}(B) \cap \text{Var}(B') \neq \emptyset \).

In [30], Moser & Tardos introduced a remarkably simple algorithm to find configurations for the variable-assignment LLL setting, which we refer to as the MT algorithm:

1. Draw each variable independently from the distribution \( \Omega \).
2. While there is some true bad-event:
   3. Choose a true bad-event \( B \) arbitrarily.
   4. Resample \( \text{Var}(B) \) according to the distribution \( \Omega \).

Moser & Tardos showed that this algorithm terminates quickly whenever the symmetric LLL criterion (or, more generally, the asymmetric LLL criterion) is satisfied. Later work of [31] and [24] showed that it terminates under more general LLL criteria, including Shearer’s criterion. See Appendix A for background on the LLL and MT algorithm.

In order to implement the MT algorithm, one must find a bad-event \( B \) which is true on the current configuration \( X \) (if any). We refer to this as a Bad-Event Checker. The simplest way to implement this step would be to loop over all bad-events and test them one by one, which would have a running time on the order of \( m \). As noted by [16] [18], the running time of the MT algorithm can often be polynomial in \( n \) and independent of \( m \) if a more-efficient Bad-Event Checker is used.

1.1 The Lopsided Lovász Local Lemma

In [10], Erdős & Spencer noted that positive correlation among bad-events (again, in a certain technical sense) is as good as independence for the LLL. This generalization has been referred to as the Lopsided Lovász Local Lemma (LLLL). We say \( B, B' \) are lopsidependent and write \( B \sim B' \) if \( B, B' \) are not positively correlated in this sense.

Although the variable-assignment LLL covers the vast majority of application to combinatorics, there have been a few applications of various forms of the LLLL. The original paper [10] introducing
the LLLL, for example, used a probability space based on random permutations to construct Latin transversals for certain types of arrays. Other applications include hamiltonian cycles on $K_n$ [27] and matchings of $K_{n \times 2}$. The variable-assignment setting provides one of the simplest forms of the LLLL, and the original algorithm of Moser & Tardos applies to it. In [19], an algorithm similar to the MT algorithm was developed for the probability space of random permutations, which includes the Latin transversal application of [10]. These algorithms were very problem-specific; a more recent line of research has been developing generic LLLL algorithms, which can cover most of the probabilistic forms of the LLLL. Notably, work of [1, 21] has led to “meta-algorithms” which have the same flavor as the MT algorithm: whenever a bad-event is true, we “resample” it (re-randomize its relevant variables). These results have led to constructive counterparts to combinatorial results involving spanning trees, matchings of $K_n$ (both discussed in [21]) and hamiltonian cycles of $K_n$ (subsequently developed in [20]).

1.2 Parallel algorithms for the LLL

This sequential MT algorithm terminates in polynomial time whenever the LLL criterion is satisfied, thus turning nearly all applications of the LLL in combinatorics into efficient algorithms. Moser & Tardos also gave a simple parallel algorithm:

1. Draw each variable independently from the distribution $\Omega$.
2. While there is some true bad-event:
   3. Select a maximal independent set (MIS) $I$ of true bad-events.
   4. Resample, in parallel, $\bigcup_{B \in I} \text{Var}(B)$.

This parallel algorithm requires a slightly stronger criterion, which we refer to as $\epsilon$-slack; for instance, the symmetric LLL requires $\epsilon p(1 + \epsilon)(d + 1) \leq 1$; if this satisfied, then it terminates after $O(\epsilon^{-1} \log m)$ rounds. If the Bad-Event Checker runs in parallel time $T$, then the overall running time is $O(\epsilon^{-1} \log m (\log^2 m + T))$ on an EREW PRAM. (The factor of $\log^2 m$ here is the cost of an MIS computation.)

In [15], Haeupler & Harris showed that the parallel MT algorithm could be implemented in time $O(\epsilon^{-1} \log n (\log^2 n + T))$ (avoiding the dependence on $m$) and gave an alternative parallel algorithm running in time $O(\epsilon^{-1} \log^2 m)$. As shown in [24], this parallel algorithm can usually be implemented even for more general LLL criteria, including the asymmetric LLL and Shearer’s LLL criterion.

Many graph problems, such as various forms of vertex coloring, can be solved via the LLL. The parallel MT algorithm typically leads in a straightforward way into distributed coloring algorithms in $O(\epsilon^{-1} \log^2 m)$ communication rounds. There has been extensive research into obtaining faster distributed and parallel LLL algorithms. In [7], Chung, Pettie & Su gave a distributed algorithm running in $O(\log m)$ rounds if an alternate LLL criterion (much stronger than the LLL criterion with $\epsilon$-slack) is satisfied, and running in $O(\epsilon^{-1}(\log^2 d) \log m)$ rounds for the symmetric LLL with $\epsilon$-slack. The latter result was subsequently improved by Ghaffari [13] to $O(\epsilon^{-1}(\log d)(\log m))$ rounds. Another distributed algorithm (requiring somewhat much more stringent conditions than the LLL) was given by Fischer & Ghaffari [14] running in time $2^{O((\log^2 \log n))}$. On the other side, Brandt et al. [5] showed that generic distributed LLL algorithms must require time $\Omega(\log \log n)$.

---

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

2 On a CRCW PRAM (in which multiple processors can write to a memory cell simultaneously), one can typically save a factor of $\log m$ in runtime. In order to simplify the discussion, we will be conservative and use only the EREW PRAM model throughout this paper.
1.3 The Kolmogorov framework for parallel LLLL algorithms

Much less progress has been made for parallel LLLL algorithms. Frustratingly, although the sequential MT algorithm works for the variable-assignment LLL just as it does for the variable-assignment LLL, this is not true of the parallel MT algorithm. There have been only a handful of parallel algorithms for the LLLL, such as the variable-assignment LLLL algorithm of Harris [17] and the permutation LLL algorithm of Harris & Srinivasan [19].

In [25] Kolmogorov proposed a general framework for constructing parallel LLLL algorithms. Assuming the existence of a “resampling oracle” for the probability space (using the language of [21]), this can be summarized as follows:

Algorithm 1 Kolmogorov’s framework for parallel resampling algorithms

1: Draw the state $X$ from the distribution $\Omega$
2: while there is a true bad-event on $X$ do
3: Set $V$ to be the set of currently-true bad-events
4: while $V \neq \emptyset$ do
5: Select, arbitrarily, some bad-event $B \in V$.
6: Resample $B$ using the resampling oracle.
7: Remove from $V$ all bad-events $A$ such that either (i) $A$ is no longer true; or (ii) $A \sim B$

Each iteration of the loop (steps 3 – 6) is referred to as a round. Kolmogorov shows that if the resampling oracle satisfies an additional property known as “commutativity,” and the LLL criterion is satisfied with $\epsilon$-slack, then this procedure terminates whp after $O(\epsilon^{-1} \log n)$ rounds. Thus, this gives an RNC algorithm if one can implement an individual round in polylogarithmic time. Kolmogorov further showed that many LLL probability spaces, including permutations and perfect matchings of $K_{2n}$, satisfy this commutativity property.

The Kolmogorov framework thus overcomes one of the major hurdles of developing parallel LLLL algorithms; however, there remain two other hurdles which are equally significant.

The first hurdle is that, in the LLLL, it is possible that there are two bad-events $A, B$ which are currently true, and $A \not\sim B$, and resampling $A$ makes $B$ false. In such cases, it is not necessarily valid to simultaneously resample $A$ and $B$; one must select (arbitrarily) one of the two bad-events to resample first, and then only resample the second one if it remains still true. Thus, there is an inherently sequential ordering which must be maintained for the resamplings within each round. This effect does not hold for the variable-assignment LLL: all the true bad-events may be resampled simultaneously without this potential for mutual cancellation. One critical challenge for LLLL algorithms is to simulate (in parallel) the process of selecting an ordering of the bad-events and resampling them in sequence.

The second major hurdle is the need to develop a parallel-compatible resampling oracle. To illustrate the problem, consider the resampling oracle developed by Harvey & Vondrák for perfect matchings on $K_n$. If the current state $M$ contains some edge $e$ that we need to resample, then their resampling rule selects some other edge $f \in M$, and performs a “swapping operation” between $e$ and $f$. This would break down if two separate edges $e, e'$ simultaneously tried to swap with the same edge $f$. So this operation requires each resampling to “lock” the full state. We contrast this with variable-assignment LLL; if we select two bad-events $B, B'$ to resample, then necessarily $\text{var}(B)$ and $\text{var}(B')$ are disjoint and so their resamplings can be executed simultaneously.

The parallel LLLL algorithms of Harris and Harris & Srinivasan overcome these hurdles to a limited extent. However they still suffer from a number of shortcomings. Although they run in polylogarithmic time, the exponent is quite high (and is not computed explicitly). They also require
that the bad-events only involve a polylogarithmic number of variables. Finally, and perhaps most seriously, these algorithms are highly specialized and only apply to a single probability space.

1.4 Our contribution and overview: new parallel algorithms for the LLLL

The main focus of this paper is to develop simpler, more-efficient parallel LLLL algorithms and to develop a more generalized framework for parallel LLLL algorithms. Our algorithms will follow the Kolmogorov framework, but will overcome its two main hurdles. We briefly summarize how we are able to do so.

In order to simulate the sequential ordering of multiple bad-events, we construct a directed graph whose vertex set is labeled by the bad-events $V$ for each round. We construct an edge from $A$ to $B$ if resampling $A$ makes $B$ false or $A \sim B$. A consistent sequential ordering of the bad-events to resample corresponds to a structure which is similar to a lexicographically-first maximal-independent-set (LFMIS) of this graph. We show that such this object can be computed efficiently in the parallel and distributed settings, generalizing an analysis of Blelloch, Fineman, Shun [4] for undirected graphs. We believe that this new graph primitive may be useful for other distributed and parallel algorithms.

To implement parallelizable resampling oracles, we encode a number of LLLL probability spaces as subsets of the symmetric group $S_n$. This includes matchings of $K_{2n}$ and hamiltonian cycles of $K_n$, which are superficially unrelated to permutations. The resampling oracle for a bad-event $B$ can then be interpreted as multiplication of the current state of the system $\pi$ (which is an element of $S_n$) by some random permutations $\sigma_B$. Crucially, the choice of permutation $\sigma_B$ depends only on $B$, not on the state $\pi$ itself. In this way, resampling multiple bad-events $B_1, \ldots, B_t$ can be interpreted as the functional composition $\pi \sigma_{B_1} \cdots \sigma_{B_t}$, which can be computed easily in polylogarithmic time.

This algebraic approach to the permutation LLL does more than provide efficient algorithms. It also allows us unify a number of probability spaces in a single framework. Many of the proofs and constructions for these spaces had previously been presented in ad-hoc fashion; we instead construct a concise list of “axioms” that must be satisfied by a probability space that automatically leads to efficient parallel and sequential randomized algorithms.

We organize our paper as follows. In Section 2, we define the notion of LFMIS for directed graphs and describe an algorithm to compute it in $O(\log^2 n)$ distributed rounds.

In Section 3, we give a new algorithm for the variable-assignment LLLL; we can summarize this as follows:

**Theorem 1.1** (Informal). *Suppose that $ep(1 + \epsilon)(d + 1) \leq 1$ holds for the variable-assignment LLLL. Then there is a parallel algorithm in time $O(\epsilon^{-1} \log^4 n)$ and a distributed algorithm running in time $O(\epsilon^{-1} \log^3 m)$ to find a variable assignment avoiding $B$.***

We give two simple applications of this result:

**Proposition 1.2.** *Suppose we have a k-SAT instance on $n$ variables and $m$ clauses, in which each variable appears in at most $L = 2^{k+1}(1-1/k)^k - 2/k$ clauses for $\epsilon > 0$. Then there is a parallel algorithm to find a satisfying assignment in $O(\epsilon^{-1} \log^4 (mn))$ time and $\text{poly}(m, n, \epsilon^{-1})$ processors whp.*

**Proposition 1.3.** *Suppose $H$ is a hypergraph in which each edge has cardinality $k$ and each vertex appears in at most $L$ edges. Then there is an algorithm in the CONGEST model of distributed computing running in $O(\epsilon^{-1} \log^3 n)$ rounds to construct a non-monochromatic $c$-coloring of the vertices of $H$ when $L = c^k(1-1/k)^k - 1/k(\epsilon^{-1}(1+\epsilon)).$***

---

4In [17], an alternate parallel algorithm was given, which avoids this shortcoming, but requires a different property, namely the probabilities of the variables must be far from one.
In Section 4, we review the permutation LLL, and extend it to a general framework for subsets of permutations. We summarize our algorithm:

**Theorem 1.4 (Informal).** Suppose that $\Omega$ is a probability space corresponding to a subset $W \subseteq S_n$, which has an appropriate resampling oracle. (This includes $W = S_n$ as well as other probability spaces corresponding to matchings of $K_{2n}$ and hamiltonian cycles of $K_n$). Suppose that the Shearer criterion is satisfied with $\epsilon$-slack. Then there is a parallel algorithm for the permutation LLL running in time $O(\epsilon^{-1} \log^4 n)$.

In particular we achieve this running time if the symmetric criterion $ep(1 + \epsilon)(d + 1) \leq 1$ or the asymmetric or cluster-expansion criteria are satisfied with $\epsilon$-slack.

This provides the first RNC algorithm for matchings of $K_{2n}$ and hamiltonian cycles of $K_n$, answering a problem posed by Kolmogorov [25]. As an example, we give an algorithm to find Latin transversals with special cycle structures.

**Proposition 1.5.** Suppose $A$ is an $n \times n$ matrix whose entries are labeled by colors, and each color appears at most $\Delta \leq 0.105$ times in $A$. Then there is a parallel algorithm to find a Latin transversal $\pi$ of $A$ in $O(\log^4 n)$ time whp.

If $n$ is even we can ensure that the cycle structure of $\pi$ consists of exactly $n/2$ transpositions. If $\Delta \leq 0.052n$ we can ensure that the cycle structure of $\pi$ consists of exactly one $n$-cycle.

## 2 LFMIS for directed graphs

Given an undirected graph $G = (V, E)$, an independent set of $G$ is a set $S \subseteq V$ with the property that no two vertices in $S$ are adjacent in $G$. A maximal independent set (MIS) has the additional property that no $T \supseteq S$ is an independent set of $G$. There is a trivial sequential algorithm to find an MIS of $G$: namely, add vertices one-by-one to $S$ as long as they are not adjacent with an already added element of $S$. The MIS produced by this sequential algorithm is referred to as the lexicographically first MIS.

More generally, given any permutation $\pi : V \rightarrow [n]$, we may define the LFMIS of $G$ with respect to $\pi$, as the LFMIS of $G$ when the vertices are re-ordered in increasing of $\pi$. It is P-complete to find the LFMIS of a graph $G$ for any fixed permutation $\pi$ [8]. If $\pi$ is selected uniformly at random from $S_n$, however, then an algorithm of Blelloch, Fineman, Shun [4] can find the LFMIS in $O(\log^2 n)$ rounds. This was later strengthened by Fischer & Noever [11] to $O(\log n)$ rounds, matching the running time for the general MIS.

We will need a a generalization of LMFIS to a directed graph $G$; we refer to this (abusing terminology somewhat) as the directed LFMIS of $G$ with respect to a permutation $\pi$. This is the set $I$ produced by the following sequential process:

**Algorithm 2** The sequential FIND-LFMIS algorithm for an directed graph $G$.

1: Initialize $I \leftarrow \emptyset$.  
2: Mark all the vertices of $G$ as alive  
3: for $i = 1, \ldots, n$ do  
4: if vertex $\pi(i)$ is alive then  
5: Add $\pi(i)$ to $I$  
6: For any directed edge $(\pi(i), \pi(j)) \in E$ mark $\pi(j)$ as dead.
Note that any undirected graph \( G \) can be viewed as a directed graph \( G' \), where every edge \((u, v) \in G\) corresponds to two directed edges \((u, v), (v, u) \in G'\). The LFMIS (in the usual sense) of \( G \) is then identical to the directed LFMIS of \( G' \).

This directed LFMIS will be a basic building block for our parallel LLLL algorithm, playing a similar role to the MIS for the ordinary LLL. The key property that makes the LFMIS so useful is that the decision on whether to place \( v \) into \( I \) depends only on edges of the form \((w, x)\) where \( w \in I \) and \( \pi(w) < \pi(v) \). Thus, if \( \pi \) is a random variable which is chosen in a way independent of \( G \), then the LFMIS of \( G \) with respect to \( \pi \) has strong independence and randomness properties.

Given a directed graph \( G \) and permutation \( \pi \), there is a simple parallel algorithm for finding its LFMIS, which we refer to as the greedy algorithm:

**Algorithm 3** The parallel greedy algorithm to find the LFMIS of a directed graph \( G \)

1: Form the directed graph \( G' \) with directed edge set \( E' = \{(u, v) \mid (u, v) \in E, \pi(u) < \pi(v)\} \)
2: Initialize \( I \leftarrow \emptyset \).
3: while \( G' \) is non-empty do
4: Let \( J \) denote the source nodes of \( G' \) (i.e. the nodes with in-degree zero).
5: Add \( J \) to \( I \).
6: For each edge \((u, v) \in E'\) for \( u \in J \), remove \( u \) and \( v \) from \( G' \).

This can be viewed as a parallel algorithm — each individual step, of identifying the source nodes and adding them to \( I \), can be done in \( O(\log n) \) time and \( O(m + n) \) processors. Alternatively, it can be viewed as a distributed algorithm, where the graph \( G \) itself corresponds to the communication network (in which case they can be executed in \( O(1) \) distributed communication rounds).

In order to obtain both parallel and distributed algorithms, we will need to bound the number of steps of the greedy algorithm when \( \pi \) is selected uniformly at random. (We can use the usual trick to generate \( \pi \) in a distributed fashion: each vertex \( v \) selects a random \( \rho(v) \) uniformly in the range \([0, 1]\); then \( \pi \) is formed by sorting in increasing order of \( \rho \).) The analysis is very similar to the proof given in [4]. The main difference is that we show that the in-degrees of the vertices are rapidly reduced during the greedy algorithm; [4] showed that the (undirected) degrees are rapidly reduced when \( G \) is an undirected graph.

**Theorem 2.1.** When \( \pi \) is selected uniformly at random, the GREEDY-FIND-LFMIS algorithm terminates in \( O(\log^2 n) \) rounds whp.

We defer the proof of Theorem 2.1 to Appendix B which shows a slightly stronger result. We also note that this corresponds to \( O(\log^3 n) \) time on an EREW PRAM, as it would require this time to find the source nodes.

The analysis of [11] gives a faster run-time when \( G \) is undirected; we conjecture that a similar analysis should hold and GREEDY-FIND-LFMIS should also run in \( O(\log n) \) rounds whp.

### 3 The variable-assignment LLLL

In the variable-assignment LLLL setting, the probability space \( \Omega \) is defined by \( n \) independent variables \( X_1, \ldots, X_n \). Each bad-event \( B \) is a conjunction of events of the form \( X_i = j \), that is, we can write

\[
B \equiv X_{i_1} = j_1 \land X_{i_2} = j_2 \land \cdots \land X_{i_r} = j_r
\]

We will abuse notation so that a bad-event \( B \) is identified with the set \( \{(i_1, j_1), \ldots, (i_r, j_r)\} \). Thus, we write \((i, j) \in B\) if one of the necessary conditions for \( B \) is that \( X_i = j \).
We define the lopsidependency structure on $B$ as follows: We have $B \sim B'$ if $(i, j) \in B, (i, j') \in B'$ for some $i \in [n]$ and $j \neq j'$. That is, $B$ demands the variable $X_i$ take on value $j$, while $B'$ demands that $X_i$ take on value $j'$; they disagree on variable $X_i$. Observe that $B \sim B'$ iff $B, B'$ are mutually exclusive events. We also write $B \approx B'$ if $B \sim B'$ or $B = B'$. This probability space has been used for a number of combinatorial constructions, including proper vertex colorings in hypergraphs [29] and $k$-SAT with bounded variable occurrences [12].

In order to keep this paper self-contained, we will analyze our LLLL algorithm under the relatively simple asymmetric LLLL, which can be stated as follows. We say that a weighting function $\mu : B \rightarrow [0, \infty)$ satisfies the asymmetric LLLL criterion with $\epsilon$-slack if

$$\forall B \in B \quad \mu(B) \geq P_{\Omega}(B)(1 + \epsilon)\left[\mu(B) + \prod_{A \sim B} (1 + \mu(A))\right]$$

In analyzing this criterion, one frequently used parameter will be $W = \sum_{B \in B} \mu(B)$. Moser & Tardos showed that if the asymmetric LLLL criterion is satisfied with 0-slack, the sequential MT algorithm terminates with probability 1 and the expected number of resamplings is at most $W$. In [17] Harris showed an alternate criterion for the convergence of the MT algorithm in this setting, and gave a parallel LLLL algorithm under this criterion. We will analyze the following nearly-matching parallel Algorithm 4.

**Algorithm 4** The parallel algorithm for the variable-assignment LLLL

1. Draw $X$ from the distribution $\Omega$.
2. while there are true bad-events do for $t = 1, 2, \ldots$:
3. Let $V_t$ denote the set of bad-events which are currently true.
4. For each $i = 1, \ldots, n$, let $a_i$ denote the current value of $X_i$.
5. For each $B \in V_t$ and each $i \in \text{Var}(B)$, draw a random variable $x_{B,i}$ from the distribution $\Omega$.
6. Construct the directed graph $G$, whose vertex set is $V_t$, and whose directed edge set is

$$E(G) = \{(B, B') \mid \text{there is } i \in \text{Var}(B) \cap \text{Var}(B') \text{ with } x_{B,i} \neq a_i\}$$

7. Find the LFMIS $I_t$ of $G$ with respect to a random permutation $\pi$.
8. For each $i \in [n]$, if there is some $B \in I_t$ with $x_{B,i} \neq a_i$, then set $X_i = x_{B,i}$.

We refer to each iteration of the main loop (lines (3) – (8)) as a round. If $i \in [n]$ has the property that some $B \in I_t$ has $x_{B,i} \neq a_i$, then we say that $i$ switches in round $t$. All steps of this algorithm (except for perhaps constructing the LFMIS and finding $V_t$) can be executed efficiently on a PRAM in $O(\log(|V_t|n))$ time.

**Proposition 3.1.** Line 8 of Algorithm 4 is well-defined: for each $i \in [n]$, there is at most one $B \in I_t$ with $x_{B,i} \neq a_i$.

**Proof.** Suppose $B, B'$ both had this property and $\pi(B) < \pi(B')$. So $G$ contains an edge $(B, B')$. But then when $B$ is added to the LFMIS $I_t$, this would force $B'$ to removed from $G$. \hfill \Box

We will show the following algorithmic result for the variable-assignment LLLL.

**Theorem 3.2.** Suppose we have a Bad-Event Checker running in time $T$ and $\text{poly}(n)$ processors.

1. Suppose that the asymmetric LLLL criterion is satisfied with $\epsilon$-slack. Then there is a parallel algorithm for the variable-assignment LLLL running in $O(\epsilon^{-1}\log(Wn)(T + \log^3(Wn)))$ time and $\text{poly}(n, W, \epsilon^{-1})$ processors whp.
2. Suppose that the symmetric LLL criterion is satisfied with $\epsilon$-slack, i.e. $e\rho(1 + \epsilon)d \leq 1$. Then there is a parallel algorithm for the variable-assignment LLLL running in $O(\epsilon^{-1}\log n(T + \log^3 n))$ time and $\text{poly}(m, n)$ whp.

3. Suppose that LLLL criterion of Harris [17] is satisfied with $\epsilon$-slack. Then there is a parallel algorithm for the variable-assignment LLLL running in $O(\epsilon^{-1}\log(Wn)(T + \log^3(Wn)))$ time and $\text{poly}(n, W, \epsilon^{-1})$ processors whp.

We now define a sequential Algorithm 5 to use for the coupling argument.

Algorithm 5 A sequential resampling algorithm for the variable-assignment LLLL

1: Draw $X$ from the distribution $\Omega$.
2: while there are true bad-events do for $t = 1, 2, \ldots$
3: Let $V_t$ denote the set of bad-events which are currently true.
4: Select a random ordering $\pi$ of $V_t$.
5: for $k = 1, \ldots, |V_t|$ do
6: If $\pi(k)$ is currently true, resample it

Proposition 3.3. Let $X_{i,t}$ denote the value of variable $i$ after round $t$. Then the random variables $X_{i,t}$ have the same distribution for Algorithm 4 and Algorithm 5.

Proof. Consider the following hybrid Algorithm 6.

Algorithm 6 A sequential resampling algorithm for the variable-assignment LLLL

1: Draw $X$ from the distribution $\Omega$.
2: while there are true bad-events do for $t = 1, 2, \ldots$
3: Let $V_t$ denote the set of bad-events which are currently true.
4: For each $B \in V_t$ and each variable $i \in \text{Var}(B)$, draw a random variable $x_{B,i}$ according to $\Omega$.
5: Select a random $\pi$ ordering of $V_t$.
6: for $k = 1, \ldots, |V_t|$ do
7: if $\pi(k)$ is currently true then
8: For each $i \in \text{Var}(\pi(k))$, set $X_i = x_{\pi(k),i}$.

We first claim that Algorithm 6 induces the same distribution on the random variables as the sequential Algorithm 5. For, Algorithm 6 does not examine the variable $x_{\pi(k),i}$ in any way before step (8), and so by the principle of deferred decisions it is equivalent to draw $X_i$ independently from $\Omega$ instead of setting $X_i = x_{\pi(k),i}$.

We next claim that Algorithm 6 induces the same distribution on the random variables as the parallel Algorithm 4. The reason for this is that if we fix the random variables $x_{B,i}$ and $\pi$, then value of the variables $X$ is identical in the two algorithms.

Thus, in order to bound the running time of the parallel algorithm, it suffices to bound the running time of Algorithm 5 which is a version of the sequential Moser-Tardos algorithm with a somewhat unusual rule for which bad-event to resample at any given time. We analyze this using an analytical tool introduced by Moser & Tardos known as the witness tree. For each $A \in V_t$, we may construct a witness tree $\hat{\tau}(A, t)$, using the following procedure:

1. Begin by setting $\hat{\tau}(A, t)$ to consist of a single root node labeled $A$.
2. For $i = t - 1, \ldots, 1$ do:
3. For $B \in I_t$, do:

4. If $\hat{\tau}(A, t)$ contains a node labeled $B'$ with $B' \approx B$, then let $v'$ be one such node of greatest depth (breaking ties arbitrarily); update $\hat{\tau}(A, t)$ by adding a new node $v$ labeled $B$ as a child of $v'$. If there are no such nodes, leave $\hat{\tau}(A, t)$ unchanged.

We refer to a labeled tree $\tau$, which could be a possible value of $\hat{\tau}(A, t)$, as a tree-structure. If the nodes of a tree-structure $\tau$ have labels $B_1, \ldots, B_t$, then we define the weight of $\tau$ by $w(\tau) = \prod_{i=1}^{t} P_{|B_i|}$. We say that a tree-structure $\tau$ with root node labeled $A$ appears if $\hat{\tau}(A, t) = \tau$ for any time $t$.

Moser & Tardos showed certain key facts about tree-structures; we recommend reviewing that work which contains more examples and details.

**Proposition 3.4 ([30]).** Suppose the asymmetric LLLL criterion is satisfied with $\epsilon$-slack. Then:

1. Any tree-structure $\tau$ with $k$ nodes appears with probability at most $w(\tau)(1 + \epsilon)^{-k}$.

2. The total weight of all tree-structures containing at least $k$ nodes is bounded from above by $W(1 + \epsilon)^{-k}$.

**Proposition 3.5.** If $A \in V_i$ for $t \geq 2$, then $A \approx B$ for some $B \in I_{t-1}$.

**Proof.** First, suppose that $A \notin V_{t-1}$. So $A$ was false at the beginning of round $t-1$ but became true by round $t$. This can only occur if some $i \in \text{Var}(A)$ switches in round $t-1$. That is, there is some $B \in I_{t-1}$ with $i \in \text{Var}(A) \cap \text{Var}(B)$ so that $x_{B,i} \neq a_i$. So $A, B$ disagree on the value of variable $X_i$, and thus $A \sim B \in I_{t-1}$ as desired.

Next, suppose that $A \in V_{t-1}$. If $A \in I_{t-1}$, then $A \approx B \in I_{t-1}$ for $B = A$. Otherwise, if $A \in V_{t-1} - I_{t-1}$, there must exist some $B \in I_{t-1}$ with $B \sim A$ (in which case we are done) or where there is $i \in \text{Var}(A) \cap \text{Var}(B)$ with $x_{B,i} \neq a_i$. This would imply that $A$ becomes false in round $t-1$ (the value of variable $x_i$ has changed, and $A$ was true in round $t-1$), which contradicts that $A \in V_t$. □

**Proposition 3.6.** If $A \in V_i$, then $\hat{\tau}(A, t)$ has depth exactly $t$.

**Proof.** For each $j \geq 1$, let us define $\tau_j$ to be the tree-structure formed by executing the loop from $i = t-1, \ldots, j$. So $\tau_1 = \hat{\tau}(A, t)$ and $\tau_1$ is the singleton node labeled $A$. We will show by induction on $j$ that $\tau_j$ has depth exactly $t-j+1$ and the nodes at depth $t-j+1$ correspond to bad-events $B \in I_j$.

The base case $j = t-1$ is very similar to the induction step, so we omit it here; we only describe the induction step where $j < t-1$. Observe that $\tau_j$ is formed from $\tau_{j+1}$ by adding nodes labeled $B \in I_j$. These nodes are all placed as children of nodes in $\tau_{j+1}$ — since $I_j$ is an independent set under $\sim$, none of the new nodes are placed as children of other new nodes. This implies that $\tau_j$ has height either $t-j+1$ or $t-j$. Furthermore, it implies that all nodes at depth $t-j+1$ (if any) will correspond to $B \in I_j$.

By induction hypothesis, the tree $\tau_{j+1}$ contains a node $v$ at depth $t-j$ which corresponds to some $B \in I_{j+1}$. By Proposition 3.5, there is some $B' \in I_j$ with $B \approx B'$. The rule for forming $\tau_j$ from $\tau_{j+1}$ will create a new node at depth $t-j+1$ labeled $B'$ (either as a child of node $v$, or as a child of another node at that depth). Thus $\tau_{j+1}$ will have depth exactly $t-j+1$. □

These facts allow us to bound the sizes of $V_t$.

**Proposition 3.7.** If the asymmetric LLLL criterion is satisfied with $\epsilon$-slack, then $\mathbb{E}[\sum_t |V_t|] \leq W$ and whp $V_t = \emptyset$ for $t \geq \Omega(\epsilon^{-1} \log(Wn))$. 

9
Proof. Since the values of $\hat{\tau}(A, t)$ are all distinct, the size of $V_t$ is at most the number of tree-structures $\tau$ satisfying $\hat{\tau}(A, t) = \tau$. By Proposition 3.6, every such tree-structure contains at least $t$ nodes. Hence, $\sum_{j \geq t} |V_j|$ is at most the number of appearing tree-structures with $\geq t$ nodes.

By Proposition 3.4, this implies that $\mathbb{E}[\sum_{j \geq t} |V_j|]$ is at most the weight of all tree-structures with $\geq t$ nodes, which is at most $W(1 + \epsilon)^{-t}$. Setting $t = 0$, we see that $\mathbb{E}[\sum_{t} |V_t|] \leq W$. Setting $t \geq c^{-1} \log(Wn)$, we see that $\mathbb{E}[|V_t|] \leq n^{-\Omega(1)}; \text{by Markov's inequality this implies that } V_t = \emptyset \text{ whp.}$

Proof of Theorem 3.2. To show part (1), note that Propositions 3.7 and 3.3 ensure that Algorithm 1 terminates after $O(\epsilon^{-1} \log(Wn))$ rounds whp. In each round, we must compute the LFMIS of $V_t$; by Theorem 2.1 this takes $O(\log^3 |V_t|)$ expected time. Proposition 3.7 shows $\mathbb{E}[\sum_{t} |V_t|] \leq W$ and so $\sum_{t} |V_t| \leq Wn^{100}$ whp. By concavity, this implies that $\sum_{t} \log^3 |V_t| \leq O(\log^3(Wn))$ whp. The bound on processor count is similar.

To show part (2), note that if the symmetric criterion is satisfied, then by setting $\mu(B) = eP_{2\Omega}(B)$ we also satisfy the asymmetric LLLL criterion with $\epsilon$-slack and $W \leq O(n)$. Furthermore, even if $\epsilon \leq 1$, then we can satisfy the asymmetric LLLL criterion with $\epsilon$-slack for $\epsilon = \text{poly}(1/d)$. So, we may assume that $\epsilon^{-1} \leq \text{poly}(d) \leq \text{poly}(m)$.

Part (3) can easily be shown using arguments from Harris [17].

Harris gave in [17] a number of applications of the LLLL, which translate almost immediately into parallel algorithms.

Proposition 3.8. Suppose we have a $k$-SAT instance with $m$ clauses, in which each variable appears in at most

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

clauses, where $\epsilon > 0$. Then there is a parallel algorithm to find a satisfying assignment using $O(\epsilon^{-1} \log^4(mn))$ time and $\text{poly}(m, n, \epsilon^{-1})$ processors whp.

Proof. This result follow from Proposition 3.2 using the criterion of Harris [17]. Using arguments from that paper (specifically, Theorem 4.1), we see that the Harris criterion can be satisfied with $\epsilon$-slack under these conditions using the weighting function

$$\mu(B) = \frac{1 + \epsilon}{(2 - 2/kk)} \text{ for all } B \in \mathcal{B}$$

Furthermore, in this case, we have $W \leq m$ and we can implement a bad-event checker by running through every clause.

3.1 Distributed algorithms

A common task in distributed communication problems is to solve a graph problem (e.g. vertex coloring) on a graph $G$, in which there are limited communication pathways among the nodes in a graph. One popular model is LOCAL, in which there are synchronous communication rounds, and each node may only communicate with its neighbors in a given round. A stricter model is CONGEST, in which each node can only send $O(\log n)$ bits to each neighbor per timestep.

Since the LLL is a basic method for combinatorics, it is natural to use the LLL or its variants to solve the resulting distributed communication problem. In this setting, we need to implement the LLL algorithm as a distributed process, in which each vertex corresponds to a variable and each violated constraint (e.g. a monochromatic edge) corresponds to a bad-event. Typically in
such cases, the number of bad-events \( m \) is a polynomial in the size of the graph \( G \), and bad-events corresponding to edges or vertices of \( G \) are dependent iff the distance between them in \( G \) is bounded by \( O(1) \). In these cases, it is convenient to translate the distributed computation on \( G \) onto a new graph \( H \), whose vertices are the bad-events \( B \), and where there is an edge from \( B \) to \( B' \) on \( H \) if \( B, B' \) overlap on some variable. Each vertex \( B \) of \( H \) will keep track of the value of \( \text{Var}(B) \); we can view the distributed computation on \( G \) as a computation on \( H \), with the goal to find a globally consistent variable assignment.

Vertices in \( H \) are joined by an edge if their corresponding vertices of \( G \) have distance \( O(1) \); thus, a distributed round of computation in \( H \) can be simulated by \( O(1) \) distributed rounds of computation in \( G \). Thus, it suffices to bound the running time of a distributed algorithm on \( H \) (up to constant factors). See \[7\] for a much more thorough discussion of this model of computation and applications to a number of graph-coloring problems.

We can transform our parallel algorithm into a distributed algorithm for \( H \), in the obvious way.

**Proposition 3.9.**

1. Suppose that the asymmetric LLLL criterion is satisfied with \( \epsilon \)-slack. There is a distributed algorithm in the CONGEST model for the variable-assignment LLLL running in \( O(\epsilon^{-1}\log^3(Wn)) \) time whp.

2. Suppose that the symmetric LLL criterion is satisfied with \( \epsilon \)-slack, i.e. \( cp(1 + \epsilon)d \leq 1 \). Then there is a distributed algorithm in the CONGEST model for the variable-assignment LLLL running in \( O(\epsilon^{-1}\log^3 n) \) time whp.

3. Suppose that the LLLL criterion of Harris \([17]\) is satisfied with \( \epsilon \)-slack. There is a distributed algorithm in the CONGEST model for the variable-assignment LLLL running in \( O(\epsilon^{-1}\log^3(Wn)) \) time whp.

**Proof.** All of the steps in a round \( t \) (except from the computation of the LFMIS) can be implemented in \( O(1) \) communication steps in the CONGEST model. The GREEDY-LFMIS algorithm can be implemented in \( O(\log^2 n) \) rounds. Observe that Algorithm 4 only creates an edge between \( B, B' \) if \( B, B' \) overlap on a variable. Thus, each individual round of the GREEDY-LFMIS algorithm can be implemented in \( O(1) \) communication steps in CONGEST.

**Proposition 3.10.** Suppose \( H \) is a hypergraph in which each edge has cardinality \( k \) and each vertex appears in at most \( L \) edges. Then there is a CONGEST procedure in \( O(\epsilon^{-1}\log^3 n) \) rounds to construct a non-monochromatic \( c \)-coloring of \( H \) for \( L \leq \frac{c^k(1-1/k)^{k-1}}{\epsilon(c-1)(1+\epsilon)} \).

**Proof.** We had a bad-event that a given edge of \( H \) is monochromatic with any given color. Bad-events are connected if the corresponding edges share a vertex; hence, distributed computation in \( H \) is equivalent to distributed computation on the graph whose vertex set is the bad-events. Using arguments of \[17\], one can show that the criterion of Harris is satisfied with \( \epsilon \)-slack and \( W \leq \text{poly}(n) \).
Let us define $Z = [n] \times [n]$; we identify such a bad-event $B$ with the corresponding set $\{(x_1, y_1), \ldots, (x_r, y_r)\} \subseteq Z$. We will define our dependency graph by defining a relation $\sim$ on $Z$, and setting $B \sim B'$ if $B \ni z \sim z' \in B'$; the relation $\sim$ will depend on the precise choice of $W$.

As usual, we seek an efficient algorithm to find some permutation $\pi \in W$ avoiding all these bad-events. Our overall plan is to define an appropriate resampling oracle and use the generic LLL algorithm of Harvey & Vondrák [21]. In order to be able to simulate this algorithm in a parallel setting, we define the resampling oracle $R$ to have a particularly nice algebraic form. First, we will have some group $G$ acting on $W$, which we call the resampling group; next, for each bad-event $B$, we have a set $R(B) \subseteq G$, which we call the resampling generator for $B$; we then define the resampling oracle by

$$R_B(\pi) = r\pi$$

where $r$ is drawn uniformly from $R(B)$.

The precise definition of $G$ and $R(B)$ depends on $W$; the analysis is quite technical and problem-specific. However, there are only a few high-level algebraic properties it must satisfy in order to provide an efficient parallel resampling algorithm. We are thus able to provide a unified treatment of a number of distinct probability spaces, including the full set of permutations (i.e. $W = S_n$), the set of hamiltonian cycles of $K_n$ (corresponding to the case in which $W$ consists of $n$-long cycles), and the set of perfect matchings of $K_n$ (corresponding to the case in which $W$ consists of $n/2$ transpositions). We find it beautiful that one can separate the “clean” part of the parallel LLLL algorithm from the “dirty” part in this way.

We provide an overview now of Section 4. In Section 4.2, we define the resampling generator $R(B)$, and the properties it must satisfy. For technical reasons, we will define $R(u)$ for all sets $u \subseteq Z$ (not only for when $u$ is itself a bad-event); this depends on $W$ but not on $B$. We define a series of “axioms” that $R$ must satisfy in order to lend itself to parallel algorithms. In Appendix C, we list a number of probability spaces which can be cast into this framework; these proofs are technically quite complex.

In Section 4.4, we show a sequential algorithm for the permutation LLLL, with respect to any set $W \subseteq S_n$ satisfying the criteria of Section 4.2. The key idea here is to show that $R_B$ is an appropriate resampling oracle for the probability space $\Omega$.

In Section 4.5, we show how to parallelize the algorithm of Section 4.4. Our main result in this section is the following:

**Theorem 4.1.** Suppose that we a set of bad-events $B$ on a probability space $W \subseteq S_n$ with a resampling rule satisfying the criteria of Section 4.2. Also, suppose we have a Bad-Event Checker running in time $T$ and $\text{poly}(n)$ processors.

1. If the Shearer criterion is satisfied with $\epsilon$-slack, then there is a parallel algorithm for the permutation-LLL running in $O(\epsilon^{-1} \log n)(T + \log^3 n)$ time and $\text{poly}(n/\epsilon)$ processors whp.

2. If the symmetric criterion is satisfied with $\epsilon$-slack, i.e. $\epsilon p(1 + \epsilon)d \leq 1$, then there is a parallel algorithm for the permutation LLL running in $O(\epsilon^{-1} \log n(T + \log^3 n))$ time and $\text{poly}(n/\epsilon)$ processors whp.

Finally, in Section 4.6, we discuss a few applications of this algorithm.

### 4.1 Notation

We let $S_n$ denote the set of permutations on the ground set $[n]$. We use $(a \ b)$ to refer to the permutation which swaps $a$ and $b$ (and leaves other elements unchanged). We use the multiplication
convention that $\sigma_1 \sigma_2$ is the functional composition $\sigma_1 \circ \sigma_2$, that is, the function sending $x$ to $\sigma_1(\sigma_2(x))$. Similarly, if $X_1, X_2$ are groups, then we write $X_1X_2 = \{\sigma_1 \sigma_2 \mid \sigma_1 \in X_1, \sigma_2 \in X_2\}$.

Given any pair $(x, y) \in Z$, we define $\langle z \rangle$ to be the set of permutations $\pi \in W$ such that $\pi(x) = y$; for $u \subseteq Z$ we also define $\langle u \rangle = \bigcap_{z \in u} \langle z \rangle$.

We say that $u \subseteq Z$ is valid if $\langle u \rangle \neq \emptyset$; we define $U$ to be the set of all valid subsets of $Z$.

4.2 The properties required of the resampling generator

Given some $W \subseteq S_n$, there are three ingredients to define our generic resampling oracle. The first is the resampling group, which is a group $G$ acting on $W$; we have that $gw \in W$, as well as $g_1(g_2(w)) = (g_1g_2)w$. The second ingredient is the resampling generator $R : U \rightarrow 2^G$. The third ingredient is a binary, symmetric, reflexive relation $\sim$ on $Z$; roughly speaking, $(x, y) \sim (x', y')$ if the event $\pi(x) = \pi(y)$ can increase the probability that $\pi(x') = \pi(y')$. The definitions of three objects $G, R, \sim$ depend heavily on the precise form of the ambient permutation space $W$.

For any $u, v \in U$ such that $\langle u \cup v \rangle \neq \emptyset$, we define $R(u; v)$ to be the set of $r \in R(u)$ with the property that $r \pi \in \langle v \rangle$ for all $\pi \in \langle u \cup v \rangle$.

We extend $\sim$ to a binary relation $\sim$ on $U$ by setting $u \sim v$ if $u \nexists (x, y) \sim (x', y') \in v$.

With these definitions, we say that $(G, R, \sim)$ is a resampling rule for $W$ if it satisfies the following properties:

\begin{enumerate}
\item[(C1)] (Efficiency) There is an algorithm that, given any $u \in U$, returns a uniformly generated $r \in R(u)$, running in expected time $\text{polylog}(n)$ and using $\text{poly}(n)$ processors. Likewise, we can compute $gr$ and $g_1g_2$ where $\pi \in W$ and $g_1, g_2 \in G$. (More specifically, in order avoid being a bottleneck for our algorithms, we require that these procedures run in time $O(\log^3 n)$.)

\item[(C2)] (Stability) The group $G$ acts on $W$. (This is the definition of $G$, but we state it here explicitly as an axiom as well.)

\item[(C3)] (Uniformity) For any $u \in U$ and any $\pi' \in W$, there is exactly one ordered pair $(\pi, r)$, where $\pi \in \langle u \rangle$ and $r \in R(u)$, such that $\pi' = r \pi$.

\item[(C4)] (Locality) If $u \subseteq U$, $z \in Z$ and $\pi \in \langle u \rangle - \langle z \rangle$, then $r \pi \notin \langle z \rangle$ for all $r \in R(u)$.

\item[(C5)] (Absorption) For $u, v \subseteq U$ and $u \nexists v$ and $\pi \in \langle u \cup v \rangle$, we have $R(v)R(u; v)\pi = R(u \cup v)\pi$.

\item[(C6)] (Consistency) If $u \subseteq U$ and $z \in Z$ with $u \nexists z$ then for any $r \in R(u)$ exactly one of the following conditions holds:
\begin{enumerate}
\item[(a)] For all $\pi \in \langle u \cup z \rangle$ we have $r \pi \notin \langle z \rangle$
\item[(b)] For all $\pi \in \langle u \cup z \rangle$ we have $r \pi \in \langle z \rangle$
\end{enumerate}

We make a few simple observations about these properties.

**Proposition 4.2.** If $u, v \subseteq U$ and $u \nexists v$, then $u \cup v \subseteq U$.

**Proof.** Suppose not, so that $\langle u \cup v \rangle = \emptyset$. Since $v \subseteq U$, let $\pi' \in \langle v \rangle$. By Property (C3), there exists $\pi \in \langle u \rangle$ and $r \in R(u)$ such that $\pi' = r \pi$. Since $u \cup v \nexists U$, then $\langle u \rangle - \langle v \rangle = \langle u \rangle$. So $\pi \in \langle u \rangle - \langle v \rangle$ but $\pi' \in \langle v \rangle$; this contradicts (C4). \[\square\]

**Proposition 4.3.** If $u, v \subseteq U$ and $u \nexists v$, then for $\pi \in \langle u \rangle - \langle v \rangle$ and $r \in R(u)$ we have $r \pi \notin \langle v \rangle$.

**Proof.** Since $\pi \in \langle u \rangle - \langle v \rangle$, there must exist some $z \in v$ with $\pi \in \langle u \rangle - \langle z \rangle$. By (C4) we have $r \pi \notin \langle z \rangle$. So $r \pi \notin \langle v \rangle$.
\[\square\]
Proposition 4.4. If $u,v \in U$ and $u \not\sim v$ then for any $r \in R(u)$ exactly one of the following conditions holds:

(a) For all $\pi \in \langle u \cup v \rangle$ we have $r\pi \not\in \langle v \rangle$

(b) For all $\pi \in \langle u \cup v \rangle$ we have $r\pi \in \langle v \rangle$

Proof. Suppose that (a) does not hold. So there is some $\pi(u \cup v)$ with $r\pi \not\in \langle v \rangle$. So $r\pi \not\in \langle z \rangle$ for some $z \in v$. Since $\pi \in \langle u \cup z \rangle$, by (C6) every $\pi \in \langle u \cup z \rangle$ must have $r\pi \not\in \langle z \rangle$. In particular, for any $\pi \in \langle u \cup v \rangle$ we have $\pi \in \langle u \cup z \rangle$ and hence $r\pi \not\in \langle z \rangle$ and therefore $r\pi \not\in \langle v \rangle$.

4.3 Multiple permutations

There is a form of the permutation LLLL which seems slightly more general than the one we have considered in Section 4. Suppose the probability space $\Omega$ is defined by selecting $\langle \pi \rangle$ thereof; a bad-event in this setting would be $\langle \pi \rangle$. We define $\langle \pi \rangle$ to be the set of permutations $S_{n_1} \times S_{n_2} \times \cdots \times S_{n_k}$ as a subset of the larger permutation group $S_{n_1+n_2+\cdots+n_k}$.

We do so via a cartesian-product construction. Let $W_1, W_2$ be subsets of $S_{n_1}, S_{n_2}$ respectively, with resampling rules $(G_1, R_1, \sim_1)$ and $(G_2, R_2, \sim_2)$. We will define a new resampling rule $(G, R, \sim)$ for $W_1 \times W_2 \subseteq S_{n_1+n_2}$.

It will be useful to relabel the elements, so that $W_1$ is a subset of permutations on a ground-set $A_1$ and $W_2$ is a subset of permutations on a ground-set $A_2$, where $A_1, A_2$ are disjoint sets with $|A_1| = n_1$ and $|A_2| = n_2$. Then we define $W$ to be the set of permutations $W_1 W_2$ on ground set $A = A_1 \cup A_2$.

We define $\sim$ by setting $z_1 \subseteq z_2$ iff one the following conditions hold:

1. $z_1 \subseteq A_1 \times A_1, z_2 \subseteq A_1 \times A_1$, and $z_1 \sim_1 z_2$; OR
2. $z_1 \subseteq A_2 \times A_2, z_2 \subseteq A_2 \times A_2$, and $z_1 \sim_2 z_2$; OR
3. $z_1 = z_2$.

We define $G = G_1 G_2$. For $u = u_1 \cup u_2$ where $u \in U_1$ and $u_2 = U_2$, we define $R(u) = R_1(u_1) R_2(u_2)$.

One may easily check that $\times$ is an associative map (up to isomorphism) and hence one may easily define a cartesian product of multiple resampling rules $W_1, \ldots, W_k$. Crucially, the cartesian product of a polynomial number of resampling rules is itself a resampling rule.

Proposition 4.5. Suppose that $(G_1, R_1, \sim_1), \ldots, (G_k, R_k, \sim_k)$ satisfy (C1) — (C6), and $k = poly(n)$. Then the resampling rule $(G_1, R_1, \sim_1) \times \cdots \times (G_k, R_k, \sim_k)$ satisfies (C1) — (C6) as well for the set $W = W_1 \ldots W_k$.

Proof. Observe that any $\pi \in W$ must map $A_i$ to $A_i$. Thus, any $u \in U$ can be written uniquely as $u = u_1 \cup u_2 \cup \cdots \cup u_k$ where $u_i \in U_i$. This decomposition can be executed in $O(\log n)$ time. Also, observe that $\langle u \rangle = \langle u_1 \rangle \cap \langle u_2 \rangle \cap \cdots \cap \langle u_k \rangle$. With these facts one may easily check the properties (C1) — (C6).
4.4 A sequential algorithm for the permutation LLLL

Before we describe our parallel algorithm, we begin with a simple sequential algorithm to find a good \( \pi \in W \); this is simply a special case of the generic LLL resampling-oracle framework of [21].

Algorithm 7 The sequential resampling algorithm

1: Draw \( \pi \) uniformly from \( W \).
2: while there is some bad-event true on \( \pi \) do
3:   Let \( B \) be any bad-event which is true on \( \pi \).
4:   Update \( \pi \leftarrow R_B(\pi) \).

That is, whenever a bad-event is true, we “resample” it (by applying our resampling oracle \( R \)).

Proposition 4.6. The randomized function \( R \) is a commutative resampling oracle, in the sense of [21, 25].

Proof. By property (C2), \( R_B(\pi) \in W \) for any \( \pi \in W \). Thus \( R_B \) is a randomized function mapping \( W \) to \( W \).

The definition of a resampling oracle has two parts. First, if \( \pi \) is selected uniformly at random from \( W \), conditional on that \( \pi \) satisfies bad-event \( B \), then \( R_B(\pi) \) must be distributed uniformly in \( W \). Note that the set of permutations in \( W \) for which \( B \) is true is precisely the set \( \langle B \rangle \). Property (C3) ensures that \( R_B \) maps the uniform distribution on \( \langle B \rangle \) to the uniform distribution on \( W \).

The second part is that if \( B_1 \not\sim B_2 \) and \( \pi \) satisfies the property that \( B_1 \) is true on \( \pi \) and \( B_2 \) is false on \( \pi \), then \( B_2 \) remains false on \( R_B(\pi) \). This follows immediately from Proposition 4.3.

In order to show that the resampling action is commutative, we first note that \( \pi \) is atomic, in the sense that \( \pi \) and \( B \) are fixed and we know that \( \pi' = R_B(\pi) \), then there is only one possible value for \( \pi' \). This follows immediately from (C3). For an atomic resampling action, (weak) commutativity requires that if \( \pi' = R_B(\pi) \) and \( \pi' \) satisfies \( B_2 \) and \( \pi'' = R_B(\pi') \), then there is a positive probability that one can obtain \( \pi'' = R_B(R_B(\pi)) \).

So suppose that \( \pi' = r_1 \pi \) for \( r_1 \in R(B_1) \) and \( \pi'' = r_2 \pi' \) for \( r_2 \in R(B_2) \) and \( \pi' \in \langle B_2 \rangle \). By property (C6), necessarily \( r_1 \in R(B_1; B_2) \). Therefore, by property (C5), there exists \( r \in R(B_1 \cup B_2) \) such that \( \pi'' = r \pi \).

Applying (C5) a second time, this implies that there is \( r'_2 \in R(B_2; B_1) \) and \( r'_1 \in R(B_1) \) such that \( \pi'' = r'_1 r'_2 \pi \). This means that \( \pi'' \) can be realized as \( \pi'' = R_B(R_B(\pi)) \).  \( \square \)

4.5 Parallelizing the sequential algorithm

We now turn to developing a parallel form of Algorithm 7 using two key ideas. First, we use the Kolmogorov framework to divide the resampling steps into distinct rounds. Second, we use the algebraic form of \( R \) to execute multiple resampling steps in parallel.
Algorithm 8 The parallel algorithm for the permutation LLL

1: Draw the permutation $\pi \in W$ uniformly at random.
2: While there are true bad-events do for $t = 1, 2, \ldots$:
3: Let $V_t$ denote the set of bad-events which are currently true,
4: For each $B \in V_t$, independently and uniformly select $r_B \in R(B)$.
5: Form directed graph $G$ on vertex set $V_t$, with edge set
   \[\{(B, B') \mid B \sim B' \text{ or } r_B \notin R(B; B')\}\]
6: Choose a permutation $\tau$ on $V_t$ uniformly at random.
7: Find the LFMIS $I$ of $G$ with respect to $\tau$.
8: Enumerate $I = \{B_1, \ldots, B_s\}$, sorted so that $\tau(B_1) < \tau(B_2) < \cdots < \tau(B_s)$. Update $\pi$ by
   \[\pi \leftarrow r_{B_s} r_{B_{s-1}} \cdots r_{B_2} r_{B_1} \pi\]

Let us note that $|I| \leq n^2$; the reason for this is that for any $z \in Z$, there can be at most one $B \in I$ with \(z \in B\). Thus, step (8) can be performed in $O(\log n)$ time using property (C1); we first compute the composition $r_{B_s} \cdots r_{B_1}$ via a multiplication tree and then apply it to $\pi$. Similarly, step (7) can be performed in time $O(\log^3 |V_t|)$, and a Bad-Event Checker can be used to execute step (3).

Thus, our main task will be to show that this algorithm terminates after a small number of rounds. In order to do so, we will couple Algorithm 8 to a form of Algorithm 7 (with an unusual choice of resampling rule).

Algorithm 9 A sequential Swapping Algorithm

1: Draw the permutation $\pi \in W$ uniformly.
2: While there are true bad-events do for $t = 1, 2, \ldots$:
3: Let $V_t$ denote the set of bad-events which are currently true.
4: Mark every element of $V_t$ as “alive”
5: Choose a permutation $\tau$ on $V_t$ uniformly at random.
6: Set $I_t = \emptyset$
7: For $j = 1, \ldots, |V_t|$
   8: if $\tau(j)$ is currently marked alive then
      9: Update $I_t \leftarrow I_t \cup \{\tau(j)\}$.
     10: Update $\pi \leftarrow r_{\tau(j)}(\pi)$.
    11: For any $B \in V_t$ with either (i) $B$ is false on $\pi$ or (ii) $B \sim \tau(j)$ do
       12: Mark $B$ as dead.

We refer to each iteration of the loop (lines 3 — 13) of Algorithm 9 as a round, and similarly we refer to each iteration of the loop (lines 3 — 8) of Algorithm 8 as a round.

By the principle of deferred decisions there is no difference in selecting the resampling action $r_B$ for any $B \in V_t$ in an “online” way (as in Algorithm 9) or in a preprocessed way (as in Algorithm 8). We further claim that if we fix the value of $\pi, \tau, r_B$ in a given round, then the two algorithms produce precisely the same result. This implies that the algorithms have the same probability distribution on $\pi$. In particular, running time bounds on Algorithm 9 will imply corresponding bounds on Algorithm 8.

This equivalence between the two algorithms follows from the following result:
Proposition 4.7. If the random variables \( r_B, \tau, \pi \) are all fixed at the beginning of round \( t \) and \( I, I' \) are the LFMIS produced for Algorithms 8 and 9 respectively for round \( t \), then \( I = I' \).

Proof. Let \( \pi_j \) denote the permutation after iteration \( j \) of round \( t \) (and \( \pi_0 \) is the permutation at the beginning of round \( t \)). Enumerate \( V_i \) in sorted order as \( \{B_1, B_2, \ldots, B_k\} \) where \( \tau^{-1}(B_1) < \tau^{-1}(B_2) < \cdots < \tau^{-1}(B_k) \). We write \( r_k \) as short-hand for \( r_{B_k} \).

With this notation, observe that \( B_j \in I \) iff there is no \( i < j \) with \( B_i \in I \) and either (a) \( B_i \sim B_j \) or (b) \( r_i \in R(B_i; B_j) \). Similarly, \( B_j \in I' \) iff there is no \( i < j \) with \( B_i \in I' \) and either (a) \( B_i \sim B_j \) or (b) \( B_j \) is false on \( \pi_i \).

We show by induction on \( j \) that \( B_j \in I \) iff \( B_j \in I' \).

Suppose first that \( B_j \in I' - I \). Since \( B_j \notin I \), there must be some \( i < j \) with \( B_i \in I \) such that \( B_i \sim B_j \) or \( r_i \notin R(B_i; B_j) \). In the former case, by our induction hypothesis \( B_i \in I' \) and this would contradict that \( B_j \in I' \). In the latter case, note that since \( B_j \in I' \), it must be that \( B_j \) is true on \( \pi_i \) and \( \pi_{i-1} \) and \( B_i \) is true on \( \pi_{i-1} \). Thus, \( \pi_{i-1} \in \langle B_i \cup B_j \rangle \) and \( \pi_i = r_i \pi_{i-1} \in \langle B_j \rangle \). By Proposition 4.2 this implies that \( r_i \in R(B_i; B_j) \).

Next, suppose that \( B_j \in I - I' \). Since \( B_j \notin I' \), there must be some \( i < j \) with \( B_i \in I' \) such that \( B_i \sim B_j \) or \( B_j \) is false on \( \pi_i \). Let \( i \) be minimal subject to these conditions. In the former case, by our induction hypothesis \( B_i \in I \) and this would contradict that \( B_j \in I \). Otherwise, by minimality of \( i \), it must be that \( B_j \) becomes false after resampling \( B_i \). Thus, \( \pi_{i-1} \in \langle B_i \cup B_j \rangle \) and \( \pi_i = r_i \pi_{i-1} \notin \langle B_j \rangle \). By Proposition 4.2 this implies that \( r_i \notin R(B_i; B_j) \).

This sequential Algorithm \( 9 \) is part of the Kolmogorov framework, thus results of Kolmogorov \( 25 \) show it terminates in a small (typically polylogarithmic) number of rounds. We will show that individual rounds of Algorithm \( 8 \) can be implemented efficiently in parallel. We also strengthen this “black-box” application of \( 25 \) by taking advantage of the fact that, during intermediate states of the parallel LLL algorithm, the permutation \( \pi \) has a “random-like” distribution. This phenomenon was exploited by Harris & Srinivasan \( 18 \) and Haeppler & Harris \( 15 \) for the standard MT algorithms. Some technical details of these constructions are discussed in more detail in Appendix \( \mathcal{A} \). This will also allow us to give running time bounds which do not depend on the (somewhat mysterious) weighting function used in formulations of the asymmetric LLL and cluster-expansion LLL.

As we execute Algorithm \( 9 \) we let \( T_j \) denote the total number of resamplings before round \( j \) (so \( T_1 = 0 \)). We also set \( \pi^{(i)} = \pi^{T_i} \) denote the value of the permutations immediately at the beginning of round \( i \).

Lemma 4.8. If \( A \in V_t \) for \( t \geq 2 \), then \( A \sim B \) for some \( B \in I_{t-1} \).

Proof. By definition, \( A \) must be true on the permutations \( \pi^{(i)} \). Observe that either \( A \) is true at time \( T_{i-1} \) or \( A \sim B \in I_{t-1} \); otherwise, by property (C4), \( A \) would remain false false after all the resamplings in round \( t \) - 1.

If \( A \sim B \in I_{t-1} \) we are done. Otherwise, suppose \( A \in V_{t-1} - I_{t-1} \). This can only be the case if \( A \) was marked as dead in round \( t - 1 \). Suppose that this first occurs at time \( i \), during the resampling of some \( B \in I_{t-1} \). If \( A \sim B \), we are done; otherwise, \( A \) must be false on the permutation \( \pi^i \).

However, \( A \) is true at the beginning of round \( t - 1 \). By (C4), the only way it can later become false during round \( t - 1 \) is if there is some \( B' \) resampled between times \( T_{i-1} \) and \( i \) with \( B' \sim A \). Thus \( B' \in I_{t-1} \), \( B' \sim A \) as desired.

As we describe in more detail in Appendix \( \mathcal{A} \) for each bad-event \( A \) that occurs at some time \( t \) during Algorithm \( 9 \) one may define a corresponding stable-set sequence \( S(A, t) \). If \( B \in V_t \), then \( B \) is true after \( T_i \) resamplings, and so we may form the corresponding stable-set sequence \( S(B, T_i) \).
**Proposition 4.9.** For $B \in I_i$, then $\hat{S}(B, T_i)$ has depth at least $i$.

Proof. We show this by induction on $i$. When $i = 1$, this is vacuously true.

Consider $\hat{S}(B, T_i) = \langle S_0, \ldots, S_N \rangle$ for $B \in I_i$. For the induction step, note that by Lemma 4.8 there is some $A \sim B$ with $A \in I_{i-1}$. By induction hypothesis, $\hat{S}(A, T_{i-1})$ has depth at least $i - 1$; let us enumerate this as

$$\hat{S}(A, T_{i-1}) = \langle S'_0, S'_1, \ldots, S'_{N'} \rangle$$

where $N' \geq i - 2$.

We now observe that $S'_j \subseteq S_{j+1}$ for $j = 0, \ldots, N'$. In particular, $N \geq N' + 1$. □

**Proposition 4.10.** The values of $\hat{S}(B, T_i)$ are all distinct, where $i$ ranges over all rounds and $B \in V_t$.

Proof. If $B \neq B'$, then obviously $\hat{S}(B, T_i) \neq \hat{S}(B', T_i)$ (they have different roots). If $i < i'$, then an argument similar to Proposition 4.9 shows that the depth of $\hat{S}(B, T_i)$ is strictly larger than that of $\hat{S}(B, T_i_1)$.

□

**Corollary 4.11.** Whp, Algorithm 8 terminates after $O(\epsilon^{-1} \log n)$ rounds and $\sum_t |V_t| \leq O(\epsilon^{-1} n^{100})$.

Proof. By Proposition 4.7 it suffices to show this for Algorithm 9. By Proposition 4.9 if $V_t \neq \emptyset$, then there is some $S \in \Gamma$ which appears and whose depth is at least $t$. Thus, if the algorithm runs for $t$ stages, then, then there must be at least $t/2$ appearing singleton stable-set sequences whose depth is at least $t/2$.

The dependency graph can be covered in $n^2$ cliques — namely, for each $z \in [n] \times [n]$, the set of all bad-events $B$ such that $z \in B$ is a clique. By Proposition A.5 the expected number of appearing singleton stable-set sequences with at least $t/2$ nodes is at most $en^2 t^2 (1 + \epsilon)^{-t/2}$. So by Markov’s inequality, the probability of the algorithm running for $t$ stages is at most $\frac{en^2 t^2 (1 + \epsilon)^{-t/2}}{t^2} \leq en^2 (1 + \epsilon)^{-t/2}$; this is $n^{-\Omega(1)}$ for $t = O(\epsilon^{-1} \log n)$. (In this case, the condition that $t/2 \geq 1 + 1/\epsilon$ is clearly satisfied as well).

By Proposition 4.10 each $B \in V_t$ corresponds to a distinct $\hat{S}(B, T_i) \in \Gamma$. Thus, $\sum_t |V_t|$ is at most the number of appearing $S \in \Gamma$. Therefore, by Proposition A.5 $E[\sum_t |V_t|] \leq O(n^2/\epsilon)$ and so $\sum_t |V_t| \leq O(n^{100}/\epsilon)$ whp.

□

We are now ready to prove Theorem 4.1.

Proof of Theorem 4.1. We show only the first part; the second follows immediately. Let $s = cc^{-1} \log n$. Corollary 4.11 ensures that Algorithm 8 terminates after $s$ rounds whp, where $c$ is a sufficiently large constant. In each round, we must execute the Bad-Event Checker (which takes time $T$) and perform other tasks which take time $O(\log^2 |V_t|)$ whp. Finally, we must compute the LFMIS of $V_t$. By Theorem 2.1 this takes time $O(\log^3 |V_t|)$ whp.

Thus, the overall time is at most $O(\sum_{t=1}^s \log^3 |V_t| + \sum_{t=1}^s \log^3 |V_t|)$. By concavity, we have

$$\sum_{t=1}^s \log^3 |V_t| \leq s \log^3 \left(\frac{\epsilon^{-1} n^{100}}{cc^{-1} \log n} \right) = O(\epsilon^{-1} \log^4 n)$$

and a similar bound holds for the term $\sum_t (\log |V_t|)$. The overall time is $O(\epsilon^{-1} \log n (T + \log^3 n))$. □
4.6 Applications

We illustrate with the classic application of the permutation LLL to 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 \( A(i, \pi(i)) = A(i', \pi(i')) \).

**Proposition 4.12.** Suppose that each color appears at most \( \Delta \) times in \( A \). Then, we can find a Latin transversal \( \pi \in S_n \) in \( O(\log^4 n) \) time and \( \text{poly}(n) \) processors, under one of the following conditions:

1. If \( \Delta \leq 0.105n \);
2. If \( \Delta \leq 0.105n \) and \( n \) is even, and \( A \) has the property that \( A(x, y) \neq A(y, x) \) for \( x \neq y \), then we can furthermore ensure that the cycle-structure of \( \pi \) has exactly \( n/2 \) transpositions
3. If \( \Delta \leq 0.052n \), then we can furthermore ensure that the cycle-structure of \( \pi \) has exactly one \( n \)-cycle.\(^4\)

**Proof.** The cluster-expansion LLL criterion is satisfied with slack of \( \epsilon = O(1) \) in these cases. \( \square \)

As another application of our algorithm, we consider the strong chromatic number. Suppose we have a graph \( G \), with a given partition of the vertices into \( k \) blocks each of size \( b \), i.e., \( V = V_1 \uplus \cdots \uplus V_k \). We would like to find a proper \( b \)-coloring such that every block has exactly \( b \) colors; 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 has studied the strong chromatic number of graphs, typically in terms of their maximum degree \( \Delta \). In [22], Haxell showed that such a coloring exists when \( b \geq (11/4 + \epsilon)\Delta \) and \( \Delta \) is sufficiently large; this is the best bound currently known. Furthermore, the constant 11/4 cannot be improved to any number strictly less than 2. In [19], a variety of algorithms are given for constructing strong colorings. These algorithms require \( b \geq c\Delta \), where \( c \) is a constant; however, the constant \( c \) does not match those given by [22]. Two algorithms in particular are noteworthy: the first is a sequential algorithm requiring \( b \geq 5\Delta \), and the second is an RNC algorithm requiring \( b \geq \frac{256}{27}(1 + \epsilon)\Delta \) (precise running time not stated). Our permutation LLLL algorithms gives a crisper version of this, which is perhaps the first parallel algorithm for strong-coloring with reasonable bounds on both the number of colors and the running time:

**Proposition 4.13.** For \( b \geq \frac{256}{27}(1 + \epsilon)\Delta \), a strong coloring of \( G \) can be found in \( O(\epsilon^{-1} \log^4 n) \) time.

Finally, we consider a hypergraph packing problem introduced in [26]. We are given two \( r \)-uniform hypergraphs \( H_1, H_2 \) and an integer \( n \), and we want to find a pair of injective maps \( \phi_1 : V(H_1) \to [n], \phi_2 : V(H_2) \to [n] \) with the property that \( \phi(H_1) \) is edge-disjoint to \( \phi(H_2) \). That is, there are not edges \( f_1, f_2 \) of \( H_1, H_2 \) respectively with

\[
\{ \phi_1(v) \mid v \in f_1 \} = \{ \phi_2(v) \mid v \in f_2 \}
\]

\(^4\)In [20], a similar result is shown using the bound \( \Delta \leq 0.026n \). In fact, these two results are essentially the same. The discrepancy is that [20] assumes implicitly that the cells \( A(i, j) \) and \( A(j, i) \) have the same color, and these two cells only count for a single occurrence of that color. The reader should be careful that many sources on rainbow hamiltonian cycles use the latter convention.
Proposition 4.14. Suppose $H_i$ has $m_i$ edges and each edge intersects with at most $d_i$ other edges of $H_i$, for $i = 1, 2$. Also suppose that $(d_1 + 1)m_2 + (d_2 + 1)m_1 < \binom{n}{r}/e$.
Then there is an algorithm to find injective maps $\phi_1, \phi_2$ such that $\phi(H_1)$ is edge-disjoint to $\phi(H_2)$ in $O(e^{-1} \log^4 n)$ time whp.

Proof. We briefly review a construction of [26] applying the permutation LLL to this setting. Assume without loss of generality that $V(H_1) = [n]$ and $V(H_2) = n$ and that $\phi_1$ is the identity permutation. Thus, we need to select the bijection $\phi_2 : H_2 \rightarrow [n]$. We use the Swapping Algorithm to construct this permutation $\phi_2$.

Given a pair of edges $f_1 = \{u_1, \ldots, u_r\} \in E(H_1), f_2 = \{v_1, \ldots, v_r\} \in E(H_2)$, form $r!$ separate bad-events: for each permutation $\sigma \in S_r$ we have the bad-event 

$$\phi_2(v_1) = u_{\sigma 1} \land \cdots \land \phi_2(v_r) = u_{\sigma r}$$

The stated hypothesis ensures that the LLL criterion is satisfied for this collection of bad-events.

The number of bad-events is $m_1 m_2 r!$, which could potentially be exponentially large. However, there is a simple Bad-Event Checker here which can be implemented in $O(\log n)$ time; namely, for each pair of edges $f_1, f_2$, we sort $f_1$ and $\phi(f_2)$ and see if they are the same set.

Note that [19] only gives an RNC algorithm if $r = \text{polylog}(n)$ while Proposition [4.14] makes no restriction on the size of $r$.

References
[1] Achlioptas, D., Iliopoulos, F.: Random walks that find perfect objects and the Lovász Local Lemma. Journal of the ACM 63-3, Article #22 (2016)
[2] Albert, M., Frieze, A., Reed, B.: Multicoloured Hamilton Cycles. The Electronic Journal of Combinatorics 2-1, R10 (1995)
[3] 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)
[4] Blelloch, G., Fineman, J., Shun, J.: Greedy sequential maximal independent set and matching are parallel on average. Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 308-317 (2012)
[5] Brandt, S., Fischer, O., Hirvonen, J., Keller, B., Lempääinen, T., Rybicki, J., Suomela, J., Uitto, J.: A lower bound for the distributed Lovász Local Lemma. Proceedings of the 48th ACM Symposium on Theory of Computing (STOC), pp. 479-488 (2015)
[6] Chandrasekaran, K., Goyal, N., Haeupler, B.: Deterministic algorithms for the Lovász local lemma. SIAM Journal on Computing 42-6, pp. 2132-2155 (2013)
[7] Chung, K., Pettie, S., Su, H.: Distributed algorithms for the Lovász local lemma and graph coloring. Proceedings of the 2014 ACM Symposium on Principles of Distributed Computation (PODC), pp. 134-143 (2014)
[8] Cook, S.: A Taxonomy of problems with fast parallel algorithms. Information and Control 64, pp. 2-22 (1985)
[9] 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)

[10] Erdős, P., Spencer, J.: Lopsided Lovász Local Lemma and Latin transversals. Discrete Applied Math 30, pp. 151-154 (1990)

[11] Fischer, M., Noever, A.: Tight analysis of randomized greedy MIS. arxiv: 1707.05124. (2017)

[12] Gebauer, H., Szabó, T., Tardos, G.: The local lemma is asymptotically tight for SAT. Journal of the ACM 63-5, Article #43 (2016)

[13] Ghaffari, M.: An improved distributed algorithm for maximal independent set. Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 270-2777 (2016)

[14] Fischer, M., Ghaffari, M.: Sublogarithmic distributed algorithms for Lovász Local lemma, and the complexity hierarchy. arXiv:1705.04840 (2017)

[15] Haeupler, B., Harris, D.: Parallel algorithms and concentration bounds for the Lovász Local Lemma via witness DAGs. Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1170 - 1187 (2017)

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

[17] Harris, D.: Lopsidedependency in the Moser-Tardos framework: beyond the Lopsided Lovász Local Lemma. ACM Transactions on Algorithms 13-1, Article #17 (2016)

[18] Harris, D., Srinivasan, A.: Algorithmic and enumerative aspects of the Moser-Tardos distribution. ACM Transactions on Algorithms 13-3, Article # 33 (2017)

[19] Harris, D., Srinivasan, A.: A constructive algorithm for the Lovász Local Lemma on permutations. To appear in Theory of Computing.

[20] Harvey, N., Liaw, C.: Rainbow Hamilton cycles and lopsidedependency. Discrete Mathematics 340, pp. 1261-1270 (2017)

[21] Harvey, N., Vondrák, J.: An algorithmic proof of the Lopsided Lovász Local Lemma via resampling oracles. Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 1327-1346 (2015)

[22] Haxell, P.: An improved bound for the strong chromatic number. Journal of Graph Theory 58-2, pp. 148-158 (2008)

[23] Iliopoulos, F.: The algorithmic LLL and the witness tree lemma. arXiv:1704.02796 (2017)

[24] Kolipaka, K., Szegedy, M.: Moser and Tardos meet Lovász. Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC), pp. 235-244 (2011)

[25] Kolmogorov, V.: Commutativity in the algorithmic Lovász Local Lemma. Proceedings of the 57th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 780-787 (2016)

[26] Lu, L., Székely, L.: Using Lovász Local Lemma in the space of random injections. The Electronic Journal of Combinatorics 13-R63 (2007)
A Background on the LLL

The simplest criterion for the LLL or the LLLL is the symmetric LLL criterion $ep(d + 1) \leq 1$. This depends on only two properties of the collection $\mathcal{B}$: namely, the dependency structure among the bad-events, and their probabilities. There are a number of other criteria which can be stated in these terms, and which are frequently used in combinatorial constructions, including the asymmetric LLL criterion and the cluster-expansion criterion. The strongest criterion that can be stated in these terms is the Shearer criterion, due to [32].

We will describe the Shearer criterion in terms of stable-set sequences, which is more useful for analyzing the MT algorithms. The connection between stable-set sequences and the original form of Shearer’s criterion was developed by Kolipaka & Szegedy [24].

Define the dependency graph to be a graph $G$ whose vertex set is $\mathcal{B}$, and in which there is an undirected edge $(B, B')$ if $B \sim B'$. For any $B \in \mathcal{B}$, we define the inclusive neighborhood of $B$ as

$$N(B) = \{B\} \cup \{A \in \mathcal{B} \mid A \sim B\} = \{A \in \mathcal{B} \mid A \approx B\}$$

Likewise, we say that a set $J \subseteq \mathcal{B}$ is independent if it is an independent set of $G$; that is, there are not $B, B' \in J$ with $B \sim B'$.

We define a stable-set sequence to be a sequence $S = \langle S_0, S_1, S_2, \ldots, S_N \rangle$, where each $S_i \subseteq \mathcal{B}$ is non-empty and independent, and such that $S_i \subseteq \bigcup_{B \in S_{i-1}} N(B)$ for $i \geq 2$.

We say that the $S$ is rooted at $S_0$, and is singleton if $|S_0| = 1$; we sometime abuse notation and say that $S$ is rooted at $B$ if it is rooted in the singleton set $\{B\}$. We define the depth of $S$ to be $N$, the size of $S$ to be $\sum_{i=0}^{N} |S_i|$ and the weight of $S$ to be $w(S) = \prod_{i=0}^{N} \prod_{B \in S_i} P_{\Omega}(B)$. We define $\Gamma$ to be the set of all singleton stable-set sequences.

For many parallel algorithms, we need a slightly stronger criterion which we refer to as $\epsilon$-slack; this means that the vector of probabilities $(1 + \epsilon)P_\Omega(B)$ satisfies Shearer’s criterion. A related notion (which can also be defined for witness-trees) is the adjusted weight in terms of a weighting factor $\rho$, which is defined by $a_\rho(S) = w(S)(1 + \rho)^{|S|}$.

**Theorem A.1** ([24]). If Shearer’s criterion is satisfied, then $\sum_{S \in \Gamma} w(S) < \infty$. If Shearer’s criterion is satisfied with $\epsilon$-multiplicative slack, then $\sum_{S \in \Gamma} a_\epsilon(S) < \infty$.

Shearer’s criterion is difficult to work with in practice, so there are a number of simpler LLL criteria which can be related to Shearer’s criterion.
Theorem A.2. 1. (The asymmetric LLL criterion) Suppose some function \( x : \mathcal{B} \rightarrow [0, 1] \) has the property that for all \( B \in \mathcal{B} \)

\[
P_{\Omega}(B)(1 + \epsilon) \leq x(B) \prod_{A \sim B, A \neq B} (1 - x(A))
\]

Then Shearer’s criterion is satisfied with \( \epsilon \)-multiplicative slack.

2. (The cluster-expansion LLL criterion) Suppose that some function \( \mu : \mathcal{B} \rightarrow [0, \infty) \) has the property that for all \( B \in \mathcal{B} \)

\[
\mu(B) \geq P_{\Omega}(B)(1 + \epsilon) \prod_{I \subseteq N(B) \text{ independent}} \prod_{A \in B} \mu(A)
\]

Then Shearer’s criterion is satisfied with \( \epsilon \)-multiplicative slack.

Haeupler & Harris [15] showed a few useful results counting stable-set sequences when the dependency graph was defined in terms of the variable-assignment LLL. Although they used slightly different terminology, we can reformulate their results as follows:

Theorem A.3 ([15]). Suppose that the dependency graph \( G \) can be covered by \( b \) cliques. We have the following bounds on the weights of the stable-set sequences:

1. We have \( \sum_{S \in \Gamma} w(S) \leq O(b/\epsilon) \).

2. For \( t \geq 1 + 1/\epsilon \), we have \( \sum_{S \in \Gamma, |S| \geq t} w(S) \leq eb^t(1 + \epsilon)^{-t} \)

(We note that [15] analyzed the Shearer criterion only for the variable-assignment LLL setting. However, the only property of that setting that they used for this was the observation that, in the variable-assignment LLL setting with \( n \) variables, the dependency graph can be covered by \( n \) cliques.)

A.1 Commutative resampling oracles

In [23], Iliopoulous showed how bounds on stable-set sequences gave corresponding bounds on the probabilities of bad-events becoming true during the execution of the resampling algorithm. These bounds apply to any LLL setting with an object \( \mathcal{R} \) known as a commutative resampling oracle (using the terminology of [21, 25]). In such cases, we can use the following generic LLL algorithm:

Algorithm A general resampling algorithm
1: Draw \( \bar{X} \sim \Omega \) uniformly.
2: while some bad-event \( B \) is true on \( \pi \) do
3: Select, according to some arbitrary rule, some \( B \) true on \( X \)
4: Update \( \pi \leftarrow \mathcal{R}_B(X) \)

For example, we use precisely this framework for Algorithm 7. Please see [21] and [25] for much more technical details on such resampling oracles.

Suppose that when we run this algorithm we resample bad-events \( B_1, \ldots, B_T \) in sequence, and that a bad-event \( B \) is true after time \( T \). We may construct a corresponding singleton stable-set sequence rooted in \( B \); we refer to such a sequence at \( \hat{S}(B, T) \). Given any stable-set sequence \( S \) rooted at \( B \), we say that \( G \) appears if \( \hat{S}(B, T) = S \) for any value of \( T \).

A slight reformulation of results of Iliopoulous give the following result:
Proposition A.4. For a commutative resampling oracle, any stable-set sequence $S$ satisfies
\[ P(S \text{ appears}) \leq w(S) \]

Combining Theorem A.3 and Proposition A.4 gives the following result.

Proposition A.5. Suppose that $G$ satisfies the Shearer criterion with $\epsilon$-slack and the dependency graph can be covered in $b$ cliques. Then
\[ \sum_{S \in \Gamma} P(S \text{ appears}) \leq O(b/\epsilon) \]

If $t \geq 1 + 1/\epsilon$, then
\[ \sum_{S \in \Gamma, |S| \geq t} P(S \text{ appears}) \leq ebt(1 + \epsilon)^{-t} \]

B Some omitted proofs for Section 2

We use the following notation for directed graphs. For any vertex $v$, we refer to an edge $(v, w)$ (respectively $(w, v)$) as an an out-edge (respectively in-edge) of $v$. The set of vertices $\{w \mid (v, w) \in E\}$ are the out-neighbors of $v$, and the out-degree of $v$ is the cardinality of this set. Similarly the set of vertices $\{w \mid (w, v) \in E\}$ are the in-neighbors of $v$, and the in-degree of $v$ is the cardinality of this set.

Let $G = (V, E)$ be the input directed graph and let $\pi$ be a permutation mapping $V$ to $[n]$. We define the graph $G'$ with vertex set $V$ and edge set $E' = \{(u, v) \mid (u, v) \in E, \pi(u) < \pi(v)\}$.

For any integers $1 \leq j \leq n$, define $V_{[0,j]}$ to be vertex set
\[ V_{[0,j]} = \{v \mid \pi(v) \leq j\} \]
and we define $I_j = \{v \in I \mid \pi(v) \leq j\} = I \cap V_{[0,j]}$ where $I$ is the LFMIS of $G$ with respect to $\pi$.

For any integers $0 \leq i \leq j \leq n$, we define the vertex set $V_{(i,j)}$ as
\[ V_{(i,j)} = V_{[0,j]} - I_i - \text{out neighbors of } I_i \]

As shown in [4], instead of analyzing the greedy algorithm directly, we consider an alternative process. Given non-negative integers $n_0, n_1, \ldots, n_k$ with $1 = n_0 \leq n_1 \leq n_2 \leq \cdots \leq n_k = n$. We now consider the following process, which we call SLOW-GREEDY:

Algorithm 11 The SLOW-GREEDY algorithm
1: Initialize $I \leftarrow \emptyset$.
2: for $i = 1, \ldots, k$ do
3: Form $H_i$ as the induced subgraph $H_i = G'[V_{[n_{i-1}, n_i]}]$.
4: Using the greedy algorithm, find the LFMIS $J_i$ of $H_i$ with respect to the ordering $\pi$
5: Add $J_i$ to $I$.

As shown in [4], this produces the LFMIS of $G$ with respect to $\pi$ and its running time is at most that of the greedy algorithm. Thus, it suffices to bound the run-time of SLOW-GREEDY. More specifically, we will set $k = O(\log n)$ and select integers $n_0 \leq n_1 \leq \cdots \leq n_k$ such that the greedy algorithm on each $H_i$ takes time $O(\log n)$. To do this, we will show that every directed path in $H_i$ has length $O(\log n)$. 
**Proposition B.1.** The following property holds with probability at least $1 - n^{-100}$: for any integers $1 \leq i \leq j \leq n$ and any $v \in V$, the in-degree of $v$ in $G[V_{(i,j)}]$ is at most $O(n \log n)$.

**Proof.** Consider the sequential process to generate $I$, in which we successively select a random vertex $w$, add it to $I$, and remove $w$ and its out-neighbors from $G$.

1: Initialize $I \leftarrow \emptyset$.
2: Mark all the vertices of $G$ as alive.
3: for $k = 1, \ldots, n$ do
   4: Select $\pi(k)$ uniformly at random from $V - \{\pi(1), \ldots, \pi(k-1)\}$.
   5: if vertex $\pi(k)$ is alive then
      6: Add $\pi(k)$ to $I$
      7: Mark $\pi(k)$ and its out-neighbors as dead

Let us fix some vertex $v$ and a parameter $d$. For each $k = 0, \ldots, n$ define $E_k$ to be the event that $v$ has at least $d$ live in-neighbors after step $k$, and none of the in-neighbors of $v$ is in $I_k$.

We compute the probability of $E_k$ conditional on $E_1, \ldots, E_{k-1}$. As $E_1, \ldots, E_{k-1}$ are determined by $\pi(1), \ldots, \pi(k-1)$, it suffices to compute the probability of $E_k$ conditional on $\pi(1), \ldots, \pi(k-1)$; this allows us to compute the set of live in-neighbors after step $k-1$. If this number is less than $d$, then $E_k$ holds with probability zero. Otherwise, there is a probability of at least $\frac{d}{n-k+1}$ that $\pi(k)$ is an in-neighbor of $v$, in which case $E_k$ fails. Thus,

$$P(E_k \mid E_1, \ldots, E_{k-1}) \leq (1 - \frac{d}{n-k+1})$$

This implies that

$$P(E_k) \leq (1 - \frac{d}{n})(1 - \frac{d}{n-1}) \cdots (1 - \frac{d}{n-i+1}) \leq (1 - \frac{d}{n})^i \leq e^{-di/n}$$

This is at most $O(n^{-200})$ for $d = \Omega(n \log n)$. Taking a union bound over all $n^3$ possibilities for $v, i, j$, we see that the overall probability it fails to hold is at most $n^{-197}$ as desired. We finally note that $E_{i-1}$ is a necessary condition for the in-degree of $v$ in $G[V_{(i,j)}]$ to exceed $d$. 

**Proposition B.2.** Suppose that $1 \leq i \leq j \leq n$. Then the following holds whp: all paths in $G'[V_{(i,j)}]$ have length $O((j/i) \log n)$.

**Proof.** We first condition on $\pi(1), \ldots, \pi(i)$. At this point, $I_i$ is fixed. Consider $H = G[V_{(i,n)}]$; this is the graph induced on the vertices which are are not in $I_i$ or out-neighbors of $I_i$. Clearly $G[V_{(i,j)}]$ is a subgraph of $H$. Also, given the fixed value for $\pi(1), \ldots, \pi(i)$, we can determine $H$ — this does not depend on the precise identity of $\pi(i+1), \ldots, \pi(n)$.

Now, (barring events with low probability), the graph $H$ has maximum in-degree $\frac{cn \log n}{i}$, for some constant $c \geq 0$. Thus, the number of length $k$-paths in $H$ is at most $n \times (\frac{cn \log n}{i})^{k-1}$. A necessary condition for a path $P = v_1, \ldots, v_k$ to survive in $G[V_{(i,j)}]$ is that $\pi(v_1) < \pi(v_2) < \cdots < \pi(v_k) \leq j$. Having conditioned on $\pi(1), \ldots, \pi(i)$, this event has probability

$$\frac{i-i}{n-i} \times \frac{i-i-1}{n-i-1} \times \cdots \times \frac{j-i-k+1}{n-i-k+1} \leq (j/n)^k / k!$$

Taking a union-bound over all such paths $P$,

$$P(G^\pi[V_{(i,j)}]) \text{ has some length-} k \text{ path} \leq n \left(\frac{cn \log n}{i}\right)^{k-1} (j/n)^k / k! \leq n(c(j/i) \log n)^k / k!$$

For $k \geq \Omega(j/i \log n)$, this is $n^{-\Omega(1)}$. 


Theorem B.3. Suppose that every vertex of $G$ has in-degree at most $d$. Let $x = d/\log n$.

If $\pi$ is chosen uniformly at random from $S_n$, the algorithm GREEDY-FIND-LFMIS algorithm takes $O(\log^{(1+x)} n\log n) \leq O(\log d\log n) \leq O(\log^2 n)$ rounds whp.

Proof. First consider the case when $d \leq \frac{\log n}{2}$. In this case, observe that there are at most $nd^k$ paths in $G[V]$ of length $k$. Each such path survives to $G'$ with probability $1/k!$; thus, the expected number of length-$k$ paths in $G'$ is at most $nd^k/k!$. This is negligible for $k = \Omega(\frac{\log n}{\log(\log n)})$.

Next, suppose $d \geq \frac{1}{2} \log n$. Consider the algorithm SLOW-GREEDY, with $n_1 = \frac{n\log n}{d}$ and $n_j = \min(n, n2^{j-1})$ for $j = 2, \ldots, k = \lceil \log_2(\frac{d}{\log n}) \rceil$.

There are at most $nd^k$ length-$k$ paths in $G[V]$; each survives to $G'$ with probability $1/k!$ and survives to $H_1 = G'[V_{(0,n_1)}]$ with probability $(n_1/n)^{k+1}$. Hence the probability that $H_1$ contains a $k$-long path is at most $n(n\log n)^k/k!$. This is negligible for $k \geq \Omega(\log n)$.

By Proposition B.2 for $i > 1$ each graph $H_i$ has maximum path length $O(\frac{n}{n_i-1} \log n) = O(\log n)$.

Thus, overall, the SLOW-GREEDY algorithm takes $O(1 + \log_2(\frac{d}{\log n})) \times O(\log n) \leq O(\log(x + 1)(\log n))$ rounds whp. \qed

C Probability spaces amenable to the permutation LLL

In this section we give some examples of probability spaces which satisfy the conditions (C1)—(C6) needed for efficient parallel resampling oracles.

It is often convenient to define function $F_u : \langle u \rangle \times R(u) \to W$ by $F_u(\pi, r) = r\pi$. Note that property (C3) is equivalent to requiring that $F_u$ is bijective for all $u \in U$.

C.1 The full set $S_n$

Here, we have $W = S_n$. We define the relation $\sim$ as $(x, y) \sim (x', y')$ iff $x = x'$ or $y = y'$. For $u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \subseteq Z$ we define

$$T(u) = \{(x_1 z_1)(x_2 z_2)\ldots(x_k z_k) \mid z_i \in [n] - \{x_{i+1}, \ldots, x_k\}\}$$

We define $G = S_n$. We define the group action for each $\sigma \in G$ and $\pi \in W$ by setting,

$$\sigma \pi := \pi \sigma$$

The definition of $T(u)$ superficially appears to depend on the ordering of the pairs $(x_1, y_1), \ldots, (x_k, y_k)$. The following result shows that this is not the case.

Proposition C.1. $T(u)$ is well-defined, regardless of the ordering of $u$.

Proof. It suffices to show that we can swap the ordering of any adjacent entries $(x_j, y_j), (x_{j+1}, y_{j+1})$. In order to show this, consider $\sigma = (x_1 z_1)\ldots(x_k z_k)$ where $z_i \in [n] - \{x_{i+1}, \ldots, x_k\}$. We need to find an appropriate choice of $z'_1, \ldots, z'_k$ such that

$$\sigma = (x_1 z'_1)\ldots(x_j z'_{j-1})(x_{j+1} z'_{j+1})(x_j z'_j)\ldots(x_k z'_k)$$

We will take $z'_i = z_i$ for $i \neq j, j + 1$. Thus, in order to achieve (1), it suffices to find $z'_{j+1} \in [n] - \{x_j, x_{j+2}, \ldots, x_k\}$, $z'_j \in [n] - \{x_{j+1}, \ldots, x_k\}$ such that

$$(x_j z_j)(x_{j+1} z_{j+1}) = (x_{j+1} z'_{j+1})(x_j z'_j)$$

There are few cases. Note that we may assume $z_j \neq x_{j+1}$.
1. Suppose that $z_j \neq x_{j+1}$ and $z_{j+1} \neq x_j$ and $z_j \neq z_{j+1}$. Then the terms $(x_j z_j), (x_{j+1} z_{j+1})$ commute, and so

$$ (x_j z_j)(x_{j+1} z_{j+1}) = (x_{j+1} z_{j+1})(x_j z_j) $$

with $z_{j+1}' = z_{j+1}, z_j' = z_j$. Our hypothesis $z_j \neq x_{j+1}$ ensures that $z_{j+1}' \neq x_j$.

2. Suppose that $z_j \neq x_{j+1}$ and $z_{j+1} \neq x_j$ and $z_j = z_{j+1}$. Let $z = z_j = z_{j+1}$. Then

$$ (x_j z_j)(x_{j+1} z_{j+1}) = (x_j z)(x_{j+1} z) = (x_{j+1} x_j)(x_j z) $$

So setting $z_{j+1}' = x_j, z_j' = z$ works. Our hypothesis $z \neq x_j$ ensures that $z_{j+1}' \neq x_j$.

\[ \square \]

**Proposition C.2.** For any $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$, we have $|T(u)| = \frac{n!}{(n-k)!}$.

Proof. Clearly $|T(u)| \leq \frac{n!}{(n-k)!}$. In order to show equality, we need to show that every choice of $z_1, \ldots, z_k$ gives rise to a unique element of $T(u)$.

Suppose that $\sigma = (x_1 z_1) \ldots (x_k z_k)$ with $z_i \in [n] - \{x_i+1, \ldots, x_k\}$. Observe that $\sigma z_k = (x_1 z_1) \ldots (x_{k-1} z_{k-1}) x_k$; this is equal to $x_k$ as $z_1, \ldots, z_{k-1}$ are not equal to $x_k$. So $z_k = \sigma^{-1} x_k$. Thus we can write $\sigma(x_k z_k) = (x_1 z_1) \ldots (x_{k-1} z_{k-1})$. Continuing in this way we can determine $z_1, \ldots, z_k$.

We now verify properties (C1) — (C6). Property (C1) and (C2) are clear.

**Proposition C.3.** Property (C3) holds.

Proof. Consider $u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \in U$. By counting it suffices to show that the function $F_u$ is injective. That is, if we are given $\tau = \pi \sigma$ for $\pi \in \langle u \rangle$ and $\sigma \in T(u)$, then we must show that $\pi, \sigma$ are uniquely determined from $\tau$.

Suppose that $\sigma = (x_1 z_1) \ldots (x_k z_k)$ where $z_i \in [n] - \{x_{i+1}, \ldots, x_k\}$. Then

$$ \tau z_k = \pi(x_1 z_1) \ldots (x_{k-1} z_{k-1}) x_k = \pi x_k = y_k. $$

So $z_k = \tau^{-1} y_k$. In a similar fashion to Proposition C.2 we peel off $z_k, \ldots, z_1$ and finally $\pi$.

\[ \square \]

**Proposition C.4.** Property (C4) holds.

Proof. Let $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$, and let $v = (x, y)$ where $\pi x \neq y$. Let $z_1, \ldots, z_k$ be chosen such that $z_i \in [n] - \{x_{i+1}, \ldots, x_k\}$.

Suppose for contradiction that $\pi(x_1 z_1) \ldots (x_k z_k) x = y$. Let $i \leq k$ be minimal such that $\pi(x_1 z_1) \ldots (x_i z_i) x = y$ and let $\sigma = \pi(x_1 z_1) \ldots (x_{i-1} z_{i-1})$. It must be the case that $z_i = x$ or $x_i = x$. The latter cannot occur as $(x_i, y_i) \not\in (x, y)$.

So $z_i = x$ and hence $\sigma(x_i x) = y$. This implies that $\sigma x_i = y$. So $\pi(x_1 z_1) \ldots (x_{i-1} z_{i-1}) x_i = y$. Since $z_1, \ldots, z_{i-1}$ are distinct from $x_i$, this implies that $\pi x_i = y$. But $\pi x_i = y_i \neq y$.

\[ \square \]

**Proposition C.5.** Suppose $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ and let $v = (a, b)$ where $u \not\in \langle u \cup v \rangle$ and let $\sigma = (x_1 z_1) \ldots (x_k z_k)$ where $z_i \in [n] - \{x_{i+1}, \ldots, x_k\}$. Then $\pi \sigma \in \langle v \rangle$ iff $z_1, \ldots, z_k$ are all distinct from $a$.

Proof. The reverse direction is clear: if $a$ is distinct from $z_1, \ldots, z_k$ then $\pi \sigma a = \pi a = b$.

To show the forward direction, suppose that $i \leq k$ is minimal such that $z_i = a$. Let $\tau = \pi(x_1 z_1) \ldots (x_{i-1} z_{i-1}) x_i = \pi x_i = y_i \neq b$. Since $x_{i+1} z_{i+1} \ldots (x_k z_k) \in Z((x_{i+1}, y_{i+1}), \ldots, (x_k, y_k))$ and $\pi \sigma = \tau (x_{i+1} y_{i+1}) \ldots (x_k z_k)$, then by Proposition C.4 this implies $\pi a \neq b$.

\[ \square \]
Corollary C.6. Property (C6) holds.

Proof. By Proposition C.5, we have that $\pi \sigma \in \langle (a, b) \rangle$ iff $z_1, \ldots, z_k$ are distinct from $a$. This criterion does not depend on the precise value of $\pi$, so it either holds for all $\pi \in \langle u \cup v \rangle$ or none of them. □

Proposition C.7. Property (C5) holds.

Proof. Let us write $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ and $v = \{(a_1, b_1), \ldots, (a_{\ell}, b_{\ell})\}$. With our definition of $R$, we need to show that

$$T(u; v) T(v) = T(u \cup v)$$

where $T(u; v)$ is the set of all permutations $\sigma \in T(u)$ such that $\langle u \cup v \rangle \subseteq \langle v \rangle$.

To show the LHS contains in the RHS, let $\pi \in \langle u \cup v \rangle$, let $\sigma = (x_1 z_1) \ldots (x_k z_k)$ where $z_i \in [n] - \{x_{i+1}, \ldots, x_k\}$ and let $\rho = (a_1 c_1) \ldots (a_{\ell} c_{\ell})$ where $c_i \in [n] - \{a_{i+1}, \ldots, a_{\ell}\}$. By Proposition C.7, $z_1, \ldots, z_k$ are all distinct from $a_1, \ldots, a_{\ell}$. It follows that $\tau = (x_1 z_1) \ldots (x_k z_k)(a_1 c_1) \ldots (a_{\ell} c_{\ell})$ is an element of $T(u \cup v)$, as desired.

To show the RHS contains in the LHS, let $\tau \in T(u \cup v)$. By Proposition C.1, we have $\tau = (x_1 z_1) \ldots (x_k z_k)(a_1 c_1) \ldots (a_{\ell} c_{\ell})$ where $z_i \in [n] - \{x_{i+1}, \ldots, x_k\}$ and $c_i \in [n] - \{a_{i+1}, \ldots, a_{\ell}\}$. Then let $\sigma = (x_1 z_1) \ldots (x_k z_k)$ and $\rho = (a_1 c_1) \ldots (a_{\ell} c_{\ell})$. We clearly have $\sigma \in T(u)$ and $\rho \in T(v)$. Also, $z_1, \ldots, z_k$ are distinct from $c_1, \ldots, c_{\ell}$ so by Proposition C.6, we have $\pi \sigma \in \langle v \rangle$. □

C.2 Matchings of $K_n$

Throughout this section, we suppose that $n$ is an even integer. The set of matchings on $K_n$ is probability space for the LLLL, yet it is not obviously associated with any permutation group. In order to bring it into our permutation-based framework, we define $W \subseteq S_n$ to be the set of permutations $\pi$ whose cycle structure consists of $n/2$ disjoint two-cycles; that is, $\pi$ has the form

$$\pi = (a_1 a_2)(a_3 a_4) \ldots (a_{n-1} a_n)$$

where $a_1, \ldots, a_n$ are all distinct. We refer to such permutations as full involutions. There is an obvious correspondence between full involutions and matchings of $K_n$.

We define the relation $\sim$ as $(x, y) \sim (x', y')$ iff any of the four values $x, y, x', y'$ are equal to each other.

In order to define $R(u)$, we first define $T(u)$ for $u$ an ordered list of pairs $(x_1, y_1), \ldots, (x_k, y_k)$; namely, we define

$$T(u) = \{ (x_1 z_1)(x_2 z_2) \ldots (x_k z_k) \mid z_i \in [n] - \{x_{i+1}, \ldots, x_k\} - \{y_i, \ldots, y_k\} \}$$

Thus, for example, if $u = ((1, 2), (3, 4))$ then $T(u)$ is the set of permutations of the form $(1 z_1)(3 z_3)$, where $z_3 \in [n] - \{4\}$ and $z_1 \in [n] - \{2, 3, 4\}$. Note that this definition depends in a fundamental way on the ordering of $x_i$.

We define $G = S_n$. We define the group action, for each $\sigma \in G$ and $\pi \in W$, by setting

$$\sigma \pi := \sigma^{-1} \pi \sigma$$

For $u \subseteq Z$, we define $R(u)$ as follows: enumerate $u$ in sorted order as $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ where $x_1 < x_2 < \cdots < x_k$, and set $R(u) = Z((x_1, y_1), \ldots, (x_k, y_k))$. In other words, given some bad-event $B$, the resampling oracle $R_B(\pi)$ is conjugation of $\pi$ by a random element of $Z(B)$.

Although the definition of $T(u)$ and $R(u)$ depends on the ordering of $x_1, \ldots, x_k$, the resulting group action will not. We show this in Proposition C.9.
Proposition C.8. Suppose $x_1, y_1, x_2, y_2$ are distinct values and $\rho$ is a full-involution with $\rho x_1 = y_1, \rho x_2 = y_2$. Then for any $z_1 \in [n] - \{y_1, x_2, y_2\}, z_2 \in [n] - \{y_2\}$ there are $w_2 \in [n] - \{x_1, y_1, y_2\}, w_1 \in [n] - \{y_1\}$ such that

$$(x_2 z_2)(x_1 z_1)\rho(x_1 z_1)(x_2 z_2) = (x_1 w_1)(x_2 w_2)\rho(x_2 w_2)(x_1 w_1)$$

Proof. By relabeling, we may assume wlg that $x_1 = 1, y_1 = 3, x_2 = 2, y_2 = 4$ and that $z_1, z_2 \in [6]$. So $z_1 \in \{1, 5, 6\}$ and $z_2 \in \{1, 2, 3, 5, 6\}$ and we wish to show that there are $w_1, w_2$ such that

$$(2 z_2)(1 z_1)\rho(1 z_1)(2 z_2) = (1 w_1)(2 w_2)\rho(2 w_2)(1 w_1)$$

and $w_2 \in [n] - \{1, 3, 4\}, w_1 \in [n] - \{3\}$.

Let us first suppose that $\rho$ contains the transposition $(56)$. The observe that as long as we select $w_1, w_2 \in \{1, \ldots, 6\}$ then the LHS and RHS will agree on permutations entries $t > 6$. So, it will suffice to show that (2) holds where $\rho = (13)(24)(56)$ and $n = 6$.

Next let suppose that $\rho$ does not send 5 to 6. By relabeling, we may suppose wlg that $\rho$ contains the transpositions $(57)$ and $(68)$. Again, as long as we select $w_1, w_2 \in \{1, \ldots, 8\}$ then the LHS will agree on all permutations entries $t > 8$. So it will suffice to show that (2) holds when $\rho = (13)(24)(57)(68)$ and $n = 8$.

We have exhaustively tested all choices of $z_1, z_2, w_1, w_2$. In the majority of these cases, taking $w_1 = z_1, w_2 = z_2$ works. The following table shows how to select $w_1, w_2$ in the remaining cases (if a row is omitted, it means that $w_1 = z_1, w_2 = z_2$ is a satisfying solution.)

| \(\rho = (1\ 3)(2\ 4)(5\ 6)\) | \(z_1\) | \(z_2\) | \(w_1\) | \(w_2\) |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 3 | 4 | 2 |
| 5 | 1 | 5 | 5 |
| 5 | 3 | 4 | 6 |
| 5 | 5 | 2 | 5 |
| 6 | 1 | 6 | 6 |
| 6 | 3 | 4 | 5 |
| 6 | 6 | 2 | 6 |
| 1 | 1 | 2 | 2 |
| 1 | 3 | 4 | 2 |
| 5 | 1 | 5 | 5 |
| 5 | 3 | 4 | 7 |
| 6 | 1 | 6 | 6 |
| 6 | 3 | 4 | 8 |
| 6 | 6 | 2 | 6 |
| 7 | 1 | 7 | 7 |
| 7 | 3 | 4 | 5 |
| 8 | 1 | 8 | 8 |
| 8 | 3 | 4 | 6 |
| 8 | 8 | 2 | 8 |

Proposition C.9. Suppose that $\pi \in \langle u \rangle$. The following are equivalent.

(a) There is some ordering $(x_1, y_1), \ldots, (x_k, y_k)$ of $u$ and some $\sigma \in Z((x_1, y_1), \ldots, (x_k, y_k))$ such that $\tau = \sigma^{-1} \pi \sigma$.

(b) For every ordering $(x_1, y_1), \ldots, (x_k, y_k)$ of $u$ there is some $\sigma \in Z((x_1, y_1), \ldots, (x_k, y_k))$ such that $\tau = \sigma^{-1} \pi \sigma$.
Proof. We show this by induction on $k$. It suffices to swap adjacent elements of $u$. Suppose that 
$\tau = \sigma^{-1}\pi\sigma$ where $\sigma = (x_1 \ z_1) \ldots (x_k \ z_k)$ and we swap $j$ and $j+1$ entry. Let $(x'_i, y'_i)$ be the result after this swap; that is, $(x_i, y_i) = (x'_i, y'_i)$ if $i \neq j, i \neq j+1$ and $(x'_j, y'_j) = (x_{j+1}, y_{j+1})$ and $(x'_{j+1}, y'_{j+1}) = (x_j, y_j)$. We want to show that there are $z'_1, \ldots, z'_k$ such that

$$\tau = (x'_k \ z'_k) \ldots (x'_1 \ z'_1)\pi(x'_1 \ z'_1) \ldots (x'_k \ z'_k)$$

and such that $z'_i \in [n] - \{x'_{i+1}, \ldots, x'_k\} - \{y'_i, \ldots, y'_k\}$.

Let us first suppose that $j \leq k-2$. Let $\sigma' = (x_1 \ z_1) \ldots (x_{k-1} \ z_{k-1})$ and consider the permutation $\tau' = (\sigma'^{-1})\pi(\sigma')$. By relabeling, we may assume wlg that $y_k = n-1, x_k = n$; observe that $\tau'$ swaps $n-1$ and $n$. So the permutation $\tau'(n-1 \ n)$ can be regarded as a full involution on the set $[n-2]$, and for each $i < k$ we have $z_i \in [n-2] - \{x_{i+1}, \ldots, x_{k-1}, y_i, \ldots, y_k\}$. Furthermore, if we set $\pi' = \pi(n-1 \ n)$ we can see that $\pi'$ is a full involution on $[n-2]$ which satisfies $\pi' x_i = y_i$ for $i = 1, \ldots, k-1$.

We see that $\tau'(n-1 \ n) = (\sigma'^{-1})\pi'\sigma'$ where $\pi' \in \{(x_1, y_1), \ldots, (x_{k-1}, y_{k-1})\}$ and $\pi', \tau', \sigma'$ live in $S_{n-2}$. By induction hypothesis, we may swap the orderings of $(x_j, y_j)$ and $(x_{j+1}, y_{j+1})$. Thus, there exist $z'_1, \ldots, z'_{k-1} \in [n-2]$ such that

$$\tau(n-1 \ n) = (x'_{k-1} \ z'_{k-1}) \ldots (x'_1 \ z'_1)\pi(n-1 \ n)(x'_1 \ z'_1) \ldots (x'_{k-1} \ z'_{k-1})$$

As none of the $x', z'$ entries are equal to $n, n-1$ this implies that

$$\tau' = (x'_{k-1} \ z'_{k-1}) \ldots (x'_1 \ z'_1)\pi(x'_1 \ z'_1) \ldots (x'_{k-1} \ z'_{k-1})$$

which in turn implies that

$$\tau = (x_k \ z_k)(x_{k-1} \ z_{k-1}) \ldots (x'_1 \ z'_1) \ldots (x'_{k-1} \ z'_{k-1})(x_k \ z_k)$$

Setting $z'_k = z_k$, this satisfies [3] as $z'_1, \ldots, z'_{k-1}$ are all distinct from $x_k, y_k$.

Thus, in order to prove this result, it suffices to consider the case when $j \leq k-1$, that is, we

need to show that we can swap the orderings of $(x_{k-1}, y_{k-1})$ and $(x_k, y_k)$.

So suppose that $\tau = \sigma^{-1}\pi\sigma$ where $\sigma = (x_1 \ z_1) \ldots (x_k \ z_k)$. Now let

$$\rho = (x_{k-2} \ z_{k-2})(x_{k-3} \ z_{k-3}) \ldots (x_1 \ z_1)\pi(x_1 \ z_1) \ldots (x_{k-3} \ z_{k-3})(x_{k-2} \ z_{k-2})$$

Since $z_{k-2}, \ldots, z_1$ are all distinct from $x_k, y_k, x_{k-1}, y_{k-1}$ observe that $\rho x_{k-1} = y_k$ and $\rho x_{k-1} = y_{k-1}$. Also, $\rho$ is clearly a full involution since it is a conjugation of $\pi$. Hence by Proposition C.8 there exists $w_{k-1}, w_k$ such that

$$(x_{k-1} \ w_{k-1})(x_k \ w_k)\rho(x_k \ w_k)(x_{k-1} \ w_{k-1}) = (x_k \ z_k)(x_{k-1} \ z_{k-1})\rho(x_{k-1} \ z_{k-1})(x_k \ z_k)$$

and $w_{k-1} \in [n] - \{y_k\}, w_k \in [n] - \{x_{k-1}, y_{k-1}, y_k\}$. Now observe that if we set

$$\sigma' = (x_1 \ z_1) \ldots (x_{k-2} \ z_{k-2})(x_k \ w_k)(x_{k-1} \ w_{k-1})$$

then $\tau = (\sigma')^{-1}\pi\sigma'$ and $\sigma' \in Z((x_1, y_1), \ldots, (x_{k-2}, y_{k-2}), (x_k, y_k), (x_{k-1}, y_{k-1}))$ as desired. \hfill $\square$

In light of Proposition C.9 we will sometimes abuse notation and refer to $T(u)$ where $u$ is an unordered subset of $Z$. Formally, we would have to specify some canonical ordering on $T(u)$ in order for this to be well-defined; however, as long as we only use $T(u)$ in the context of conjugations, then the precise ordering of $u$ is irrelevant.

We now show that the necessary properties are satisfied. Property (C1) is clear. Since conjugation preserves cycle structure, it is clear that property (C2) holds.

30
Proposition C.10. Property (C3) holds.

Proof. Consider \( u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \subset U \). Note that \( R(u) \) has size at most \((n - 1)!!/(n - 1 - 2k)!!\) and \( |u| \) has size precisely \((n - 1 - 2k)!!\) and so \(|u| \times R(u)| \leq |W|\). In order to show that the function \( F_u \) is a bijection onto \( W \), it thus suffices to show that \( F_u \) is injective.

Suppose that \( \pi \in \langle u \rangle \) and \( \sigma = (x_k z_k) (x_1 z_1) \pi (x_1 z_1) \ldots (x_k z_k) \) for \((x_1 z_1) \ldots (x_k z_k) \in T(u)\).

We claim that \( z_k, \ldots, z_1 \) are uniquely determined. For, observe that \( \sigma z_k = y_k \); this follows since \( z_2, \ldots, z_1 \) are all distinct from \( x_k, y_k \) and \( z_k \neq y_k \).

So we can peel off \((x_k z_k)\) to determine \((x_k z_k) \sigma (x_k z_k) = (x_k z_k - 1) \pi (x_1 z_1) \ldots (x_k z_k - 1)\).

Continuing in this way, we can determine \( z_k, z_{k - 1}, \ldots, z_1 \) and finally \( \pi \) itself.

\( \square \)

Corollary C.11. For \( u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \subset U \), we have \(|R(u)| = (n - 1)!!/(n - 1 - 2k)!!\).

Proof. By Proposition C.10, \( F_u \) is a bijection between \( \langle u \rangle \times R(u) \) and \( W \). Note that \(|R(u)| = |Z((x_1, y_1), \ldots, (x_k, y_k))|\). Hence \(|R(u)| = |Z((x_1, y_1), \ldots, (x_k, y_k))| = \frac{|W|}{|u|} = \frac{(n-1)!!}{(n-1-2k)!!} \). \( \square \)

Proposition C.12. Property (C4) holds.

Proof. Let \( u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \) in sorted order and let \((x, y) \in v\) where \( \pi x \neq y \). Let \( z_1, \ldots, z_k \) be chosen such that \( z_i \in [n] - \{x_{i + 1}, \ldots, x_k, y_i, \ldots, y_k\} \) and let \( \sigma = (x_1 z_1) \ldots (x_k z_k) \).

Suppose for contradiction that \( \sigma^{-1} \pi x = y \). Let \( i \leq k \) be minimal such that

\[
(x_i z_i) \ldots (x_1 z_1) \pi (x_1 z_1) \ldots (x_i z_i)x = y.
\]

We must have \( i > 0 \) as \( \pi x \neq y \). It must be the case that either \( z_i \) or \( x_i \) is equal to \( x \) or \( y \). But it cannot be \( x_i \) since \( (x_i, y_i) \neq (x, y) \). So suppose \( z_i = x \). Then observe that

\[
(x_i z_i) \ldots (x_1 z_1) \pi (x_1 z_1) \ldots (x_i z_i)x = (x_i x) (x_i x_{i-1} z_{i-1}) \ldots (x_1 z_1) \pi (x_1 z_1) \ldots (x_i z_i x_i x)\]

\[
= (x_i x) y_i = y_i \neq y
\]

A similar argument holds if \( z_i = y \). \( \square \)

Proposition C.13. Suppose \( u = \{(x_1, y_1), \ldots, (x_k y_k)\} \) and let \( v = (a, b) \) where \( u \not\sim v \). Let \( \pi \in \langle u \cup v \rangle \) and let \( \sigma \in T(u) \) have the form \( \sigma = (x_1 z_1) \ldots (x_k z_k) \) where \( z_i \in [n] - \{x_{i+1}, \ldots, x_k, y_i, \ldots, y_k\} \). Then \( \sigma^{-1} \pi \sigma \in \langle v \rangle \) iff \( z_1, \ldots, z_k \) are all distinct from \( a, b \).

Proof. The reverse direction is clear: if \( a, b \) are distinct from \( z_1, \ldots, z_k \) then \( \sigma^{-1} \pi \sigma = \sigma^{-1} \pi a \).

To show the forward direction, suppose that \( \sigma^{-1} \pi a = \sigma^{-1} \pi b \). Let \( \tau = (x_1 z_1) \ldots (x_i z_i) \); for concreteness let us say that \( z_i = a \) (the other case is identical.)

One can see that \( \tau^{-1} \pi \sigma = (x_i a) (x_{i-1} z_{i-1}) \ldots (x_1 z_1) \pi (x_1 z_1) \ldots (x_{i-1} z_{i-1}) (x_i a) = y_i \neq b \) (we use here the fact that \( a, b \not\sim u \). Since \( (x_{i+1}, z_{i+1}) \ldots (x_k z_k) \in Z((x_{i+1}, y_{i+1}), \ldots, (x_k, y_k)) \) and \( \sigma^{-1} \pi \sigma = (x_k z_k) \ldots (x_{i+1} z_{i+1}) \pi (x_{i+1} y_{i+1}) \ldots (x_k z_k) \), then by Proposition C.12 we have \( \sigma^{-1} \pi \sigma \neq b \), a contradiction. \( \square \)

Corollary C.14. Property (C6) holds.

Proof. By Proposition C.13 we have that \( \sigma^{-1} \pi \sigma \in \langle (a, b) \rangle \) iff \( z_1, \ldots, z_k \) are distinct from \( a, b \). This criterion does not depend on the precise value of \( \pi \), so it either holds for all \( \pi \in \langle u \cup v \rangle \) or none of them.
Proposition C.15. Property (C5) holds.

Proof. Let us write $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ and $v = \{(a_1, b_1), \ldots, (a_\ell, b_\ell)\}$.

We first show the LHS is contained in the RHS. Let $\pi \in (u \cup v)$, let $\pi = (x_1 z_1) \ldots (x_k z_k)$ where $z_i \in [n] - \{x_{i+1}, \ldots, x_k, y_i, \ldots, y_k\}$ and let $\rho = (a_1 c_1) \ldots (a_\ell c_\ell)$ where $c_i \in [n] - \{a_{i+1}, \ldots, a_\ell, b_1, \ldots, b_\ell\}$.

By Proposition C.5 we must have $z_1, \ldots, z_k$ all distinct from $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell$. It follows that $\pi = (x_1 z_1) \ldots (x_k z_k)(a_1 c_1) \ldots (a_\ell c_\ell)$ is an element of $T(u \cup v)$, as desired.

We next show the RHS is contained in the LHS. Suppose $\pi' = \sigma^{-1} \pi \sigma$ for some $\sigma \in T(u \cup v)$.

By Proposition C.9 we have

$$\pi' = \pi^{-1} \rho^{-1} \pi \rho$$

where $z = (x_1 z_1) \ldots (x_k z_k)$, $\rho = (a_1 c_1) \ldots (a_\ell c_\ell)$, and $z_i \in [n] - \{x_{i+1}, \ldots, x_k, y_i, \ldots, y_k\}$ and $c_i \in [n] - \{a_{i+1}, \ldots, a_\ell, b_1, \ldots, b_\ell\}$.

By Proposition C.9 we have $\rho^{-1} \pi \rho \in R(u)\pi$. Also, since $z_1, \ldots, z_k$ are distinct from $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell$, by Proposition C.13 we have $\rho^{-1} \pi \rho \in R(u; v)$ as well. Furthermore, by Proposition C.9 we have $\pi^{-1} \rho^{-1} \pi \rho \in R(v)(\rho^{-1} \pi \rho)$. So $\pi' \in R(v)R(u; v)\pi$.

C.3 Hamiltonian cycles of $K_n$

The set of hamiltonian cycles on $K_n$ is a probability space for the LLLL not obviously associated with any permutation group. In order to bring it into our permutation-based framework, we define $W \subseteq S_n$ to be the set of full-cycle permutations, that is, $\pi$ of the form

$$\pi = (a_1 a_2 \ldots a_n)$$

where $a_1, \ldots, a_n$ are all distinct. There is an obvious correspondence between these full-cycle permutations and hamiltonian cycles of $K_n$.

We define the relation $\sim$ as $(x, y) \not\sim (x', y')$ iff $x, x', y, y'$ are all distinct. We define $G$ to be $S_n$, and we define the group action of $\sigma \in G$ on $\pi \in W$ by

$$\sigma \pi := \sigma \pi$$

Before defining $R(u)$, we define a permutation $\lambda^u$ which “normalizes” $(u)$. We begin by defining a directed graph $G^u$, on vertex set $u$, with an edge from $(x, y)$ to $(y, z)$. In order for $u$ to be valid, this graph must necessarily have in-degree at most one and out-degree at most one. So it is a union of cycles and paths.

Suppose that $G^u$ has a cycle $(a_1, a_2), (a_2, a_3), \ldots, (a_{k-1}, a_k), (a_k, a_1)$. In this case, any permutation $\pi \in (u)$ must satisfy $\pi(a_1) = a_2, \pi(a_2) = a_3, \ldots, \pi(a_k) = a_1$. In other words, one of the cycles in $\pi$ is $(a_1 a_2 \ldots a_k)$. Since $\pi$ contains a single cycle of length $n$, necessarily $k = n$ and in fact $(u)$ contains only a single permutation $\pi$; in this case we set $\lambda^u = \pi^{-1}$ and we say that $u$ is degenerate.

Putting aside this case, we define a path of $u$ to be a sequence $a_1, a_2, \ldots, a_k$ corresponding to a directed path $(a_1, a_2), (a_2, a_3), \ldots, (a_{k-1}, a_k)$ in $G^u$. For a path $p$, we define start$(p)$ and end$(p)$ to the first and last elements in the path (here, $a_1$ and $a_k$) respectively. We define $\lambda^p$ to be $(a_k a_{k-1} \ldots a_1)$.

We now define $\lambda^u$ to be the product of $\lambda^p$ over all such paths $p$. One can observe the following facts about $\lambda^u$: Consider $\pi \in (u)$, with a single cycle written as $(a_1 a_2 \ldots a_n)$. Let $a'$ be the length $n - k$ list obtained from $a$ by deleting every element of $q(u)$. Then

$$\lambda^u \pi = (a'_1 \ldots a'_k)$$
Furthermore, if \( u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \), then for any cycle \( \sigma \) of length \( n - k \) on set \([n] - \{x_1, \ldots, x_k\}\), the permutation \( \sigma(\lambda^u)^{-1} \) is an element of \((u)\).

Also, observe that if \( u \preceq v \), then the paths in the directed graphs corresponding to \( u, v \) respectively do not interact, and thus \( \lambda^u \lambda^v = \lambda^v \lambda^u = \lambda^{uv} \).

We may now define, for any \( u \in U \):

\[
T(u) = \left\{ (x_k z_k) \cdots (x_2 z_2)(x_1 z_1)\lambda^u \mid z_i \in [n] - \{x_i, \ldots, x_k\} \right\}
\]

**Proposition C.16.** Suppose \( u \cup v \in U \) and \( u \not\preceq v \). Then \( u, v, \) and \( u \cup v \) are not degenerate.

**Proof.** If \( u \) is degenerate, then it must size have \( n \). So \( v \) must overlap with some element of \( u \), which contradicts that \( u \not\preceq v \).

Suppose \( u \cup v \) is degenerate. Since \( u, v \) are not degenerate, let \( p \) be some path of \( u \). The endpoint of this path must be a startpoint of a path of \( v \), and so \( u \sim v \).

**Proposition C.17.** The definition of \( T(u) \) does not depend on the ordering of \((x_1, y_1), \ldots, (x_k, y_k)\) (only on the unordered set \( \{(x_1, y_1), \ldots, (x_k, y_k)\}\)).

**Proof.** The definition of \( \lambda^u \) is clearly independent of the ordering of \((x_1, y_1), \ldots, (x_k, y_k)\). So it suffices to show that we can swap adjacent elements of \( u \). Suppose that \( \sigma = (x_k z_k) \cdots (x_1 z_1)\lambda^u \) where \( z_i \in [n] - \{x_1, \ldots, x_k\} \).

Suppose we swap the \( j \) and \( j + 1 \) entry; we will show that there exist \( w_j, w_{j+1} \) such that

\[
\sigma = (x_k z_k) \cdots (x_j w_j)(x_{j+1} w_{j+1})(x_{j-1} z_{j-1}) \cdots (x_1 z_1)\lambda^u
\]

and \( w_j \notin \{x_j, x_{j+2}, \ldots, x_k\}, w_{j+1} \notin \{x_j, x_{j+1}, x_{j+2}, \ldots, x_k\} \). This will show that we can swap the ordering of \((x_j, y_j), (x_{j+1}, y_{j+1})\) in the ordering of \( u \). Note that we must have \( z_j \neq x_j, z_j \neq x_{j+1}, z_{j+1} \neq x_{j+1} \).

It suffices to show that

\[
(x_j w_j)(x_{j+1} w_{j+1}) = (x_{j+1} z_{j+1})(x_j z_j)
\]

There are a few cases.

1. If \( z_j, z_{j+1} \) are both distinct from \( x_j, x_{j+1} \) and each other, then \((x_{j+1} z_{j+1})(x_j z_j) = (x_{j+1} z_{j+1})(x_j z_j)\) and so \( w_j = z_j, w_{j+1} = z_{j+1} \) works.

2. If \( z_j = z_{j+1} = z \), then \((x_{j+1} z_{j+1})(x_j z_j) = (x_j x_{j+1} z) = (x_j x_{j+1})(x_{j+1} z)\). Thus taking \( w_j = x_{j+1} \) and \( w_{j+1} = z \) works.

3. If \( z_{j+1} = x_j \), then \((x_{j+1} z_{j+1})(x_j z_j) = (x_j z_j x_{j+1}) = (x_j z_j)(x_{j+1} z_j)\). Thus taking \( w_j = z_j, w_{j+1} = z_j \) works.

\[\square\]

We now define the resampling generator by \( R(u) = T(u) \). Thus, a resampling map \( \mathcal{R}_B(\pi) \) maps \( \pi \) to \( \sigma \pi \) for \( \sigma \in T(u) \).

**Proposition C.18.** Let \( u \in U \). If \( \pi \in (u) \) and \( \sigma \in T(u) \) then \( \sigma \pi \in W \).
Proof. Let us write $\pi = (a_1 \ a_2 \ldots \ a_n)$ and let $q(u) = \{x_1, \ldots, x_k\}$. Let $a'$ be the list obtained from $a_1, a_2, \ldots, a_n$ by deleting all occurrences of $x_1, \ldots, x_k$. As we have seen, $\rho_0$ contains $k$ fixed points $x_1, \ldots, x_k$, and contains a cycle of length $n - k$ of the form $(a'_1 \ a'_2 \ldots a'_{n-k})$.

Next, we claim that for $i = 0, 1, \ldots, k$, the permutation $\rho_i = (x_i \ z_i)\ldots(x_1 \ z_1)\lambda^n \pi$ contains a single cycle of length $n - k + i$ on the set $[n] - \{x_{i+1}, \ldots, x_k\}$, as well as $k - i$ fixed points $x_{i+1}, \ldots, x_k$. We have already shown the base case $i = 0$. For the induction step, observe that $\rho_{i+1}x_j = (x_{i+1} \ z_{i+1})\rho_i x_j = x_j$ for $j > i$, as neither of $z_{i+1}$ or $x_{i+1}$ is equal to $x_j$. Also, the action of multiplying by $(x_{i+1} \ z_{i+1})$ on the left (for $z_i \neq x_i$) inserts $x_i$ in between the elements $\rho_i^{-1}z_i$ and $z_i$ in the cycle structure of $\rho_i$. \hfill \square

Proposition C.19. Property (C3) holds.

Proof. Consider $u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \in U$. Note that $R(u)$ has size at most $(n-1)!/(n-1-k)!$. The correspondence between $\langle u \rangle$ and length $n - 1 - k$ cycles shows that $|\langle u \rangle|$ has size precisely $(n-1-k)!$. So $|\langle u \rangle \times R(u)| \leq |W|$. In order to show that the function $F_u$ is a bijection onto $W$, it thus suffices to show that $F_u$ is injective.

Suppose that $\pi \in \langle u \rangle$. Let $\rho = \lambda^n \pi$ and suppose

$$\sigma = (x_k \ z_k)\ldots(x_1 \ z_1)\lambda^n \pi = (x_k \ z_k)\ldots(x_1 \ z_1)$$

where $(x_1 \ z_1)\ldots(x_k \ z_k)\lambda^n \in T(u)$.

We claim that $z_k, \ldots, z_1$ are uniquely determined. For, observe that $\sigma x_k = z_k$; this follows since $z_{k-1}, \ldots, z_1$ are all distinct from $x_k$ and $\rho x_k = x_k$.

So we determine $z_k$ and peel of $(x_k \ z_k)$ to determine $(x_{k-1} \ z_{k-1})\ldots(x_1 \ z_1)\rho$. Continuing in this way, we can determine $z_k, z_{k-1}, \ldots, z_1$ and finally $\pi$. \hfill \square

Corollary C.20. For any $u \in U$ we have $|T(u)| = (n-1)!/(n-1-k)!$.

Proof. By Proposition C.19 $F_u$ is a bijection between $\langle u \rangle \times R(u)$ and $W$. So $|T(u)| = |W|/|\langle u \rangle| = (n-1)!/(n-1-k)!$. \hfill \square

Proposition C.21. Property (C4) holds.

Proof. Let $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ and $(x, y) \not\in u$. Consider $\pi \in \langle u \rangle$ with $\pi x = y$.

Let $z_1, \ldots, z_k$ be chosen such that $z_i \in [n] - \{x_i, \ldots, x_k\}$ and let $\sigma = (x_k \ z_k)\ldots(x_1 \ z_1)\lambda^n$. We will show that that $\sigma \pi x = y$.

Let $\rho = \lambda^n \pi$. Since $x, y$ are distinct from $x_1, \ldots, x_k, y_1, \ldots, y_k$ we have $\lambda^n x = x$; thus $\rho x = \pi x = y$.

For each $i = 0, \ldots, k$ define $\sigma_i = (x_i \ z_i)\ldots(x_1 \ z_1)\rho$. Suppose that $i$ is minimal such that $\sigma_i x = y$. Since $\sigma_0 = \rho$, we must have $i > 0$. This implies that either (a) $x_i = \sigma_{i-1} x_i$, $y_i = y$ or (b) $z_i = \sigma_{i-1} x_i$, $x_i = y$. The former cannot occur as $\sigma_{i-1} x_i = x_i$ and the latter cannot occur as $x_i \neq y$. \hfill \square

Proposition C.22. Suppose $u = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ and let $v = (a, b)$ where $u \not\in v$. Let $\pi \in \langle u \cup v \rangle$ and let $\sigma = (x_k \ z_k)\ldots(x_1 \ z_1)\lambda^n$ where $z_i \in [n] - \{x_i, \ldots, x_k\}$. Then $\sigma \pi \in \langle v \rangle$ iff $z_1, \ldots, z_k$ are all distinct from $b$.

Proof. For each $i = 0, \ldots, k$ let $\sigma_i = (x_i \ z_i)\ldots(x_1 \ z_1)\lambda^n$.

For the reverse direction: if $b$ is distinct from $z_1, \ldots, z_k$ then $\sigma \pi a = \sigma b = b$.

To show the forward direction, let $i \leq k$ be minimal such that $z_i = b$. Then $\sigma_i \pi a = (x_i \ b) (x_{i-1} \ z_{i-1}) \ldots (x_1 \ z_1) \pi a = x_i$. \hfill \square

34
We claim now that for all \( j \geq i \) we have \( \sigma_j \pi a \in \{x_1, \ldots, x_k\} \). We show this by induction on \( j \). The base case \( j = i \) is already shown. To show the induction step, suppose that \( \sigma_{j-1} \pi a = x_r \). If \( z_i \neq x_r \) we have \( \sigma_j \pi a = \sigma_{j-1} \pi a = x_r \) as desired. If \( z_j = x_r \), then \( \sigma_j \pi a = (x_j \ x_r) \sigma_{j-1} x_r = x_j \), again as desired.

Thus, we see that if some of the \( z_i \) are equal to \( b \) then \( \sigma \pi a \in \{x_1, \ldots, x_k\} \), and in particular \( \sigma \pi a \neq b \). \( \square \)

**Corollary C.23.** Property (C6) holds.

**Proof.** By Proposition C.22 \( \sigma \pi \in \langle (a, b) \rangle \) iff \( z_1, \ldots, z_k \) are distinct from \( b \). This criterion does not depend on the precise value of \( \pi \), so it either holds for all \( \pi \in \langle u \cup v \rangle \) or none of them. \( \square \)

**Proposition C.24.** Property (C5) holds.

**Proof.** With our definition of \( R \), we need to show that

\[
T(v) T(u; v) = T(u \cup v)
\]

where \( T(u; v) \) is the set of all permutations \( \sigma \in T(u) \) such that \( \sigma \langle u \cup v \rangle \subseteq \langle v \rangle \).

We first show that LHS is contained in the RHS. By Proposition C.16 neither \( u \) nor \( v \) nor \( u \cup v \) can be degenerate (as then \( u \sim v \)). Let \( u = \{(x_1, y_1), \ldots, (x_k, y_k)\} \) and \( v = \{(a_1, b_1), \ldots, (a_\ell, b_\ell)\} \). Let \( \sigma \in T(u; v) \) and \( \rho \in Z(v) \). Proposition C.23 shows that

\[
\begin{align*}
\sigma &= (x_1 z_1) \ldots (x_k z_k) \lambda^u \\
&\quad \text{for } z_i \in [n] - \{x_1, \ldots, x_k, b_1, \ldots, b_\ell\} \\
\rho &= (a_1 c_1) \ldots (a_\ell c_\ell) \lambda^v \\
&\quad \text{for } c_i \in [n] - \{a_1, \ldots, a_\ell\}
\end{align*}
\]

Now, consider any path \( p = r_1, \ldots, r_j \in v \); there is a corresponding cycle \( \lambda^p = (r_j \ldots r_1) \) in \( \lambda^v \). The values \( r_2, \ldots, r_j \) will appear in the list \( b_1, \ldots, b_\ell \), and the values of \( x_1, \ldots, x_k \) are distinct from all the entries in \( r \).

If \( z_i \neq r_1 \), then \( \lambda^p(x_i z_i) = (x_i z_i) \lambda^p \). Otherwise, for \( z_i = r_1 \), we have \( \lambda^p(x_i z_i) = \lambda^p(x_i r_1) = (x_i r_1 \ldots r_1) = (x_i r_1) \lambda^p \). Thus, we see that

\[
\lambda^p(x_k z_k) \ldots (x_1 z_1) = (x_k z_k') \ldots (x_1 z_1') \lambda^p
\]

where

\[
z_i' = \begin{cases} r_j & \text{if } z_i = r_1 \\ z_i & \text{otherwise} \end{cases}
\]

Since \( \lambda^v \) is a product of such cycles, we have

\[
\lambda^v(x_k z_k) \ldots (x_1 z_1) = (x_k z_k') \ldots (x_1 z_1') \lambda^v
\]

where \( z_i'' \) is defined as

\[
z_i'' = \begin{cases} \text{end}(p) & \text{if } z_i = \text{start}(p) \text{ for some path } p \text{ of } v \\ z_i & \text{otherwise} \end{cases}
\]

So we have

\[
\rho \sigma = (a_1 c_1) \ldots (a_\ell c_\ell) \lambda^v(x_k z_k) \ldots (x_1 z_1) \lambda^u
\]
\[
= (a_1 c_1) \ldots (a_\ell c_\ell)(x_k z_k') \ldots (x_1 z_1') \lambda^v \lambda^u
\]
\[
= (a_1 c_1) \ldots (a_\ell c_\ell)(x_k z_k') \ldots (x_1 z_1') \lambda^{u+v}
\]
where the last line follows as \( u \not\sim v \).

We now claim that \( z''_i \notin \{x_i, \ldots, x_k, a_1, \ldots, a_\ell\} \). First, the start and endpoints of the paths of \( v \) are all distinct from \( x_1, \ldots, x_k \), and we know that \( z_i \notin \{x_i, \ldots, x_k\} \). Suppose that \( z''_i = a_j \). Since \( a_j \) is not a endpoint of a path of \( v \), we have \( z''_i = z_i \). If \( a_j \) is a startpoint a path of \( u_2 \), then \( z''_i \) would be the endpoint of that path, and that has not occurred. So \( a_j \) is in the middle of such a path, and \( a_j = b_{j'} \) for some \( j' \). By this would mean that \( z_i = b_{j'} \), which cannot occur.

Consequently,

\[
\rho \sigma = (a_1 c_1) \ldots (a_\ell c_\ell) (x_k z''_k) \ldots (x_1 z''_1) \lambda^u v \in T(u \cup v)
\]

To show the RHS is contained in the LHS, consider any permutation

\[
\sigma = (a_1 c_1) \ldots (a_\ell c_\ell) (x_k z_k) \ldots (x_1 z_1) \lambda^u v \in T(u \cup v)
\]

where \( z_i \notin \{x_i, \ldots, x_k, a_1, \ldots, a_\ell\} \) and \( c_i \notin \{a_i, \ldots, a_\ell\} \). By the same reasoning as above, we have

\[
(x_k z_k) \ldots (x_1 z_1) \lambda^u = \lambda^u (x_k z''_k) \ldots (x_1 z''_1)
\]

where we define

\[
z''_i = \begin{cases} 
\text{start}(p) & \text{if } z_i = \text{end}(p) \text{ for some } p \text{ of } v \\
 z_i & \text{otherwise}
\end{cases}
\]

So we have

\[
\sigma = (a_1 c_1) \ldots (a_\ell c_\ell) \lambda^u (x_k z''_k) \ldots (x_1 z''_1) \lambda^u
\]

We claim that \( z''_i \notin \{x_i, \ldots, x_k, b_1, \ldots, b_\ell\} \). For, suppose \( z''_i = x_j \) for \( j \geq i \). Since \( u \not\sim v \), \( x_j \) cannot be equal to any entry of \( a \), so \( x_j \) is not a startpoint of a path of \( v \). Therefore, \( z''_i = z_i = x_j \), which cannot occur.

Next suppose \( z''_i = b_j \). Since \( b_j \) is not a startpoint of a path of \( v \), we have \( z''_i = z_i \). If \( b_j \) is an endpoint of a path of \( u_2 \), then \( z''_i \) would be the startpoint of that path, and that has not occurred. So \( b_j \) is the middle of such a path, and \( b_j = a_{j'} \) for some \( j' \). So \( z_i = a_{j'} \), which cannot occur.

So, \( (x_k z''_k) \ldots (x_1 z''_1) \lambda^u \in T(u; v) \). Clearly, \( (a_1 c_1) \ldots (a_\ell c_\ell) \lambda^v \in Z(v) \). So \( \sigma \in T(u; v)T(v) \).

\( \square \)