Uniform Sampling Through the Lovász Local Lemma

HENG GUO, University of Edinburgh
MARK JERRUM, Queen Mary, University of London
JINGCHENG LIU, University of California, Berkeley

We propose a new algorithmic framework, called *partial rejection sampling*, to draw samples exactly from a product distribution, conditioned on none of a number of bad events occurring. Our framework builds new connections between the variable framework of the Lovász Local Lemma and some classical sampling algorithms such as the cycle-popping algorithm for rooted spanning trees. Among other applications, we discover new algorithms to sample satisfying assignments of k-CNF formulas with bounded variable occurrences.

CCS Concepts: • **Theory of computation → Generating random combinatorial structures**; *Random walks and Markov chains*;

Additional Key Words and Phrases: Exact sampling, Lovász Local Lemma, #SAT

ACM Reference format:
Heng Guo, Mark Jerrum, and Jingcheng Liu. 2019. Uniform Sampling Through the Lovász Local Lemma. *J. ACM* 66, 3, Article 18 (April 2019), 31 pages.
https://doi.org/10.1145/3310131

1 INTRODUCTION

The Lovász Local Lemma (LLL) [9] is a classical gem in combinatorics that guarantees the existence of a perfect object that avoids all events deemed to be "bad." The original proof is non-constructive, but there has been great progress in the algorithmic aspects of the local lemma. After a long line of research [2, 3, 8, 30, 34, 37], the celebrated result by Moser and Tardos [31] gives efficient algorithms to find such a perfect object under conditions that match the LLL in the so-called variable framework. However, it is natural to ask whether, under the same condition, we can also sample a perfect object uniformly at random instead of merely finding one.

Roughly speaking, the resampling algorithm by Moser and Tardos [31] works as follows. We initialize all variables randomly. If bad events occur, then we arbitrarily choose a bad event and resample all the involved variables. Unfortunately, it is not hard to see that this algorithm can produce biased samples. This seems inevitable. As Bezáková et al. [4] showed, sampling can be N-hard

H. Guo and M. Jerrum were supported by the EPSRC grant EP/N004221/1. J. Liu was supported by NSF grant CCF-1420934. This work was done (in part) while the authors were visiting the Simons Institute for the Theory of Computing. H. Guo was also supported by a Google research fellowship at the Simons Institute.
Authors' addresses: H. Guo, School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK; email: hguo@inf.ed.ac.uk; M. Jerrum, School of Mathematical Sciences, Queen Mary, University of London, London, E1 4NS, UK; email: mj@qmul.ac.uk; J. Liu, Department of EECS, University of California, 94720 Berkeley, Berkeley, CA; email: liuexp@berkeley.edu.

0004-5411/2019/04-ART18 $15.00
https://doi.org/10.1145/3310131

even under conditions that are stronger than those of the local lemma. On the one hand, the symmetric LLL only requires $ep\Delta \leq 1$, where p is the probability of bad events and Δ is the maximum degree of the dependency graph. On the other hand, translating the result of Bezáková et al. [4] to this setting, one sees that as soon as $p\Delta^2 \geq C$ for some constant C, then even approximately sampling perfect objects in the variable framework becomes NP-hard.

The starting point of our work is a new condition (see Condition 5) under which we show that the output of the Moser-Tardos algorithm is in fact uniform (see Theorem 8). Intuitively, the condition requires any two dependent bad events to be disjoint. Indeed, instances satisfying this condition are called *extremal* in the study of the LLL. For these extremal instances, we can in fact resample in a parallel fashion, since the occurring bad events form an independent set in the dependency graph. We call this algorithm *partial rejection sampling*,[1] in the sense that it is like rejection sampling but only resamples an appropriate subset of variables.

Our result puts some classical sampling algorithms under a unified framework, including the cycle-popping algorithm by Wilson [39] for sampling rooted spanning trees, and the sink-popping algorithm by Cohn et al. [7] for sampling sink-free orientations of an undirected graph. Indeed, Cohn et al. [7] coined the term *partial rejection sampling* and asked for a general theory, and we believe that extremal instances under the variable framework is a satisfactory answer. With our techniques, we are able to give a new algorithm to sample solutions for a special class of k-CNF formulas, under conditions matching the LLL (see Corollary 19), which is an NP-hard task for general k-CNF formulas. Furthermore, we provide *explicit* formulas for the expected running time of these algorithms (see Theorem 13), which matches the running time upper bound given by Kolipaka and Szegedy [26] under the condition of Shearer [35].

The next natural question is thus whether we can go beyond extremal instances. Indeed, our main technical contribution is a general uniform sampler (see Algorithm 6) that applies to *any* problem under the variable framework. The main idea is that instead of only resampling occurring bad events, we resample a larger set of events so that the choices made do not block any perfect assignments in the end to make sure of uniformity in the final output.

As a simple example, we describe how our algorithm samples independent sets. The algorithm starts by choosing each vertex with probability 1/2 independently. At each subsequent round, in the induced subgraph on the currently chosen vertices, the algorithm finds all the connected components of size ≥ 2. Then it resamples all of these vertices and their boundaries (which are unoccupied). And it repeats this process until there is no edge with both endpoints occupied. What seems surprising is that this simple process does yield a uniformly random independent set when it stops. Indeed, as we will show in Theorem 35, this simple process is an exact sampler for *weighted independent sets* (also known as the hard-core model in statistical physics). In addition, it runs in expected linear time under a condition that matches, up to a constant factor, the *uniqueness threshold* of the model (beyond which the problem of approximate sampling becomes NP-hard).

In the more general setting, we will choose the set of events to be resampled, denoted by Res, iteratively. We start from the set of occurring bad events. Then we include all neighboring events of the current set Res until there is no event A on the boundary of Res such that the current assignment, projected on the common variables of A and Res, can be extended so that A may happen. In the worst case, we will resample all events (there is no event in the boundary at all). In that scenario, the algorithm is the same as a naive rejection sampling, but typically we resample

[1] Despite the apparent similarity in names, our algorithm is different from partial resampling in Harris and Srinivasan [20, 21]. We resample *all* variables in certain sets of events, whereas partial resampling only resamples a subset of variables from some bad event.

fewer variables in every step. We show that this is a uniform sampler on assignments that avoid all bad events once it stops (see Theorem 25).

One interesting feature of our algorithm is that unlike Markov chain–based algorithms, ours does not require the solution space (or any augmented space) to be connected. Moreover, our sampler is exact—that is, when the algorithm halts, the final distribution is precisely the desired distribution. Prior to our work, most exact sampling algorithms were obtained by coupling from the past [32]. We also note that previous work on the Moser-Tardos output distribution, such as Haeupler et al. [19], is not strong enough to guarantee a uniform sample (or ε-close to uniform in terms of total variation distances).

We give sufficient conditions that guarantee a linear expected running time of our algorithm in the general setting (see Theorem 26). The first condition is that $p\Delta^2$ is bounded above by a constant. This is optimal up to constants in observance of the NP-hardness result in Bezáková et al. [4]. Unfortunately, the condition on $p\Delta^2$ alone does not make the algorithm efficient. In addition, we also need to bound the expansion from bad events to resampling events, which leads to an extra condition on intersections of bad events. Removing this extra condition seems to require substantial changes to our current algorithm.

To illustrate the result, we apply our algorithm to sample satisfying assignments of k-CNF formulas in which the degree of each variable (the number of clauses containing it) is at most d. We say that a k-CNF formula has intersection s if any two *dependent* clauses share at least s variables. The extra condition from our analysis naturally leads to a lower bound on s. Let n be the number of variables. We (informally) summarize our results on k-CNF formulas as follows (see Corollary 31 and Theorem 33):

- If $d \leq \frac{1}{6e} \cdot 2^{k/2}$, $dk \geq 2^{3e}$ and $s \geq \min\{\log_2 dk, k/2\}$, then the general partial rejection resampling algorithm outputs a uniformly random solution to a k-CNF formula with degree d and intersection s in expected running time $O(n)$.
- If $d \geq 4 \cdot 2^{k/2}$ (for an even k), then even if $s = k/2$, it is NP-hard even to *approximately* sample a solution to a k-CNF formula with degree d and intersection s.

As shown in the hardness result, the intersection bound does not render the problem trivial.

Previously, sampling/counting satisfying assignments of k-CNF formulas required the formula to be monotone and $d \leq k$ to be large enough [4] (see also [5, 28]). Although our result requires an additional lower bound on intersections, not only does it improve the dependency of k and d *exponentially* but it also achieves a *matching* constant $1/2$ in the exponent. Furthermore, the samples produced are exactly uniform. Thus, if the extra condition on intersections can be removed, we will have a sharp phase transition at around $d = O(2^{k/2})$ in the computational complexity of sampling solutions to k-CNF formulas with bounded variable occurrences. A similar sharp transition has been recently established, for example, for sampling configurations in the hard-core model [12, 36, 38].

Simultaneous to our work, Hermon et al. [24] showed that Markov chains for monotone k-CNF formulas are rapidly mixing, if $d \leq c2^{k/2}$ for a constant c. In another parallel work, Moitra [29] gave a novel algorithm to sample solutions for general k-CNF when $d \lesssim 2^{k/60}$. We note that neither results are directly comparable to ours, and the techniques are very different. Both of these two samplers are approximate, whereas ours is exact. Moreover, ours does not require monotonicity (unlike Hermon et al. [24]) and allows larger d than Moitra [29] but at the cost of an extra intersection lower bound. Unfortunately, our algorithm can be exponentially slow when the intersection s is not large enough. In sharp contrast, as shown by Hermon et al. [24], Markov chains mix rapidly for $d \leq c2^k/k^2$ when $s = 1$.

Although the study of algorithmic LLL has progressed beyond the variable framework [1, 22, 23], we restrict our focus to the variable framework in this work. It is also an interesting future direction to investigate and extend our techniques of uniform sampling beyond the variable framework. For example, one may want to sample a permutation that avoids certain patterns. The classical sampling problem of perfect matchings in a bipartite graph can be formulated in this way.

Since the conference version of this article appeared [18], a number of applications of the partial rejection sampling method have been found [15–17]. One highlight is the first fully polynomial-time randomized approximation scheme (FPRAS) for all-terminal network reliability [17]. For the extremal instances, tight running time bounds have also been obtained [14]. Moreover, partial rejection sampling is adapted to dynamic and distributed settings as well [10].

2 PARTIAL REJECTION SAMPLING

We first describe the "variable" framework. Let $\{X_1, \ldots, X_n\}$ be a set of random variables. Each X_i can have its own distribution and range D_i. Let $\{A_1, \ldots, A_m\}$ be a set of "bad" events that depend on X_i's. For example, for a constraint satisfaction problem (CSP) with variables X_i $(1 \le i \le n)$ and constraints C_j $(1 \le j \le m)$, each A_j is the set of unsatisfying assignments of C_j for $1 \le j \le m$. Let $\mathrm{var}(A_i)$ be the (index) set of variables that A_i depends on.

The dependency graph $G = (V, E)$ has m vertices, identified with the integers $\{1, 2, \ldots, m\}$, corresponding to the events A_i, and (i, j) is an edge if A_i and A_j depend on one or more common variables, and $i \ne j$. In other words, for any distinct i, j, $(i, j) \in E$ if $\mathrm{var}(A_i) \cap \mathrm{var}(A_j) \ne \emptyset$. We write $A_i \sim A_j$ if the vertices i and j are adjacent in G. The asymmetric LLL [9] states the following.

Theorem 1. *If there exists a vector $\boldsymbol{x} \in [0, 1)^m$ such that $\forall i \in [m]$,*

$$\Pr(A_i) \le x_i \prod_{(i,j)\in E} (1 - x_j), \tag{1}$$

then $\Pr(\bigwedge_{i=1}^m \overline{A_i}) \ge \prod_{i=1}^m (1 - x_i) > 0$.

Theorem 1 has a condition that is easy to verify but not necessarily optimal. Shearer [35] gave the optimal condition for the local lemma to hold for a fixed dependency graph G. To state Shearer's condition, we will need the following definitions. Let $p_i := \Pr(A_i)$ for all $1 \le i \le m$. Let $\mathcal{I}$ be the collection of independent sets of G. Define the following quantity:

$$q_I(\mathbf{p}) := \sum_{J\in\mathcal{I},\, I\subseteq J} (-1)^{|J|-|I|} \prod_{i\in J} p_i,$$

where $\mathbf{p} = (p_1, \ldots, p_m)$. When there is no confusion, we also simply write q_I instead of the more cumbersome $q_I(\mathbf{p})$. Moreover, to simplify the notation, let $q_i := q_{\{i\}}$ for $1 \le i \le m$. Note that if $I \notin \mathcal{I}$, $q_I = 0$.

Theorem 2 (Shearer [35]). *If $q_I \ge 0$ for all $I \subseteq V$, then $\Pr(\bigwedge_{i=1}^m \overline{A_i}) \ge q_\emptyset$.*

In particular, if the condition holds with $q_\emptyset > 0$, then $\Pr(\bigwedge_{i=1}^m \overline{A_i}) > 0$.

Neither Theorem 1 nor Theorem 2 yields an efficient algorithm to find the assignment avoiding all bad events, since they only guarantee an exponentially small probability. There has been a long line of research devoted to an algorithmic version of the LLL, culminating in Moser and Tardos [31] with essentially the same condition as in Theorem 1. The Resample algorithm of Moser and Tardos is very simple, described in Algorithm 1.

Moser and Tardos [31] showed that Algorithm 1 finds a good assignment very efficiently.

Theorem 3 (Moser and Tardos [31]). *Under the condition of Theorem 1, the expected number of resampling steps in Algorithm 1 is at most $\sum_{i=1}^m \frac{x_i}{1-x_i}$.*

ALGORITHM 1: The Resample Algorithm

(1) Draw independent samples of all variables $X_1, \ldots, X_n$ from their respective distributions.
(2) While at least one A_i holds, pick one such A_i arbitrarily and resample all variables in $\mathrm{var}(A_i)$.
(3) Output the current assignment.

Unfortunately, the final output of Algorithm 1 is not distributed as we would like, namely as a product distribution conditioned on avoiding all bad events.

In addition, Kolipaka and Szegedy [26] showed that up to Shearer's condition, Algorithm 1 is efficient. Recall that $q_i := q_{\{i\}}$ for $1 \le i \le m$.

Theorem 4 (Kolipaka and Szegedy [26]). *If $q_I \ge 0$ for all $I \in \mathcal{I}$ and $q_\emptyset > 0$, then the expected number of resampling steps in Algorithm 1 is at most $\sum_{i=1}^{m} \frac{q_i}{q_\emptyset}$.*

However, the cycle-popping algorithm of Wilson [39] is very similar to the Resample algorithm, but it outputs a uniformly random rooted spanning tree. Another similar algorithm is the sink-popping algorithm by Cohn et al. [7] to generate a sink-free orientation uniformly at random. Upon close examination of these two algorithms, we found a common feature of both problems.

Condition 5. *If $(i,j) \in E$ (or equivalently $A_i \sim A_j$), then $\Pr(A_i \wedge A_j) = 0$; namely, the two events A_i and A_j are disjoint if they are dependent.*

In other words, any two events A_i and A_j are either independent or disjoint. These instances have been noticed in the study of the LLL. They are the ones that minimize $\Pr(\bigwedge_{i=1}^{m} \overline{A_i})$ given Shearer's condition (namely, $\Pr(\bigwedge_{i=1}^{m} \overline{A_i}) = q_\emptyset$). Instances satisfying Condition 5 have been named *extremal* [26].

We will show that, given Condition 5, the final output of the Resample algorithm is a sample from a conditional product distribution (Theorem 8). Moreover, we will show that under Condition 5, the running time upper bound $\sum_{i=1}^{m} \frac{q_i}{q_\emptyset}$ given by Kolipaka and Szegedy (Theorem 4) is indeed the exact expected running time. See Theorem 13.

In fact, when Condition 5 holds, at each step of Algorithm 1, the occurring events form an independent set of the dependency graph G. Think of the execution of Algorithm 1 as going in rounds. In each round, we find the set I of bad events that occur. Due to Condition 5, $\mathrm{var}(A_i) \cap \mathrm{var}(A_j) = \emptyset$ for any $i, j \in I$ (i.e., I is an independent set in the dependency graph). Therefore, we can resample all variables involved in the occurring bad events without interfering with each other. This motivates Algorithm 2.

ALGORITHM 2: Partial Rejection Sampling for Extremal Instances

(1) Draw independent samples of all variables $X_1, \ldots, X_n$ from their respective distributions.
(2) While at least one bad event holds, find the independent set I of occurring A_i's. Independently resample all variables in $\bigcup_{i \in I} \mathrm{var}(A_i)$.
(3) Output the current assignment.

We call Algorithm 2 the Partial Rejection Sampling algorithm. This name was coined by Cohn et al. [7]. Indeed, they ask as an open problem how to generalize their sink-popping algorithm and Wilson's cycle-popping algorithm. We answer this question under the variable framework. Partial Rejection Sampling differs from the normal rejection sampling algorithm

Table 1. A Resampling Table With Four Variables

X_1	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	...
X_2	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	...
X_3	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	...
X_4	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	...

by only resampling "bad" events. Moreover, Algorithm 2 is uniform only on extremal instances and is a special case of Algorithm 6 given in Section 5, which is a uniform sampler for all instances.

In fact, Algorithm 2 is the same as the parallel version of Algorithm 1 by Moser and Tardos [31] for extremal instances. Suppose each event is assigned to a processor, which determines whether the event holds by looking at the variables associated with the event. If the event holds, then all associated variables are resampled. No conflict will be created due to Condition 5.

In the following analysis, we will use the resampling table idea, which has appeared in both the analysis of Moser and Tardos [31] and Wilson [39]. Note that we only use this idea to analyze the algorithm rather than to really create the table in the execution. Associate each variable X_i with an infinite stack of random values $\{X_{i,1}, X_{i,2}, \ldots\}$. This forms the *resampling table* where each row represents a variable and there are infinitely many columns, as shown in Table 1. In the execution of the algorithm, when a variable needs to be resampled, the algorithm draws the top value from the stack or equivalently moves from the current entry in the resampling table to its right.

Let t be a positive integer to denote the round of Algorithm 2. Let $j_{i,t}$ be the index of the variable X_i in the resampling table at round t. In other words, at the t-th round, X_i takes value $X_{i,j_{i,t}}$. Thus, the set $\sigma_t = \{X_{i,j_{i,t}} \mid 1 \le i \le n\}$ is the current assignment of variables at round t. This σ_t determines which events happen. Call the set of occurring events, viewed as a subset of the vertex set of the dependency graph, I_t. (For convenience, we shall sometimes identify the event A_i with its index i; thus, we shall refer to the "events in S" rather than the "events indexed by S.") As explained earlier, I_t is an independent set of G due to Condition 5. Then variables involved in any of the events in I_t are resampled. In other words,

$$j_{i,t+1} = \begin{cases} j_{i,t} + 1 & \text{if } \exists \ell \in I_t \text{ such that } i \in \mathrm{var}(A_\ell); \\ j_{i,t} & \text{otherwise.} \end{cases}$$

Therefore, any event that happens in round $t+1$ must share at least one variable with some event in I_t (possibly itself). In other words, $I_{t+1} \subseteq \Gamma^+(I_t)$, where $\Gamma^+(\cdot)$ denotes the set of all neighbors of I unioned with I itself. This inspires the notion of independent set sequences (first introduced in Kolipaka and Szegedy [26]).

Definition 6. A list $\mathcal{S} = S_1, S_2, \ldots, S_\ell$ of independent sets in G is called an *independent set sequence* if $S_i \neq \emptyset$ for all $1 \le i \le \ell - 1$ and for every $1 \le i \le \ell - 1$, $S_{i+1} \subseteq \Gamma^+(S_i)$.

We adopt the convention that the empty list is an independent set sequence with $\ell = 0$. Note that we allow S_ℓ to be $\emptyset$.

Let M be a resampling table. Suppose running Algorithm 2 on M does not terminate up to some integer $\ell \ge 1$ rounds. Define the *log* of running Algorithm 2 on M up to round ℓ as the sequence of independent sets $I_1, I_2, \ldots, I_\ell$ created by this run. Thus, for any M and $\ell \ge 1$, the log $I_1, I_2, \ldots, I_\ell$ must be an independent set sequence. Moreover, if Algorithm 2 terminates at round T, let $\sigma_t := \sigma_T$ if $t > T$. Denote by $\mu(\cdot)$ the product distribution of all random variables.

Lemma 7. *Suppose Condition 5 holds. Given any log $\mathcal{S} = S_1, S_2, \ldots, S_\ell$ of length $\ell \geq 1$ and conditioned on seeing the log $\mathcal{S}$, $\sigma_{\ell+1}$ is a random sample from the product distribution conditioned on the event $\bigwedge_{i\in[m]\setminus\Gamma^+(S_\ell)} \overline{A_i}$, namely from $\mu(\cdot \mid \bigwedge_{i\in[m]\setminus\Gamma^+(S_\ell)} \overline{A_i})$.*

We remark that Lemma 7 is not true for non-extremal instances (i.e., if Condition 5 fails). In particular, Lemma 7 says that given any log, every valid assignment is not only reachable but also with the correct probability. This is no longer the case for non-extremal instances—some valid assignments from the desired conditional product distribution could be "blocked" under the log $\mathcal{S}$. In Section 5, we show how to instead achieve uniformity by resampling an "unblocking" set of bad events.

Proof. The set of occurring events at round ℓ is S_ℓ. Hence, $\sigma_{\ell+1}$ does not make any of the A_i's happen where $i \notin \Gamma^+(S_\ell)$. Call an assignment σ *valid* if none of the A_i's happen where $i \notin \Gamma^+(S_\ell)$. To show that $\sigma_{\ell+1}$ has the desired conditional product distribution, we will show that the probabilities of getting any two valid assignments σ and σ' are proportional to their probabilities of occurrence in $\mu(\cdot)$.

Let M be the resampling table so that the log of the algorithm is $\mathcal{S}$ up to round $\ell \geq 1$, and $\sigma_{\ell+1} = \sigma$. Indeed, since we only care about events up to round $\ell + 1$, we may truncate the table so that $M = \{X_{i,j} \mid 1 \leq i \leq n,\ \ 1 \leq j \leq j_{i,\ell+1}\}$. Let $M' = \{X'_{i,j} \mid 1 \leq i \leq n,\ \ 1 \leq j \leq j_{i,\ell+1}\}$ be another table where $X'_{i,j} = X_{i,j}$ if $j < j_{i,\ell+1}$ for any $i \in [n]$ but $\sigma_{\ell+1} = \sigma'$. In other words, we only change the values in the final round $(X_{i,j_{i,\ell+1}})$ and only to another valid assignment.

We claim that the algorithm running on M' generates the same log $\mathcal{S}$. The lemma then follows by the following argument. Assuming that the claim holds, then conditioned on the log $\mathcal{S}$, every possible table M such that $\sigma_{\ell+1} = \sigma$ is one to one corresponding to another table M' so that $\sigma_{\ell+1} = \sigma'$. It implies that for every pair of valid assignments σ, σ', there is a bijection between the resampling tables resulting in them. The ratio between the probability of two corresponding tables is exactly the ratio between the probabilities of σ and σ' under $\mu(\cdot)$. Since the probability of getting a particular σ in round $\ell + 1$ is the sum over the probabilities of all resampling tables (conditioned on the log $\mathcal{S}$) leading to σ, the probability of getting σ is also proportional to its weight under $\mu(\cdot)$.

Suppose the claim fails and the logs obtained by running the algorithm on M and M' differ. Let $t_0 \leq \ell$ be the first round where resampling changes. Without loss of generality, let A be an event that occurs in S_{t_0} on M' but not on M. Moreover, there must be a non-empty set of variables $Y \subseteq \mathrm{var}(A)$ that have values $(X_{i,j_{i,\ell+1}})$, as otherwise the two runs would be identical. Since resampling history does not change before t_0, in the M' run, the assignment of variables in Y must be $(X'_{i,j_{i,\ell+1}})$ at time t_0.

We claim that $Y = \mathrm{var}(A)$. If the claim does not hold, then $Z := \mathrm{var}(A) \setminus Y \neq \emptyset$. Any variable in Z has not reached final round and must be resampled in the M run. Let $X_j \in Z$ be the first such variable being resampled at or after round t_0 in the M run. (The choice of X_j may not be unique, and we just choose an arbitrary one.) Recall that $Y \neq \emptyset$, A can no longer happen, and thus there must be $A' \neq A$ causing such a resampling of X_j. Then $\mathrm{var}(A) \cap \mathrm{var}(A') \neq \emptyset$. Consider any variable X_k with $k \in \mathrm{var}(A) \cap \mathrm{var}(A')$. It is resampled at or after time t_0 in the M run due to A'. Hence, $X_k \in Z$ for any such k. Moreover, in the M run, until A' happens, X_k has not been resampled since time t_0, because A' is the first resampling event at or after time t_0 that involves variables in Z. However, in the M' run, X_k's value causes A to happen at time t_0. Hence, there exists an assignment on variables in $\mathrm{var}(A) \cap \mathrm{var}(A')$ such that both A and A' happen. Clearly, this assignment can be extended to a full assignment so that both A and A' happen. However, $A \sim A'$ as they share the variable X_j. Due to Condition 5, $A \cap A' = \emptyset$. Contradiction! Therefore, the claim holds.

We argue that the remaining case, $Y = \mathrm{var}(A)$, is also not possible. Since A occurs in the M' run, we know, by the definition of σ', that $A \in \Gamma^+(S_\ell)$. Thus, some event whose variables intersect with

those in A must occur in the M run. But when the algorithm attempts to update variables shared by these two events in the M run, it will access values beyond the final round of the resampling table, a contradiction. □

Theorem 8. *When Condition 5 holds and Algorithm 2 halts, its output is the product distribution $\mu(\cdot)$ conditioned on avoiding all bad events.*

Proof. Let an independent set sequence $\mathcal{S}$ of length ℓ be the log of any successful run. Then $S_\ell = \emptyset$. By Lemma 7, conditioned on the log $\mathcal{S}$, the output assignment σ is $\mu(\cdot \mid \bigwedge_{i\in[m]\setminus\Gamma^+(S_\ell)} \overline{A_i}) = \mu(\cdot \mid \bigwedge_{i\in[m]} \overline{A_i})$. This is valid for any possible log, and the theorem follows. □

In other words, let Σ be the set of assignments that avoid all bad events. Let U be the output of Algorithm 2. In the case that all variables are sampled from the uniform distribution, we have $\Pr(U = \sigma) = \frac{1}{|\Sigma|}$, for all $\sigma \in \Sigma$.

3 EXPECTED RUNNING TIME OF ALGORITHM 2

In this section, we give an explicit formula for the running time of Algorithm 2. We assume that Condition 5 holds throughout the section.

We first give a combinatorial explanation of q_I for any independent set I of the dependency graph G. To simplify the notation, we denote the event $\bigwedge_{i\in S} A_i$—for instance, the conjunction of all events in S, by $A(S)$.

For any set I in the dependency graph, we denote by p_I the probability $\Pr_\mu(A(I))$ that all events in I happen (and possibly some other events too). If I is an independent set in the dependency graph, any two events in I are independent and

$$p_I = \prod_{i\in I} p_i. \tag{2}$$

Moreover, for any set J of events that is not an independent set, we have $p_J = 0$ due to Condition 5.

However, the quantity q_I is in fact the probability that exactly the events in I happen and no others do. This can be verified using inclusion-exclusion, together with Condition 5:

$$\begin{aligned}\Pr_\mu\left(\bigwedge_{i\in I} A_i \wedge \bigwedge_{i\notin I} \overline{A_i}\right) &= \sum_{J\supseteq I} (-1)^{|J\setminus I|} p_J \\ &= \sum_{J\in\mathcal{I},\ J\supseteq I} (-1)^{|J\setminus I|} p_J = q_I,\end{aligned} \tag{3}$$

where $\mathcal{I}$ denotes the collection of all independent sets of G. Since the events $(\bigwedge_{i\in I} A_i \wedge \bigwedge_{i\notin I} \overline{A_i})$ are mutually exclusive for different I's,

$$\sum_{I\in\mathcal{I}} q_I = 1.$$

Moreover, since the event $A(I)$ is the union over $J \supseteq I$ of the events $(\bigwedge_{i\in J} A_i \wedge \bigwedge_{i\notin J} \overline{A_i})$, we have

$$p_I = \sum_{J\in\mathcal{I},\ J\supseteq I} q_J. \tag{4}$$

Lemma 9. *Assume that Condition 5 holds. Let $\mathcal{S} = S_1, \ldots, S_\ell$ be an independent set sequence of length $\ell > 0$. Then in Algorithm 2,*

$$\Pr(\textit{the log is } \mathcal{S} \textit{ up to round } \ell) = q_{S_\ell} \prod_{t=1}^{\ell-1} p_{S_t}.$$

Proof. Clearly, if $q_{S_\ell} = 0$, then the said sequence will never happen. We assume that $q_{S_\ell} > 0$ in the following.

Recall that μ is the product distribution of sampling all variables independently. We need to distinguish the probability space with respect to μ from that with respect to the execution of the algorithm. We write $\Pr_{\text{PRS}}(\cdot)$ to refer to the algorithm and write $\Pr_\mu(\cdot)$ to refer to the (static) space with respect to μ. As noted before, to simplify the notation, we will use $A(S)$ to denote the event $\bigwedge_{i\in S} A_i$, where $S \subseteq [m]$. In addition, $B(S)$ will be used to denote $\bigwedge_{i\in S} \overline{A_i}$. For $I \in \mathcal{I}$, define

$$\partial I := \Gamma^+(I) \setminus I, \quad I^e := [m] \setminus \Gamma^+(I), \quad \text{and} \quad I^c := [m] \setminus I = \partial I \cup I^e.$$

Thus, ∂I is the "boundary" of I, comprising events that are not in I but which depend on I, and I^e is the "exterior" of I, comprising events that are independent of all events in I. The complement I^c is simply the set of all events not in I. Note that $B(I^c) = B(\partial I) \wedge B(I^e)$. As examples of the notation, $\Pr_\mu(A(I)) = \prod_{i\in I} p_i = p_I$ is the probability that all events in I occur under μ, and $\Pr_\mu(A(I) \wedge B(I^c)) = q_I$ is the probability that exactly the events in I occur.

By the definition of I^e, we have that

$$\Pr_\mu(B(I^e) \mid A(I)) = \Pr_\mu(B(I^e)) \tag{5}$$

and, by Condition 5, that

$$A(I) \wedge B(\partial I) = A(I). \tag{6}$$

Hence, for any $I \in \mathcal{I}$,

$$\begin{aligned} q_I &= \Pr_\mu(A(I) \wedge B(I^c)) \\ &= \Pr_\mu(A(I) \wedge B(\partial I) \wedge B(I^e)) \\ &= \Pr_\mu(A(I) \wedge B(I^e)) && \text{by (6)} \\ &= \Pr_\mu(A(I)) \Pr_\mu(B(I^e)). && \text{by (5)} \end{aligned} \tag{7}$$

We prove the lemma by induction. It clearly holds when $\ell = 1$. At round $\ell \geq 2$, since we only resample variables that are involved in $S_{\ell-1}$, we have that $S_\ell \subseteq \Gamma^+(S_{\ell-1})$. Moreover, variables are not resampled in any A_i where $i \in S^e_{\ell-1}$, and hence

$$B\left(S^c_\ell\right) \wedge B\left(S^e_{\ell-1}\right) = B\left(S^c_\ell\right). \tag{8}$$

Conditioned on $S_{\ell-1}$, by Lemma 7, the distribution of σ_ℓ at round ℓ is the product distribution conditioned on none of the events outside of $\Gamma^+(S_{\ell-1})$ occurring; namely, it is $\Pr_\mu(\cdot \mid B(S^e_{\ell-1}))$. Thus, the probability of getting S_ℓ in round ℓ is

$$\begin{aligned} &\Pr_{\text{PRS}}\left(A(S_\ell) \wedge B(S^c_\ell) \text{ holds in round } \ell \;\middle|\; \text{prior log is } S_1, \ldots, S_{\ell-1}\right) \\ &= \Pr_\mu\left(A(S_\ell) \wedge B(S^c_\ell) \mid B(S^e_{\ell-1})\right) \\ &= \frac{\Pr_\mu\left(A(S_\ell) \wedge B(S^c_\ell) \wedge B(S^e_{\ell-1})\right)}{\Pr_\mu(B(S^e_{\ell-1}))} \\ &= \frac{\Pr_\mu\left(A(S_\ell) \wedge B(S^c_\ell)\right)}{\Pr_\mu(B(S^e_{\ell-1}))} && \text{by (8)} \\ &= \frac{q_{S_\ell}}{\Pr_\mu(B(S^e_{\ell-1}))}. \end{aligned} \tag{9}$$

By (9) and the induction hypothesis, we have

$$\begin{aligned}\Pr_{\mathrm{PRS}}(\text{the log is } \mathcal{S} \text{ up to round } \ell) &= \frac{q_{S_\ell}}{\Pr_\mu(B(S^e_{\ell-1}))} \cdot q_{S_{\ell-1}} \prod_{t=1}^{\ell-2} p_{S_t} \\ &= q_{S_\ell} \prod_{t=1}^{\ell-1} p_{S_t},\end{aligned}$$

where to get the last line we used (7) on $S_{\ell-1}$. □

Essentially, the preceding proof is a delayed revelation argument. At each round $1 \le t \le \ell - 1$, we only reveal variables that are involved in S_t. Thus, at round ℓ, we have revealed all variables that are involved in $\mathcal{S}$. With respect to these variables, the sequence $\mathcal{S}$ happens with probability $p_{\mathcal{S}}$. Condition 5 guarantees that what we have revealed so far does not interfere with the final output (cf. Lemma 7). Hence, the final state happens with probability q_{S_ℓ}.

We write $p_{\mathcal{S}} = \prod_{i=1}^{\ell} p_{S_i}$ for an independent set sequence $\mathcal{S}$ of length $\ell \ge 0$. Note the convention that $p_{\mathcal{S}} = 1$ if $\mathcal{S}$ is empty and $\ell = 0$. Then, Lemma 9 implies the following equality, which is first shown by Kolipaka and Szegedy [26] in the more general (not necessarily extremal) setting of the local lemma.

Corollary 10. *Assume that Condition 5 holds. If $q_\emptyset > 0$, then*

$$\sum_{\mathcal{S} \text{ s.t. } S_1 = I} p_{\mathcal{S}} q_\emptyset = q_I,$$

where $\mathcal{S}$ is an independent set sequence and I is an independent set of G.

Proof. First we claim that if $q_\emptyset > 0$, then Algorithm 2 halts with probability 1. Conditioned on any log $\mathcal{S} = S_1, \ldots, S_{\ell-1}$, by Lemma 7, the distribution of σ_ℓ at round ℓ is $\mu(\cdot \mid B(S^e_{\ell-1}))$. The probability of getting a desired assignment is thus $\mu(B([m]) \mid B(S^e_{\ell-1})) = \frac{\mu(B([m]))}{\mu(B(S^e_{\ell-1}))} \ge \mu(B([m])) = q_\emptyset$. Hence, the probability that the algorithm does not halt at time t is at most $(1 - q_\emptyset)^t$, which goes to 0 as t goes to infinity.

Then we apply Lemma 9 when $\emptyset$ is the final independent set. The left-hand side is the total probability of all possible halting logs whose first independent set is exactly I. This is equivalent to getting exactly I in the first step, which happens with probability q_I. □

As a sanity check, the probability of all possible logs should sum to 1 when $q_\emptyset > 0$ and the algorithm halts with probability 1. Indeed, by Corollary 10,

$$\sum_{\mathcal{S}} p_{\mathcal{S}} q_\emptyset = \sum_{I \in \mathcal{I}} \sum_{\mathcal{S} \text{ s.t. } S_1 = I} p_{\mathcal{S}} q_\emptyset = \sum_{I \in \mathcal{I}} q_I = 1,$$

where $\mathcal{S}$ is an independent set sequence. In other words,

$$\sum_{\mathcal{S}} p_{\mathcal{S}} = \frac{1}{q_\emptyset}, \tag{10}$$

where $\mathcal{S}$ is an independent set sequence. This fact is also observed by Knuth [25, Theorem F] and Harvey and Vondrák [23, Corollary 5.28] in the more general settings. Our proof here gives a combinatorial explanation of this equality.

Equation (10) holds whenever $q_\emptyset > 0$. Recall that $q_\emptyset$ is the shorthand of $q_\emptyset(\mathbf{p})$, which is

$$q_\emptyset(\mathbf{p}) = \sum_{I \in \mathcal{I}} (-1)^{|I|} \prod_{i \in I} p_i, \tag{11}$$

where $\mathcal{I}$ is the collection of independent sets of the dependency graph G.

Lemma 11. *Assume that Condition 5 holds. If $q_\emptyset(\boldsymbol{p}) > 0$, then $q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m) > 0$ for any $i \in [m]$ and $0 \le z \le 1$.*

Proof. By (11),

$$q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m) = \sum_{I \in \mathcal{I},\ i \notin I} (-1)^{|I|} \prod_{j \in I} p_j + z \sum_{I \in \mathcal{I},\ i \in I} (-1)^{|I|} \prod_{j \in I} p_j.$$

Notice that $\sum_{I \in \mathcal{I},\ i \in I} (-1)^{|I|} \prod_{j \in I} p_j = -q_i(\mathbf{p}) < 0$ ($q_i(\mathbf{p})$ is the probability of exactly event A_i occurring). Hence, $q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m) \ge q_\emptyset(\mathbf{p}) > 0$. □

Let T_i be the number of resamplings of event A_i and T be the total number of resampling events. Then $T = \sum_{i=1}^m T_i$.

Lemma 12. *Assume that Condition 5 holds. If $q_\emptyset(\boldsymbol{p}) > 0$, then $\mathbb{E}\, T_i = q_\emptyset(\boldsymbol{p})\left(\frac{1}{q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m)}\right)'|_{z=1}$.*

Proof. By Lemma 11, Equation (10) holds with p_i replaced by $p_i z$ where $z \in [0, 1]$. For a given independent set sequence $\mathcal{S}$, let $T_i(\mathcal{S})$ be the total number of occurrences of A_i in $\mathcal{S}$. Then we have that

$$\sum_{\mathcal{S}} p_{\mathcal{S}} z^{T_i(\mathcal{S})} = \frac{1}{q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m)}. \tag{12}$$

Take derivative with respect to z of (12):

$$\sum_{\mathcal{S}} T_i(\mathcal{S}) p_{\mathcal{S}} z^{T_i(\mathcal{S})-1} = \left(\frac{1}{q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m)}\right)'.$$

Evaluate the preceding equation at $z = 1$:

$$\sum_{\mathcal{S}} T_i(\mathcal{S}) p_{\mathcal{S}} = \left(\frac{1}{q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m)}\right)'\Bigg|_{z=1}. \tag{13}$$

However, we have that

$$\begin{aligned}
\mathbb{E}\, T_i &= \sum_{\mathcal{S}} \Pr_{\mathrm{PRS}}(\text{the log is } \mathcal{S})\, T_i(\mathcal{S}) \\
&= \sum_{\mathcal{S}} p_{\mathcal{S}} q_\emptyset(\mathbf{p}) T_i(\mathcal{S}) && \text{(by Lemma 9)} \\
&= q_\emptyset(\mathbf{p}) \left(\frac{1}{q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m)}\right)'\Bigg|_{z=1}. && \text{(by (13))}
\end{aligned}$$

This completes the proof. □

Theorem 13. *Assume that Condition 5 holds. If $q_\emptyset > 0$, then $\mathbb{E}\, T = \sum_{i=1}^m \frac{q_i}{q_\emptyset}$.*

Proof. Clearly, $\mathbb{E}\, T = \sum_{i=1}^m \mathbb{E}\, T_i$. By Lemma 12, all we need to show is that

$$q_\emptyset(\mathbf{p}) \left(\frac{1}{q_\emptyset(p_1, \ldots, p_i z, \ldots, p_m)}\right)'\Bigg|_{z=1} = \frac{q_i(\mathbf{p})}{q_\emptyset(\mathbf{p})}. \tag{14}$$

This is because

$$\begin{aligned}
q'_\emptyset(p_1, \ldots, p_i z, \ldots, p_m) &= \sum_{i \in J,\ J \in \mathcal{I}} (-1)^{|J|} \prod_{j \in J} p_j \\
&= -q_i(\mathbf{p}),
\end{aligned}$$

and thus

$$\left(\frac{1}{q_\emptyset(p_1,\ldots,p_iz,\ldots,p_m)}\right)' = \frac{-q'_\emptyset(p_1,\ldots,p_iz,\ldots,p_m)}{q_\emptyset(p_1,\ldots,p_iz,\ldots,p_m)^2} = \frac{q_i(\mathbf{p})}{q_\emptyset(p_1,\ldots,p_iz,\ldots,p_m)^2}.$$

It is easy to see that (14) follows as we set $z = 1$ and the theorem is shown. □

The quantity $\sum_{i=1}^{m} \frac{q_i}{q_\emptyset}$ is not always easy to bound. Kolipaka and Szegedy [26] have shown that when the probability vector $\mathbf{p}$ satisfies Shearer's condition with a constant "slack," the running time is in fact linear in the number of events in the more general setting. We rewrite it in our notations.

Theorem 14 ([26, Theorem 5]). *Let $d \geq 2$ be a positive integer and $p_c = \frac{(d-1)^{(d-1)}}{d^d}$. Let $p = \max_{i\in[m]}\{p_i\}$. If G has maximum degree d and $p < p_c$, then $\mathbb{E}\,T \leq \frac{p}{p_c - p} \cdot m$.*

4 APPLICATIONS OF ALGORITHM 2

4.1 Sink-Free Orientations

The goal of this application is to sample a sink-free orientation. Given a graph $G = (V, E)$, an orientation of edges is a mapping σ so that $\sigma(e) = (u, v)$ or (v, u) where $e = (u, v) \in E$. A *sink* under orientation σ is a vertex v so that for any adjacent edge $e = (u, v)$, $\sigma(e) = (u, v)$. A sink-free orientation is an orientation so that no vertex is a sink.

Name Sampling Sink-Free Orientations
Instance A graph G.
Output A uniform sink-free orientation.

The first algorithm for this problem is given by Bubley and Dyer [6], using Markov chains and path coupling techniques.

In this application, we associate with each edge a random variable, whose possible values are (u, v) or (v, u). For each vertex v, we associate it with a bad event A_v, which happens when v is a sink. Thus, the graph G itself is also the dependency graph. Condition 5 is satisfied, because if a vertex is a sink, then none of its neighbors can be a sink. Thus, we may apply Algorithm 2, which yields Algorithm 3. This is the sink-popping algorithm given by Cohn et al. [7].

ALGORITHM 3: Sample Sink-Free Orientations

(1) Orient each edge independently and uniformly at random.
(2) While there is at least one sink, reorient all edges that are adjacent to a sink.
(3) Output the current assignment.

Let $Z_{\text{sink},0}$ be the number of sink-free orientations, and let $Z_{\text{sink},1}$ be the number of orientations having exactly one sink. Then Theorem 13 specializes into the following.

Theorem 15. *The expected number of resampled sinks in Algorithm 3 is $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}}$.*

It is easy to see that a graph has a sink-free orientation if and only if the graph is *not* a tree. The next theorem gives an explicit bound on $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}}$ when sink-free orientations exist.

Theorem 16. *Let G be a connected graph on n vertices. If G is not a tree, then $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}} \leq n(n-1)$, where $n = |V(G)|$.*

Proof. Consider an orientation of the edges of G with a unique sink at vertex v. We give a systematic procedure for transforming this orientation to a sink-free orientation. Since G is connected and not a tree, there exists an (undirected) path Π in G of the form $v = v_0, v_1, \ldots, v_{\ell-1}, v_\ell = v_k$, where the vertices $v_0, v_1, \ldots, v_{\ell-1}$ are all distinct and $0 \leq k \leq \ell - 2$. In other words, Π is a simple path of length $\ell - 1$ followed by a single edge back to some previously visited vertex. We will choose a canonical path of this form (depending only on G and not on the current orientation) for each start vertex v.

We now proceed as follows. Since v is a sink, the first edge on Π is directed (v_1, v_0). Reverse the orientation of this edge so that it is now oriented (v_0, v_1). This operation destroys the sink at $v = v_0$ but may create a new sink at v_1. If v_1 is not a sink, then halt. Otherwise, reverse the orientation of the second edge of Π from (v_2, v_1) to (v_1, v_2). Continue in this fashion: if we reach v_i and it is not a sink, then halt; otherwise, reverse the orientation of the $(i+1)$th edge from (v_{i+1}, v_i) to (v_i, v_{i+1}). This procedure must terminate with a sink-free graph before we reach v_ℓ. To see this, note that if we reach the vertex $v_{\ell-1}$, then the final edge of Π must be oriented $(v_{\ell-1}, v_\ell)$; otherwise, the procedure would have terminated already at vertex $v_k (= v_\ell)$.

The effect of the preceding procedure is to reverse the orientation of edges on some initial segment $v_0, \ldots, v_i$ of Π. To put the procedure into reverse, we just need to know the identity of the vertex v_i. So our procedure associates at most n orientations having a single sink at vertex v with each sink-free orientation. There are $n(n-1)$ choices for the pair (v, v_i), and hence $n(n-1)$ single-sink orientations associated with each sink-free orientation. This establishes the result. □

Remark. The bound in Theorem 16 is optimal up to a factor of 2. Consider a cycle of length n. Then there are *two* sink-free orientations and $n(n-1)$ single-sink orientations.

Theorem 16 and Theorem 15 together yield an n^2 bound on the expected number of resamplings that occur during a run of Algorithm 3. A cycle of length n is an interesting special case. Consider the number of clockwise-oriented edges during a run of the algorithm. It is easy to check that this number evolves as an unbiased lazy simple random walk on $[0, n]$. Since the walk starts close to $n/2$ with high probability, we know that it will take $\Omega(n^2)$ steps to reach one of the sink-free states (i.e., 0 or n).

However, if G is a regular graph of degree $\Delta \geq 3$, then we get a much better linear bound from Theorem 14. In the case $\Delta = 3$, we have $p_c = 4/27$, $p = 1/8$, and $p/(p_c - p) = 27/5$. Thus, the expected number of resamplings is bounded by $27n/5$. The constant in the bound improves as Δ increases. Conversely, since the expected running time is exact, we can also apply Theorem 14 to give an upper bound of $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}}$ when G is a regular graph.

4.2 Rooted Spanning Trees

Given a graph $G = (V, E)$ with a special vertex r, we want to sample a uniform spanning tree with r as the root.

Name Sampling Rooted Spanning Trees
Instance A graph G with a vertex r.
Output A uniform spanning tree rooted at r.

Of course, any given spanning tree may be rooted at any vertex r, so there is no real difference between rooted and unrooted spanning trees. However, since this approach to sampling generates an oriented tree, it is easier to think of the trees as being rooted at a particular vertex r.

For all vertices other than r, we randomly assign it to point to one of its neighbors. This is the random variable associated with v. We will think of this random variable as an arrow $v \to s(v)$ pointing from v to its successor $s(v)$. The arrows point out an oriented subgraph of G with $n-1$ edges $\{\{v, s(v)\} : v \in V \setminus \{r\}\}$ directed as specified by the arrows. The constraint for this subgraph to be a tree rooted at r is that it contains no directed cycles. Note that there are $2^{|E|-|V|+\kappa(G)}$ (undirected) cycles in G, where $\kappa(G)$ is the number of connected components of G. Hence, we have possibly exponentially many constraints.

Two cycles are dependent if they share at least one vertex. We claim that Condition 5 is satisfied. Suppose a cycle C is present, and $C' \neq C$ is another cycle that shares at least one vertex with C. If C' is also present, then we may start from any vertex $v \in C \cap C'$, and then follow the arrows $v \to v'$. Since both C and C' are present, it must be that $v' \in C \cap C'$ as well. Continuing this argument, we see that $C = C'$. Contradiction!

As Condition 5 is met, we may apply Algorithm 2, yielding Algorithm 4. This is exactly the cycle-popping algorithm by Wilson [39], as described in Propp and Wilson [33].

ALGORITHM 4: Sample Rooted Spanning Trees

(1) Let T be an empty set. For each vertex $v \neq r$, randomly choose a neighbor $u \in \Gamma(v)$ and add an edge (v, u) to T.
(2) While there is at least one cycle in T, remove all edges in all cycles, and for all vertices whose edges are removed, redo step (1).
(3) Output the current set of edges.

Let $Z_{\text{tree},0}$ be the number of possible assignments of arrows to the vertices of G, which yield a (directed) tree with root r, and $Z_{\text{tree},1}$ be the number of assignments that yield a unicyclic subgraph. The next theorem gives an explicit bound on $\frac{Z_{\text{tree},1}}{Z_{\text{tree},0}}$.

Theorem 17. *Suppose G is a connected graph on n vertices, with m edges. Then $\frac{Z_{\text{tree},1}}{Z_{\text{tree},0}} \leq mn$.*

Proof. Consider an assignment of arrows to the vertices of G that forms a unicyclic graph. This unicyclic subgraph has two components: a directed tree with root r and a directed cycle with a number of directed subtrees rooted on the cycle. This is because if we remove the unicyclic component, the rest of the graph has one less edge than vertices and no cycles, which must be a spanning tree.

As G is connected, there must be an edge in G joining the two components; let this edge be $\{v_0, v_1\}$, where v_0 is in the tree component and v_1 in the unicyclic component. Now extend this edge to a path $v_0, v_1, \ldots, v_\ell$, by following arrows until we reach the cycle. Thus, $v_1 \to v_2$, $v_2 \to v_3$, $\ldots$, $v_{\ell-1} \to v_\ell$ are all arrows, and v_ℓ is the first vertex that lies on the cycle. (It may happen that $\ell = 1$.) Let $v_\ell \to v_{\ell+1}$ be the arrow out of v_ℓ. Now reassign the arrows from vertices $v_1, \ldots, v_\ell$; thus, $v_\ell \to v_{\ell-1}$, $\ldots$, $v_2 \to v_1$, $v_1 \to v_0$. Notice that the result is a directed tree rooted at r.

As before, we would like to bound the number of unicyclic subgraphs associated with a given tree by this procedure. We claim that the procedure can be reversed given just two pieces of information, namely the edge $\{v_\ell, v_{\ell+1}\}$ and the vertex v_0. Note that even though the edge $\{v_\ell, v_{\ell+1}\}$ is undirected, we can disambiguate the endpoints, as v_ℓ is the vertex closer to the root r. The vertices $v_{\ell-1}, \ldots, v_1$ are easy to recover, as they are the vertices on the unique path in the tree from v_ℓ to v_0. To recover the unicyclic subgraph, we just need to reassign the arrows for vertices $v_1, \ldots, v_\ell$ as follows: $v_1 \to v_2$, $\ldots$, $v_\ell \to v_{\ell+1}$.

As the procedure can be reversed knowing one edge and one vertex, the number of unicyclic graphs associated with each tree can be at most mn. □

Theorem 17 combined with Theorem 13 yields an mn upper bound on the expected number of "popped cycles" during a run of Algorithm 4.

However, take a cycle of length n. There are n spanning trees with a particular root v, and there are $\Omega(n^3)$ unicyclic graphs (here a cycle has to be of length 2). Thus, the ratio is $\Omega(n^2) = \Omega(mn)$ since $m = n$, matching the bound of Theorem 17. Moreover, it is known that the cycle-popping algorithm may take $\Omega(n^3)$ time, such as on a dumbbell graph [33].

4.3 Extremal CNF Formulas

A classical setting in the study of algorithmic LLL is to find satisfying assignments in k-CNF formulas,[2] when the number of appearances of every variable is bounded by d. Theorem 1 guarantees the existence of a satisfying assignment as long as $d \leq \frac{2^k}{ek} + 1$. However, sampling is apparently harder than searching in this setting. As shown in Bezáková et al. [4, Corollary 30], it is NP-hard to approximately sample satisfying assignments when $d \geq 5 \cdot 2^{k/2}$, even restricted to the special case of monotone formulas.

Meanwhile, sink-free orientations can be recast in terms of CNF formulas. Every vertex in the graph is mapped to a clause, and every edge is a variable. Thus, every variable appears exactly twice, and we require that the two literals of the same variable are always opposite. Interpreting an orientation from u to v as making the literal in the clause corresponding to v false, the "sink-free" requirement is thus "not all literals in a clause are false." Hence, a sink-free orientation is just a satisfying assignment for the corresponding CNF formula.

To apply Algorithm 2, we need to require that the CNF formula satisfies Condition 5. Such formulas are defined as follows.

Definition 18. We call a CNF formula *extremal* if for every two clauses C_i and C_j, if there is a common variable shared by C_i and C_j, then there exists some variable x such that x appears in both C_i and C_j and the two literals are one positive and one negative.

Let $C_1, \ldots, C_m$ be the clauses of a formula φ. Then define the bad event A_i as the set of unsatisfying assignments of clause C_i. For an extremal CNF formula, these bad events satisfy Condition 5. This is because if $A_i \sim A_j$, then by Definition 18, there exists a variable $x \in \mathrm{var}(A_i) \cap \mathrm{var}(A_j)$ such that the unsatisfying assignment of C_i and C_j differ on x. Hence, $A_i \cap A_j = \emptyset$.

In this formulation, if the size of C_i is k, then the corresponding event A_i happens with probability $p_i = \Pr(A_i) = 2^{-k}$, where variables are sampled uniformly and independently.[3] Suppose each variable appears at most d times. Then the maximum degree in the dependency graph is at most $\Delta = (d-1)k$. Note that in Theorem 14, $p_c = \frac{(\Delta-1)^{(\Delta-1)}}{\Delta^\Delta} \geq \frac{1}{e\Delta}$. Thus, if $d \leq \frac{2^k}{ek} + 1$, then $p_i = 2^{-k} < p_c$ and we may apply Theorem 14 to obtain a polynomial time sampling algorithm.

Corollary 19. *For extremal k-CNF formulas where each variable appears in at most d clauses, if $d \leq \frac{2^k}{ek} + 1$, then Algorithm 2 samples satisfying assignments uniformly at random, with $O(m)$ expected resamplings where m is the number of clauses.*

The condition in Corollary 19 essentially matches the condition of Theorem 1. However, if we only require Shearer's condition as in Theorem 2, Algorithm 2 is not necessarily efficient. More

[2] As usual, in the study of the LLL, by "k-CNF" we mean that every clause has exactly size k.

[3] We note that to find a satisfying assignment, it is sometimes beneficial to consider non-uniform distributions. See Gebauer et al. [13].

precisely, let $Z_{\mathrm{CNF},0}$ be the number of satisfying assignments and $Z_{\mathrm{CNF},1}$ be the number of assignments satisfying all but one clause. If we only require Shearer's condition in Theorem 2, then the expected number of resamplings $\frac{Z_{\mathrm{CNF},1}}{Z_{\mathrm{CNF},0}}$ can be exponential, as shown in the next example.

Example 20. Construct an extremal CNF formula $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_{4m}$ as follows. Let $C_1 := x_1$. Then the variable x_1 is pinned to 1 to satisfy C_1. Let $C_2 := \overline{x}_1 \vee y_1 \vee y_2$, $C_3 := \overline{x}_1 \vee y_1 \vee \overline{y}_2$, and $C_4 := \overline{x}_1 \vee \overline{y}_1 \vee y_2$. Then y_1 and y_2 are also pinned to 1 to satisfy all $C_1 - C_4$.

We continue this construction by letting

$$\begin{aligned} C_{4k+1} &:= \overline{y}_{2k-1} \vee \overline{y}_{2k} \vee x_{k+1}, \\ C_{4k+2} &:= \overline{x}_{k+1} \vee y_{2k+1} \vee y_{2k+2}, \\ C_{4k+3} &:= \overline{x}_{k+1} \vee y_{2k+1} \vee \overline{y}_{2k+2}, \\ C_{4k+4} &:= \overline{x}_{k+1} \vee \overline{y}_{2k+1} \vee y_{2k+2}, \end{aligned}$$

for all $1 \le k \le m-1$. It is easy to see by induction that to satisfy all of them, all x_i's and y_i's have to be 1. Moreover, one can verify that this is indeed an extremal formula. Thus, $Z_{\mathrm{CNF},0} = 1$. Moreover, since φ has a satisfying assignment and is extremal, Shearer's condition is satisfied. Note also that φ is not a 3-CNF formula because C_1 contains a single variable.

However, if we are allowed to ignore C_1, then x_1 can be 0. In that case, there are three choices of y_1 and y_2 so that x_2 to be 0 as well. Thus, there are at least 3^m assignments that only violate C_1, where $x_1 = x_2 = \cdots = x_m = 0$. It implies that $Z_{\mathrm{CNF},1} \ge 3^m$. Hence, we see that $\frac{Z_{\mathrm{CNF},1}}{Z_{\mathrm{CNF},0}} \ge 3^m$. Due to Theorem 13, the expected running time of Algorithm 2 on this formula φ is exponential in m.

We will discuss more on sampling satisfying assignments of a k-CNF formula in Section 7.1.

5 GENERAL PARTIAL REJECTION SAMPLING

In this section, we give a general version of Algorithm 2 that can be applied to arbitrary instances in the variable framework, even without Condition 5.

Recall the notation introduced at the beginning of Section 2. Thus, $\{X_1, \ldots, X_n\}$ is a set of random variables, each with its own distribution and range D_i, and $\{A_1, \ldots, A_m\}$ is a set of bad events that depend on X_i's. The dependencies between events are encoded in the dependency graph $G = (V, E)$. As before, we will use the idea of a resampling table. Recall that $\sigma = \sigma_t = \{X_{i,j_{i,t}} \mid 1 \le i \le n\}$ denotes the current assignment of variables at round t (i.e., the elements of the resampling table that are active at time t). Given σ, let $\mathrm{Bad}(\sigma)$ be the set of occurring bad events; that is, $\mathrm{Bad}(\sigma) = \{i \mid \sigma \in A_i\}$. For a subset $S \subset V$, let ∂S be the boundary of S; that is, $\partial S = \{i \mid i \notin S \text{ and } \exists j \in S,\ (i,j) \in E\}$. Moreover, let

$$\mathrm{var}(S) := \bigcup_{i \in S} \mathrm{var}(A_i).$$

Let $\sigma|_S$ be the partial assignment of σ restricted to $\mathrm{var}(S)$. For an event A_i and $S \subseteq V$, we write $A_i \perp \sigma|_S$ if either $\mathrm{var}(A_i) \cap \mathrm{var}(S) = \emptyset$ or there is no way to extend the partial assignment $\sigma|_S$ to all variables so that A_i holds. Otherwise, $A_i \not\perp \sigma|_S$.

Definition 21. A set $S \subseteq V$ is *unblocking* under σ if for every $i \in \partial S$, $A_i \perp \sigma|_S$.

Given σ, our goal is to resample a set of events that is unblocking and contains $\mathrm{Bad}(\sigma)$. Such a set must exist because V is unblocking (∂V is empty) and $\mathrm{Bad}(\sigma) \subseteq V$. However, we want to resample as few events as possible.

Intuitively, we start by setting the resampling set R_0 as the set of bad events $\mathrm{Bad}(\sigma)$. We mark resampling events in rounds, similar to a breadth-first search. Let R_t be the resampling set of round

ALGORITHM 5: Select the Resampling Set $\mathsf{Res}(\sigma)$ Under an Assignment σ

```
R ← Bad(σ) ;                    // R is the set of events that will be resampled.
N ← ∅ ;                         // N is the set of events that will not be resampled.
U ← ∂R \ N;
while U ≠ ∅ do
    for i ∈ U do
        if A_i ⊥̸ σ|_R then
            R ← R ∪ {i};
        else
            N ← N ∪ {i};
        end
    end
    U ← ∂R \ N;
end
return R
```

$t \geq 0$. In round $t+1$, let A_i be an event on the boundary of R_t that has not been marked yet. We mark it "resampling" if the partial assignments on the shared variables of A_i and R_t can be extended so that A_i occurs. Otherwise, we mark it "not resampling." We continue this process until there is no unmarked event left on the boundary of the current R. An event outside of $\Gamma^+(R)$ may be left unmarked at the end of Algorithm 5. Note that once we mark some event "not resampling," it will never be added into the resampling set. This is because R is only grow in size during the algorithm.

In Algorithm 5, we are dynamically updating R during each iteration of going through U. This is potentially beneficial as an event A_i may become incompatible with R after some event A_j is added, where both $i, j \in U$.

We fix *a priori* an arbitrary ordering while choosing $i \in U$ in the "for" loop of Algorithm 5. Then the output of Algorithm 5 is deterministic under σ. Call it $\mathsf{Res}(\sigma)$.

Lemma 22. *Let σ be an assignment. For any $i \in \partial\mathsf{Res}(\sigma)$, $A_i \perp \sigma|_{\mathsf{Res}(\sigma)}$.*

Proof. Since $i \in \partial\mathsf{Res}(\sigma)$, it must have been marked. Moreover, $i \notin \mathsf{Res}(\sigma)$, so it must be marked as "not resampling." Thus, there exists an intermediate set $R \subseteq \mathsf{Res}(\sigma)$ during the execution of Algorithm 5 such that $A_i \perp \sigma|_R$ and $i \in \partial R$. It implies that A_i is disjoint from the partial assignment of σ restricted to $\mathsf{var}(A_i) \cap \mathsf{var}(R)$. However,

$$\mathsf{var}(A_i) \cap \mathsf{var}(R) \subseteq \mathsf{var}(A_i) \cap \mathsf{var}(\mathsf{Res}(\sigma))$$

as $R \subseteq \mathsf{Res}(\sigma)$. We have that $A_i \perp \sigma|_{\mathsf{Res}(\sigma)}$. □

If Condition 5 is met, then $\mathsf{Res}(\sigma) = \mathsf{Bad}(\sigma)$. This is because at the first step, $R = \mathsf{Bad}(\sigma)$. By Condition 5, for any $i \in \partial\mathsf{Bad}(\sigma)$, A_i is disjoint from all A_j's where $j \in \mathsf{Bad}(\sigma)$ and $A_i \sim A_j$. Since A_j occurs under σ, $A_i \perp \sigma|_R$. Algorithm 5 halts in the first iteration. In this case, since the resampling set is just the (independent) set of occurring bad events, the later Algorithm 6 coincides with Algorithm 2.

The key property of $\mathsf{Res}(\sigma)$ is that if we change the assignment outside of $\mathsf{Res}(\sigma)$, then $\mathsf{Res}(\sigma)$ does not change, unless the new assignment introduces a new bad event outside of $\mathsf{Res}(\sigma)$. More formally, we have the following lemma.

Lemma 23. *Let σ be an assignment. Let σ' be another assignment such that* $\mathsf{Bad}(\sigma') \subseteq \mathsf{Res}(\sigma)$ *and such that σ and σ' agree on all variables in* $\mathsf{var}(\mathsf{Res}(\sigma)) = \bigcup_{i \in \mathsf{Res}(\sigma)} \mathsf{var}(A_i)$. *Then,* $\mathsf{Res}(\sigma') = \mathsf{Res}(\sigma)$.

Proof. Let $R_t(\sigma), N_t(\sigma)$ be the intermediate sets R, N, respectively, at time t of the execution of Algorithm 5 under σ. Thus, $R_0(\sigma) = \mathsf{Bad}(\sigma)$ and $R_0(\sigma) \subseteq R_1(\sigma) \subseteq \cdots \subseteq \mathsf{Res}(\sigma)$. Moreover, $N_0(\sigma) \subseteq N_1(\sigma) \subseteq \cdots$. We will show by induction that $R_t(\sigma) = R_t(\sigma')$ and $N_t(\sigma) = N_t(\sigma')$ for any $t \geq 0$.

For the base case of $t = 0$, by the condition of the lemma, for every $i \in \mathsf{Bad}(\sigma) \subseteq \mathsf{Res}(\sigma)$, the assignments σ and σ' agree on $\mathsf{var}(A_i)$, or equivalently $\sigma|_{\mathsf{Res}(\sigma)} = \sigma'|_{\mathsf{Res}(\sigma)}$. Together with $\mathsf{Bad}(\sigma') \subseteq \mathsf{Res}(\sigma)$, it implies that $\mathsf{Bad}(\sigma) = \mathsf{Bad}(\sigma')$ and $R_0(\sigma) = R_0(\sigma')$. Moreover, $N_0(\sigma) = N_0(\sigma') = \emptyset$.

For the induction step $t > 0$, we have that $R_{t-1}(\sigma) = R_{t-1}(\sigma') \subseteq \mathsf{Res}(\sigma)$ and $N_{t-1}(\sigma) = N_{t-1}(\sigma')$. Let $R = R_{t-1}(\sigma) = R_{t-1}(\sigma')$ and $N = N_{t-1}(\sigma) = N_{t-1}(\sigma')$. Then we will go through $U = \partial R \setminus N$, which is the same for both σ and σ'. Moreover, while marking individual events "resampling" or not, it is sufficient to look at only $\sigma|_R = \sigma'|_R$ since $R \subseteq \mathsf{Res}(\sigma)$. Thus, the markings are exactly the same, implying that $R_t(\sigma) = R_t(\sigma') \subseteq \mathsf{Res}(\sigma)$ and $N_t(\sigma) = N_t(\sigma')$. □

ALGORITHM 6: General Partial Rejection Sampling

(1) Draw independent samples of all variables $X_1, \ldots, X_n$ from their respective distributions.
(2) While at least one bad event occurs under the current assignment σ, use Algorithm 5 to find $\mathsf{Res}(\sigma)$. Resample all variables in $\bigcup_{i \in \mathsf{Res}(\sigma)} \mathsf{var}(A_i)$.
(3) When none of the bad events holds, output the current assignment.

To prove the correctness of Algorithm 6, we will only use three properties of $\mathsf{Res}(\sigma)$, which are intuitively summarized as follows:

(1) $\mathsf{Bad}(\sigma) \subseteq \mathsf{Res}(\sigma)$.
(2) For any $i \in \partial\mathsf{Res}(\sigma)$, A_i is disjoint from the partial assignment of σ projected on $\mathsf{var}(A_i) \cap \mathsf{var}(\mathsf{Res}(\sigma))$ (Lemma 22).
(3) If we fix the partial assignment of σ projected on $\mathsf{var}(\mathsf{Res}(\sigma))$, then the output of Algorithm 5 is fixed, unless there are new bad events occurring outside of $\mathsf{Res}(\sigma)$ (Lemma 23).

Similarly to the analysis of Algorithm 2, we call $\mathcal{S} = S_1, \ldots, S_\ell$ the log if S_i is the set of resampling events in step i of Algorithm 6. Note that for Algorithm 6, the log is not necessarily an independent set sequence. In addition, recall that σ_i is the assignment of variables in step i, and $\sigma_t = \sigma_T$ if T is when Algorithm 6 terminates and $t > T$. The following lemma is an analogue of Lemma 7.

Lemma 24. *Given any log $\mathcal{S}$ of length $\ell \geq 1$, $\sigma_{\ell+1}$ has the product distribution conditioned on none of A_i's occurring where $i \notin \Gamma^+(S_\ell)$, namely from $\mu(\cdot \mid \bigwedge_{i \in [m] \setminus \Gamma^+(S_\ell)} \overline{A_i})$.*

Proof. Suppose $i \notin \Gamma^+(S_\ell)$. By construction, S_ℓ contains all occurring bad events of σ_ℓ, and hence A_i does not occur under σ_ℓ. In step ℓ, we only resample variables that are involved in S_ℓ, so $\sigma_{\ell+1}$ and σ_ℓ agree on $\mathsf{var}(A_i)$. Hence, A_i cannot occur under $\sigma_{\ell+1}$. Call an assignment σ *valid* if none of A_i occurs where $i \notin \Gamma^+(S_\ell)$. To show that $\sigma_{\ell+1}$ has the desired conditional product distribution, we will show that the probabilities of getting any two valid assignments σ and σ' are proportional to their probabilities of occurrence under the product distribution $\mu(\cdot)$.

Let M be the resampling table so that the log of Algorithm 6 is $\mathcal{S}$ up to round ℓ, and $\sigma_{\ell+1} = \sigma$. Indeed, since we only care about events up to round $\ell + 1$, we may truncate the table so that $M = \{X_{i,j} \mid 1 \leq i \leq n, \quad 1 \leq j \leq j_{i,\ell+1}\}$. Let $M' = \{X'_{i,j} \mid 1 \leq i \leq n, \quad 1 \leq j \leq j_{i,\ell+1}\}$ be another table

where $X'_{i,j} = X_{i,j}$ if $j < j_{i,\ell+1}$ for any $i \in [n]$, and $\sigma' = (X'_{i,j_{i,\ell+1}} : 1 \le i \le n)$ is a valid assignment. In other words, we only change the last assignment $(X_{i,j_{i,\ell+1}} : 1 \le i \le n)$ to another valid assignment. We will use $\sigma'_t = (X'_{i,j_{i,t}})$ to denote the active elements of the second resampling table at time t; thus, $\sigma' = \sigma'_{\ell+1}$.

The lemma follows if Algorithm 6 running on M' generates the same log $\mathcal{S}$ up to round ℓ, since, if this is the case, then conditioned on the log $\mathcal{S}$, every possible table M where $\sigma_{\ell+1} = \sigma$ is one-to-one correspondence with another table M' where $\sigma'_{\ell+1} = \sigma'$. Hence, the probability of getting σ is proportional to its weight under $\mu(\cdot)$.

Suppose otherwise and the log of running Algorithm 6 on M and M' differ. Let $t_0 \le \ell$ be the first round where resampling changes, by which we mean that $\mathsf{Res}(\sigma_{t_0}) \neq \mathsf{Res}(\sigma'_{t_0})$. By Lemma 23, either $\mathsf{Bad}(\sigma'_{t_0}) \nsubseteq \mathsf{Res}(\sigma_{t_0})$ or $\sigma_{t_0}|_{\mathsf{Res}(\sigma_{t_0})} \neq \sigma'_{t_0}|_{\mathsf{Res}(\sigma_{t_0})}$. In the latter case, there must be a variable X_i with $i \in \mathsf{var}(\mathsf{Res}(\sigma_{t_0}))$ and index $j_{i,\ell+1}$. However, $i \in \mathsf{var}(\mathsf{Res}(\sigma_{t_0}))$ means that X_i is resampled at least once more in the original run on M, and its index goes up to at least $j_{i,\ell+1} + 1$ at round $\ell + 1$. A contradiction. Thus, $\sigma_{t_0}|_{\mathsf{Res}(\sigma_{t_0})} = \sigma'_{t_0}|_{\mathsf{Res}(\sigma_{t_0})}$ and $\mathsf{Bad}(\sigma'_{t_0}) \nsubseteq \mathsf{Res}(\sigma_{t_0})$.

As $\mathsf{Bad}(\sigma'_{t_0}) \nsubseteq \mathsf{Res}(\sigma_{t_0})$, there must be a variable X_{i_0} such that $j_{i_0,t_0} = j_{i_0,\ell+1}$ (otherwise, $X_{i_0,j_{i_0,t_0}} = X'_{i_0,j_{i_0,t_0}}$) and an event A_k such that $i_0 \in \mathsf{var}(A_k)$, $k \in \mathsf{Bad}(\sigma'_{t_0})$ but $k \notin \mathsf{Res}(\sigma_{t_0})$. Suppose first that $\forall i \in \mathsf{var}(A_k)$, $j_{i,t_0} = j_{i,\ell+1}$, which means that all variables of A_k have reached their final values in the M run at time t_0. This implies that $k \notin \Gamma^+(S_t)$ for any $t \ge t_0$ as otherwise some of the variables in $\mathsf{var}(A_k)$ would be resampled at least once after round t_0. In particular, $k \notin \Gamma^+(S_\ell)$. This contradicts with σ' being valid.

Otherwise, there are some variables in $\mathsf{var}(A_k)$ that get resampled after time t_0 in the M run. Let t_1 be the first such time and $Y \subset \mathsf{var}(A_k)$ be the set of variables resampled at round t_1, namely $Y = \mathsf{var}(A_k) \cap \mathsf{var}(\mathsf{Res}(\sigma_{t_1}))$. We have that $\sigma_{t_1}|_Y = \sigma_{t_0}|_Y$ because t_1 is the first time of resampling variables in Y. Moreover, as variables of Y have not reached their final values yet in the M run, $\sigma_{t_0}|_Y = \sigma'_{t_0}|_Y$. Thus, $\sigma_{t_1}|_Y = \sigma'_{t_0}|_Y$.

Assuming $k \in \mathsf{Res}(\sigma_{t_1})$ would contradict the fact that X_{i_0} has reached its final value in the M run. Hence, $k \notin \mathsf{Res}(\sigma_{t_1})$, but nevertheless variables in $Y \subset \mathsf{var}(A_k)$ are resampled. This implies that $k \in \partial\mathsf{Res}(\sigma_{t_1})$. By Lemma 22, $A_k \perp \sigma_{t_1}|_{\mathsf{Res}(\sigma_{t_1})}$. As $\mathsf{var}(A_k)$ cannot be disjoint from $\mathsf{var}(\mathsf{Res}(\sigma_{t_1}))$, this means that A_k is incompatible with the partial assignment of σ_{t_1} restricted to $\mathsf{var}(A_k) \cap \mathsf{var}(\mathsf{Res}(\sigma_{t_1})) = Y$. Equivalently, $A_k \perp \sigma_{t_1}|_Y$. However, we know that $\sigma_{t_1}|_Y = \sigma'_{t_0}|_Y$, so $A_k \perp \sigma'_{t_0}|_Y$, contradicting $k \in \mathsf{Bad}(\sigma'_{t_0})$. □

Theorem 25. *If Algorithm 6 halts, then its output has the product distribution conditioned on none of A_i's occurring.*

Proof. Let a sequence $\mathcal{S}$ of sets of events be the log of any successful run. Then $S_\ell = \emptyset$. By Lemma 24, conditioned on the log $\mathcal{S}$, the output assignment σ is $\mu(\cdot \mid \bigwedge_{i \in [m] \setminus \Gamma^+(S_\ell)} \overline{A_i}) = \mu(\cdot \mid \bigwedge_{i \in [m]} \overline{A_i})$. This is valid for any possible log, and the theorem follows. □

6 RUNNING TIME ANALYSIS OF ALGORITHM 6

Obviously, when there is no assignment avoiding all bad events, then Algorithm 6 will never halt. Thus, we want to assume some conditions to guarantee a desired assignment. However, the optimal condition of Theorem 2 is quite difficult to work under. Instead, in this section, we will be working under the assumption that the asymmetric LLL condition (1) holds. In fact, to make the presentation clean, we will mostly work with the simpler symmetric case.

However, as mentioned in Section 4.3, Bezáková et al. [4, Corollary 30] showed that even under the asymmetric LLL condition (1), sampling can still be NP-hard. We thus in turn look for further conditions to make Algorithm 6 efficient.

Recall that $\mu(\cdot)$ is the product distribution of sampling all variables independently. For two distinct events $A_i \sim A_j$, let R_{ij} be the event that the partial assignments on $\mathrm{var}(A_i) \cap \mathrm{var}(A_j)$ can be extended to an assignment making A_j true. Thus, if $A_i \not\perp \sigma|_S$ for some event set S, then R_{ji} must hold for all $A_j \in S$ and $A_j \sim A_i$. Conversely, it is possible that each individual R_{ji} is true for all $A_j \in R$ and $A_j \sim A_i$, and yet $A_i \perp \sigma|_S$. Also note that R_{ij} is not necessarily the same as R_{ji}. Let $r_{ij} := \mu(R_{ij})$.

Define $p := \max_{i\in[m]} p_i$ and $r := \max_{A_i\sim A_j,\ i\neq j} r_{ij}$. Let Δ be the maximum degree of the dependency graph G. The main result of the section is the following theorem.

Theorem 26. *Let m be the number of events and n be the number of variables. For any $\Delta \geq 2$, if $6ep\Delta^2 \leq 1$ and $3er\Delta \leq 1$, then the expected number of resampled events of Algorithm 6 is $O(m)$.*

Moreover, when these conditions hold, the number of rounds is $O(\log m)$ and the number of variable resamples is $O(n \log m)$, both in expectation and with high probability.

The first condition $6ep\Delta^2 \leq 1$ is stronger than the condition of the symmetric LLL, but this seems necessary since Bezáková et al. [4, Corollary 30] imply that if $p\Delta^2 \geq C$ for some constant C, then the sampling problem is NP-hard. Intuitively, the second condition $3er\Delta \leq 1$ bounds the expansion from bad events to resampling events at every step of Algorithm 6. We will prove Theorem 26 in the rest of the section.

Let $S \subseteq [m]$ be a subset of vertices of the dependency graph G. Recall that $A(S)$ is the event $\bigwedge_{i\in S} A_i$ and $B(S)$ is the event $\bigwedge_{i\in S} \overline{A_i}$. Moreover, S^c is the complement of S, namely $S^c = [m] \setminus S$, and S^e is the "exterior" of S, namely $S^e = [m] \setminus \Gamma^+(S)$.

Lemma 24 implies that if we resample S at some step t of Algorithm 6, then at step $t+1$ the distribution is the product measure μ conditioned on none of the events in the exterior of S holds, namely $\Pr_\mu(\cdot \mid B(S^e))$.

Let E be an event (not necessarily one of A_i) depending on a set $\mathrm{var}(E)$ of variables. Let $\Gamma(E) := \{i \mid i \in [m],\ \mathrm{var}(A_i) \cap \mathrm{var}(E) \neq \emptyset\}$ if E is not one of A_i, and $\Gamma(A_i) := \{j \mid j \in [m],\ j \neq i \text{ and } \mathrm{var}(A_j) \cap \mathrm{var}(A_i) \neq \emptyset\}$ is defined as usual. Let $S \subseteq [m]$ be a subset of vertices of G. The next lemma bounds the probability of E conditioned on none of the events in S happening. It was first observed in Haeupler et al. [19]. We include a proof for completeness (which is a simple adaptation of the ordinary local lemma proof).

Lemma 27 ([19, Theorem 2.1]). *Suppose (1) holds. For an event E and any set $S \subseteq [m]$,*

$$\Pr_\mu(E \mid B(S)) \leq \Pr_\mu(E) \prod_{i\in\Gamma(E)\cap S} (1 - x_i)^{-1},$$

where x_i's are from (1).

Proof. We prove the inequality by induction on the size of S. The base case is when S is empty and the lemma holds trivially.

For the induction step, let $S_1 = S \cap \Gamma(E)$ and $S_2 = S \setminus S_1$. If $S_1 = \emptyset$, then the lemma holds trivially as E is independent from S in this case. Otherwise, S_2 is a proper subset of S. We have that

$$\begin{aligned}\Pr_\mu(E \mid B(S)) &= \frac{\Pr_\mu(E \wedge B(S_1) \mid B(S_2))}{\Pr_\mu(B(S_1) \mid B(S_2))} \\ &\leq \frac{\Pr_\mu(E \mid B(S_2))}{\Pr_\mu(B(S_1) \mid B(S_2))} \\ &= \frac{\Pr_\mu(E)}{\Pr_\mu(B(S_1) \mid B(S_2))},\end{aligned}$$

where the last line is because E is independent from $B(S_2)$. We then use the induction hypothesis to bound the denominator. Suppose $S_1 = \{j_1, j_2, \ldots, j_r\}$ for some $r > 0$. Then,

$$\begin{aligned}\Pr_\mu(B(S_1) \mid B(S_2)) &= \Pr_\mu\left(\bigwedge_{i\in S_1}\overline{A_i} \;\middle|\; \bigwedge_{i\in S_2}\overline{A_i}\right)\\ &= \prod_{t=1}^{r}\Pr_\mu\left(\overline{A_{j_t}} \;\middle|\; \bigwedge_{s=1}^{t-1}\overline{A_{j_s}} \wedge \bigwedge_{i\in S_2}\overline{A_i}\right)\\ &= \prod_{t=1}^{r}\left(1-\Pr_\mu\left(A_{j_t} \;\middle|\; \bigwedge_{s=1}^{t-1}\overline{A_{j_s}} \wedge \bigwedge_{i\in S_2}\overline{A_i}\right)\right).\end{aligned}$$

By the induction hypothesis and (1), we have that for any $1 \le t \le r$,

$$\begin{aligned}\Pr_\mu\left(A_{j_t} \;\middle|\; \bigwedge_{s=1}^{t-1}\overline{A_{j_s}} \wedge \bigwedge_{i\in S_2}\overline{A_i}\right) &\le \Pr_\mu(A_{j_t}) \prod_{i\in\Gamma(j_t)}(1-x_i)^{-1}\\ &\le x_{j_t}\prod_{i\in\Gamma(j_t)}(1-x_i)\prod_{i\in\Gamma(j_t)}(1-x_i)^{-1}\\ &= x_{j_t}.\end{aligned}$$

Thus,

$$\Pr_\mu(B(S_1) \mid B(S_2)) \ge \prod_{i\in S_1}(1-x_i).$$

The lemma follows. □

Typically, we set $x_i = \frac{1}{\Delta+1}$ in the symmetric setting. Then (1) holds if $ep(\Delta+1) \le 1$. In this setting, Lemma 27 is specialized into the following.

Corollary 28. *If $ep(\Delta+1) \le 1$, then*

$$\Pr_\mu(E \mid B(S)) \le \Pr_\mu(E)\left(1+\frac{1}{\Delta}\right)^{|\Gamma(E)|}.$$

In particular, if $ep(\Delta+1) \le 1$, for any event A_i where $i \notin S$, by Corollary 28,

$$\Pr_\mu(A_i \mid B(S)) \le p_i\left(\frac{\Delta+1}{\Delta}\right)^{\Delta} \le ep. \tag{15}$$

Let Res_t be the resampling set of Algorithm 6 at round $t \ge 1$, and let Bad_t be the set of bad events present at round t. If Algorithm 6 has already stopped at round t, then $\mathsf{Res}_t = \mathsf{Bad}_t = \emptyset$. Furthermore, let $\mathsf{Bad}_0 = \mathsf{Res}_0 = [m]$ since in the first step all random variables are fresh.

Let $C := 1 - p$.

Lemma 29. *For any $\Delta \ge 2$, if $6ep\Delta^2 \le 1$ and $3er\Delta \le 1$, then*

$$\mathbb{E}\left(|\mathsf{Res}_{t+1}| \mid \mathsf{Res}_0, \ldots, \mathsf{Res}_t\right) \le C\,|\mathsf{Res}_t|.$$

Proof. Clearly, for any $\Delta \ge 2$, the condition $6ep\Delta^2 \le 1$ implies that $ep(\Delta+1) \le 1$. Therefore, the prerequisite of Corollary 28 is met. Notice that by Lemma 24, we have that

$$\mathbb{E}\left(|\mathsf{Res}_{t+1}| \mid \mathsf{Res}_0, \ldots, \mathsf{Res}_t\right) = \mathbb{E}\left(|\mathsf{Res}_{t+1}| \mid \mathsf{Res}_t\right).$$

We will show in the following that

$$\mathbb{E}(|\mathrm{Res}_{t+1}||\text{the set of resampling events at round } t \text{ is (exactly) } \mathrm{Res}_t) \le C|\mathrm{Res}_t|,$$

where $C = 1 - p$. This implies the lemma.

Call a path $i_0, i_1, \ldots, i_\ell$ where $\ell \ge 0$ in the dependency graph G *bad* if the following holds:

(1) $i_0 \in \mathrm{Bad}_{t+1}$;
(2) the event $R_{i_{k-1}i_k}$ holds for every $1 \le k \le \ell$;
(3) any i_k ($k \in [\ell]$) is not adjacent to $i_{k'}$ unless $k' = k - 1$ or $k + 1$.

Indeed, paths having the third property are induced paths in G. If $i \in \mathrm{Res}_{t+1}$, A_i must be added by Algorithm 5 during some iteration of the while loop. In the 0th iteration, all of Bad_{t+1} are added. We claim that for any $i \in \mathrm{Res}_{t+1}$ added in $\ell \ge 0$ iteration by Algorithm 5, there exists at least one bad path such that $i_0, i_1, \ldots, i_\ell = i$. We show the claim by an induction on ℓ:

- The base case is that $\ell = 0$, and thus $i \in \mathrm{Bad}_{t+1}$. The bad path is simply i itself.
- For the induction step $\ell \ge 1$, due to Algorithm 5, there must exist $i_{\ell-1}$ adjacent to $i_\ell = i$ such that $i_{\ell-1}$ has been marked "resampling" during iteration $\ell - 1$, and $R_{i_{\ell-1}i_\ell}$ occurs. By the induction hypothesis, there exists a bad path $i_0, \ldots, i_{\ell-1}$. Since i is not marked at iteration $\ell - 1$, i is not adjacent to any vertices that has been marked up to iteration $\ell - 2$. Thus, i_ℓ is not adjacent to any i_k where $k \le \ell - 2$, and the path $i_0, \ldots, i_{\ell-1}, i_\ell$ is bad.

We next turn to bounding the number of bad paths. It is straightforward to bound the size of $\mathrm{Bad}_{t+1} \subseteq \Gamma^+(\mathrm{Res}_t)$. If $i \in \mathrm{Bad}_{t+1}$, then there are two possibilities. The first scenario is that $i \in \mathrm{Res}_t$ and then all of its random variables are fresh. In this case, it occurs with probability $p_i \le p$. Otherwise, $i \in \partial\mathrm{Res}_t$. Recall that by Lemma 24, the distribution at round $t + 1$ is $\Pr_\mu(\cdot \mid B(\mathrm{Res}_t^e))$. By Corollary 28, for any $i \in \partial\mathrm{Res}_t$,

$$\Pr_\mu\left(A_i \mid B\left(\mathrm{Res}_t^e\right)\right) \le p\left(1 + \frac{1}{\Delta}\right)^\Delta \le ep.$$

This implies that

$$\begin{aligned} &\mathbb{E}\left(|\mathrm{Bad}_{t+1}| \mid \text{the set of resampling events at round } t \text{ is (exactly) } \mathrm{Res}_t\right) \\ &\le p\,|\mathrm{Res}_t| + ep\,|\partial\mathrm{Res}_t| \le p(1 + e\Delta)\,|\mathrm{Res}_t|\,. \end{aligned} \tag{16}$$

Next we bound the size of $|\mathrm{Res}_{t+1} \setminus \mathrm{Bad}_{t+1}|$. Let $P = i_0, \ldots, i_\ell$ be an induced path; that is, for any $k \in [\ell]$, i_k is not adjacent to $i_{k'}$ unless $k' = k - 1$ or $k + 1$. Only induced paths are potentially bad. Moreover, P contributes to $|\mathrm{Res}_{t+1} \setminus \mathrm{Bad}_{t+1}|$ only if its length $\ell \ge 1$. Let D_P be the event that P is bad. In other words, $D_P := A_{i_0} \wedge R_{i_0 i_1} \wedge \ldots \wedge R_{i_{\ell-1}i_\ell}$. By Lemma 24, we have that

$$\begin{aligned} &\Pr(P \text{ is bad at round } t + 1 \mid \text{the set of resampling events at round } t \text{ is } \mathrm{Res}_t) \\ &= \Pr_\mu(D_P \mid B(\mathrm{Res}_t^e)), \end{aligned} \tag{17}$$

where we recall that we denote $\mathrm{Res}_t^e = [m] \setminus \Gamma^+(\mathrm{Res}_t)$. Applying Corollary 28 with $S = \mathrm{Res}_t^e$, we have that

$$\Pr_\mu(D_P \mid B(\mathrm{Res}_t^e)) \le \Pr_\mu(D_P)\left(1 + \frac{1}{\Delta}\right)^{|\Gamma(D_P)|}. \tag{18}$$

Note that $\Gamma(R_{i_k i_{k+1}}) \subseteq \Gamma^+(A_{i_k})$ for all $0 \le k \le \ell - 1$. By the definition of D_P,

$$\begin{aligned} \Gamma(D_P) &\subseteq \Gamma(A_{i_0}) \cup \Gamma(R_{i_0 i_1}) \cup \cdots \cup \Gamma(R_{i_{\ell-1}i_\ell}) \\ &\subseteq \Gamma^+(A_{i_0}) \cup \Gamma^+(A_{i_1}) \cup \cdots \cup \Gamma^+(A_{i_{\ell-1}}), \end{aligned}$$

implying that

$$|\Gamma(D_P)| \le \ell(\Delta+1), \tag{19}$$

as $|\Gamma^+(A_k)| \le \Delta+1$ for all $0 \le k \le \ell-1$.

We claim that A_{i_0} is independent from $R_{i_{k-1}i_k}$ for any $2 \le k \le \ell$. This is because i_k is not adjacent to i_0 for any $k \ge 2$, implying that

$$\begin{aligned}\mathsf{var}(R_{i_{k-1}i_k}) \cap \mathsf{var}(A_{i_0}) &= \mathsf{var}(A_{i_{k-1}}) \cap \mathsf{var}(A_{i_k}) \cap \mathsf{var}(A_{i_0}) \\ &\subseteq \mathsf{var}(A_{i_k}) \cap \mathsf{var}(A_{i_0}) = \emptyset.\end{aligned}$$

Moreover, any two events $R_{i_{k-1}i_k}$ and $R_{i_{k'-1}i_{k'}}$ are independent of each other as long as $k < k'$. This is also due to the third property of bad paths. Since $k < k'$, we see that $|k' - (k-1)| \ge 2$ and $i_{k'}$ is not adjacent to i_{k-1}. It implies that

$$\begin{aligned}\mathsf{var}(R_{i_{k-1}i_k}) \cap \mathsf{var}(R_{i_{k'-1}i_{k'}}) &= \mathsf{var}(A_{i_{k-1}}) \cap \mathsf{var}(A_{i_k}) \cap \mathsf{var}(A_{i_{k'-1}}) \cap \mathsf{var}(A_{i_{k'}}) \\ &\subseteq \mathsf{var}(A_{i_{k-1}}) \cap \mathsf{var}(A_{i_{k'}}) = \emptyset.\end{aligned}$$

The consequence of these independences is

$$\begin{aligned}\Pr_\mu(D_P) &\le \Pr_\mu(A_{i_0} \wedge R_{i_1 i_2} \wedge \cdots \wedge R_{i_{\ell-1}i_\ell}) \\ &= \Pr_\mu(A_{i_0}) \prod_{k=2}^{\ell} \Pr_\mu(R_{i_{k-1}i_k}) \\ &\le p r^{\ell-1}.\end{aligned} \tag{20}$$

Note that in the preceding calculation, we ignore $R_{i_0 i_1}$ because it can be positively correlated to A_{i_0}.

Combining (17), (18), (19), and (20), we have that

$$\begin{aligned}&\Pr(D_P \mid \text{the set of resampling events at round } t \text{ is (exactly) } \mathsf{Res}_t) \\ &\le p r^{\ell-1}\left(1+\frac{1}{\Delta}\right)^{\ell(\Delta+1)} \le \frac{p}{r}\left(\left(1+\frac{1}{\Delta}\right)er\right)^{\ell}.\end{aligned} \tag{21}$$

To apply a union bound on all bad paths, we need to bound their number. The first vertex i_0 must be in Bad_{t+1}, implying that $i_0 \in \Gamma^+(\mathsf{Res}_t)$. Hence, there are at most $(\Delta+1)\,|\mathsf{Res}_t|$ choices. Then there are at most Δ choices of i_1 and $(\Delta-1)$ choices of every subsequent i_k where $k \ge 2$. Hence, there are at most $\Delta(\Delta-1)^{\ell-1}$ induced paths of length $\ell \ge 1$, originating from a particular $i_0 \in \Gamma^+(\mathsf{Res}_t)$. Thus, by a union bound on all potentially bad paths and (21),

$$\begin{aligned}&\mathbb{E}(|\mathsf{Res}_{t+1} \setminus \mathsf{Bad}_{t+1}| \mid \text{the set of resampling events at round } t \text{ is (exactly) } \mathsf{Res}_t) \\ &\le \sum_{\ell=1}^{\infty}(\Delta+1)\,|\mathsf{Res}_t|\,\Delta(\Delta-1)^{\ell-1}p/r\left(\left(1+\frac{1}{\Delta}\right)er\right)^{\ell} \\ &= \frac{(\Delta+1)\Delta p}{(\Delta-1)r}\,|\mathsf{Res}_t| \sum_{\ell=1}^{\infty}\left(\left(\frac{\Delta^2-1}{\Delta}\right)er\right)^{\ell} \\ &\le \frac{(\Delta+1)\Delta p}{(\Delta-1)r}\,|\mathsf{Res}_t| \sum_{\ell=1}^{\infty}(er\Delta)^{\ell} = \frac{(\Delta+1)\Delta p}{(\Delta-1)r}\cdot\frac{er\Delta}{1-er\Delta}\,|\mathsf{Res}_t| \\ &\le \frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot ep\Delta^2\,|\mathsf{Res}_t|,\end{aligned} \tag{22}$$

where we use the condition that $er\Delta \le 1/3$.

Combining (16) and (22), we have that

$$\begin{aligned}&\mathbb{E}\left(|\mathrm{Res}_{t+1}| \mid \text{the set of resampling events at round } t \text{ is (exactly) } \mathrm{Res}_t\right)\\ \le\ & \frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot ep\Delta^2\,|\mathrm{Res}_t| + p(1+e\Delta)\,|\mathrm{Res}_t|\\ =\ & p\left(\frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot e\Delta^2 + (1+e\Delta)\right)|\mathrm{Res}_t|\,.\end{aligned}$$

All that is left is to verify that

$$p\left(\frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot e\Delta^2 + (1+e\Delta)\right) \le C,$$

where $C = 1-p$. This is straightforward by the condition $6ep\Delta^2 \le 1$ and $\Delta \ge 2$, as

$$\begin{aligned}C - p\left(\frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot e\Delta^2 + (1+e\Delta)\right) &\ge 6ep\Delta^2 - p - p\left(\frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot e\Delta^2 + (1+e\Delta)\right)\\ &\ge p\left(6e\Delta^2 - 1 - \frac{\Delta+1}{\Delta-1}\cdot\frac{3}{2}\cdot e\Delta^2 - (1+e\Delta)\right) \ge 0.\end{aligned}$$

□

For $t \ge 1$, by Lemma 29 and the law of iterated expectations,

$$\mathbb{E}\,|\mathrm{Res}_t| \le C\,\mathbb{E}\,|R_{t-1}|\,.$$

Thus, $\mathbb{E}\,|\mathrm{Res}_t| \le C^t\,|R_0| = C^t m$. As $C < 1$, the expected number of resampling events is

$$\sum_{t=0}^{\infty}\mathbb{E}\,|\mathrm{Res}_t| \le \sum_{t=0}^{\infty} C^t m = \frac{1}{1-C}\cdot m.$$

This implies the first part of Theorem 26. For the second part, just observe that after $O(\log m)$ rounds, the expected number of bad events is less than m^{-c} for any constant c, and Markov inequality applies.

The first condition of Theorem 26 requires p to be roughly $O(\Delta^{-2})$. This is necessary, due to the hardness result in Bezáková et al. [4] (see also Theorem 33). In addition, in the analysis, it is possible to always add all of $\partial\mathrm{Bad}_t$ into Res_t. Consider a monotone CNF formula. If a clause is unsatisfied, then all of its neighbors need to be added into the resampling set. Such behaviors would eventually lead to the $O(\Delta^{-2})$ bound. This situation is in contrast to the resampling algorithm of Moser and Tardos [31], which only requires $p = O(\Delta^{-1})$ as in the symmetric LLL.

We also note that monotone CNF formulas, in which all correlations are positive, seem to be the worst instances for our algorithms. In particular, Algorithm 6 is exponentially slow when the underlying hypergraph of the monotone CNF is a (hyper-)tree. This indicates that our condition on r in Theorem 26 is necessary for Algorithm 6. In contrast, Hermon et al. [24] show that on a linear hypergraph (including the hypertree), the Markov chain mixes rapidly for degrees higher than the general bound. It is unclear how to combine the advantages from these two approaches.

7 APPLICATIONS OF ALGORITHM 6

7.1 k-CNF Formulas

Consider a k-CNF formula where every variable appears in at most d clauses. Then Theorem 1 says that if $d \le 2^k/(ek) + 1$, then there exists a satisfying assignment. However, as mentioned in Section 4.3, Bezáková et al. [4, Corollary 30] showed that when $d \ge 5 \cdot 2^{k/2}$, then sampling satisfying assignments is NP-hard, even restricted to monotone formulas.

To apply Algorithm 6 in this setting, we need to bound the parameter r in Theorem 26. A natural way is to lower bound the number of shared variables between any two *dependent* clauses. If this

lower bound is s, then $r = 2^{-s}$ since there is a unique assignment on these s variables that can be extended in such a way as to falsify the clauses.

Definition 30. Let $d \geq 2$ and $s \geq 1$. A k-CNF formula is said to have *degree* d if every variable appears in at most d clauses. Moreover, it has *intersection* s if for any two clauses C_i and C_j that share at least one variable, $|\text{var}(C_i) \cap \text{var}(C_j)| \geq s$.

Note that by the definition, if $k < s$ then the formula is simply isolated clauses. Otherwise, $k \geq s$ and we have that $p_i = p = 2^{-k}$ and $r \leq 2^{-s}$. A simple double counting argument indicates that the maximum degree Δ in the dependency graph satisfies $\Delta \leq \frac{dk}{s}$.

We claim that for integers d and k such that $d \geq 3$ and $dk \geq 2^{3e}$, conditions $d \leq \frac{2^{k/2}}{6e}$ and $s \geq \min\{\log_2 dk, k/2\}$ imply the conditions of Theorem 26, namely $6ep\Delta^2 \leq 1$ and $3er\Delta \leq 1$. In fact, if $s \geq \log_2 dk \geq \log_2 d$, then

$$6ep\Delta^2 \leq 6e2^{-k}\left(\frac{dk}{s}\right)^2 \leq 6e2^{-k}\left(\frac{dk}{\log_2 d}\right)^2 \leq 6e\left(\frac{k}{6e\,(k/2 - \log_2 6e)}\right)^2 < 1,$$

as $\frac{d}{\log_2 d}$ is increasing for any $d \geq 3$. Moreover,

$$3er\Delta \leq \frac{3edk}{2^s s} \leq \frac{3e}{\log_2(dk)} \leq 1.$$

Otherwise, $k/2 \leq s \leq \log_2 dk$, which implies that

$$6ep\Delta^2 \leq 6e2^{-k}\left(\frac{dk}{s}\right)^2 \leq 6e2^{-k}\left(\frac{dk}{k/2}\right)^2 \leq 6e2^{-k}\left(\frac{2^{k/2}}{3e}\right)^2 < 1,$$

and

$$3er\Delta \leq \frac{3edk}{2^s s} \leq \frac{6edk}{k2^{k/2}} \leq 1.$$

Thus, by Theorem 26, we have the following result. Note that resampling a clause involves at most k variables, and for k-CNF formulas with degree d, the number of clauses is linear in the number of variables.

Corollary 31. *For integers d and k such that $d \geq 3$ and $dk \geq 2^{3e}$, if $d \leq \frac{1}{6e} \cdot 2^{k/2}$ and $s \geq \min\{\log_2 dk, k/2\}$, then Algorithm 6 samples satisfying assignments of k-CNF formulas with degree d and intersection s in $O(n)$ time in expectation and in $O(n \log n)$ time with high probability, where n is the number of variables.*

We remark that the lower bound on intersection size s in Corollary 31 does not make the problem trivial. Note that the lower bound $\min\{\log_2 dk, k/2\}$ is at most $k/2$. The "hard" instance in the proof of Bezáková et al. [4, Corollary 30] has roughly $k/2$ shared variables for each pair of dependent clauses. For completeness, we will show that if k is even, and $d \geq 4 \cdot 2^{k/2}$ and $s = k/2$, then the sampling problem is NP-hard. The proof is almost identical to that of Bezáková et al. [4, Corollary 30]. The case of odd k can be similarly handled but with larger constants.

We will use the inapproximability result of Sly and Sun [36] (or equivalently, of Galanis et al. [12]) for the hard-core model. We first remind the reader of the relevant definitions. Let $\lambda > 0$. For a graph $G = (V, E)$, the hard-core model with parameter $\lambda > 0$ is a probability distribution over the set of independent sets of G; each independent set I of G has weight proportional to $\lambda^{|I|}$. The normalizing factor of this distribution is the partition function $Z_G(\lambda)$, formally defined as $Z_G(\lambda) := \sum_I \lambda^{|I|}$ where the sum ranges over all independent sets I of G. The hardness result we

are going to use is about approximating $Z_G(\lambda)$, but it is standard to translate it into the sampling setting as the problem is self-reducible.

Theorem 32 ([12, 36]). *For $d \geq 3$, let $\lambda_c(d) := (d-1)^{d-1}/(d-2)^d$. For all $\lambda > \lambda_c(d)$, it is* NP*-hard to sample an independent set I with probability proportional to $\lambda^{|I|}$ in a d-regular graph.*

Theorem 33. *Let k be an even integer. If $d \geq 4 \cdot 2^{k/2}$ and $s = k/2$, then it is* NP*-hard to sample satisfying assignments of k-CNF formulas with degree d and intersection s uniformly at random.*

Proof. Given a d-regular graph $G = (V, E)$, we will construct a monotone k-CNF formula C with degree d and intersection $k/2$ such that satisfying assignments of C can be mapped to independent sets of G. Replace each vertex $v \in V$ by s variables, say $v_1, \ldots, v_s$. If $(u, v) \in E$, then create a monotone clause $v_1 \vee \cdots \vee v_s \vee u_1 \vee \cdots \vee u_s$. It is easy to see that every variable appears exactly d times since G is d-regular. Moreover, the number of shared variables is always s, and the clause size is $2s = k$.

For each satisfying assignment, we map it to a subset of vertices of G. If all of $v_1, \ldots, v_s$ are *false*, then make v occupied. Otherwise, v is unoccupied. Thus, a satisfying assignment is mapped to an independent set of G. Moreover, there are $(2^{k/2} - 1)^{n-|I|}$ satisfying assignments corresponding to an independent set I, where n is the number of vertices in G. Thus, the weight of I is proportional to $(2^{k/2} - 1)^{-|I|}$, namely $\lambda = (2^{k/2} - 1)^{-1}$ in the hard-core model.

To apply Theorem 32, all we need to do is verify that $\lambda > \lambda_c$, or equivalently

$$2^{k/2} - 1 < \frac{(d-2)^d}{(d-1)^{d-1}}.$$

This can be done as follows:

$$\frac{(d-2)^d}{(d-1)^{d-1}} = (d-2)\left(1 - \frac{1}{d-1}\right)^{d-1} \geq \left(\frac{4}{5}\right)^5 (d-2) > 2^{k/2} - 1. \qquad \square$$

Due to Theorem 33, we see that the dependence between k and d in Corollary 31 is tight in the exponent, even with the further assumption on intersection s.

7.2 Independent Sets

We may also apply Algorithm 6 to sample hard-core configurations with parameter λ. Every vertex is associated with a random variable that is occupied with probability $\frac{\lambda}{1+\lambda}$. In this case, each edge defines a bad event that holds if both endpoints are occupied. Thus, $p = (\frac{\lambda}{1+\lambda})^2$. Algorithm 6 is specialized to Algorithm 7.

ALGORITHM 7: Sample Hard-Core Configurations

(1) Mark each vertex occupied with probability $\frac{\lambda}{1+\lambda}$ independently.
(2) While there is at least one edge with both end points occupied, resample all occupied components of sizes at least 2 and their boundaries.
(3) Output the set of vertices.

To see this, consider a graph $G = (V, E)$ with maximum degree d. Given a configuration $\sigma : V \to \{0, 1\}$, consider the subgraph $G[\sigma]$ of G induced by the vertex subset $\{v \in V : \sigma(v) = 1\}$. Then we

denote by $\mathsf{BadVtx}(\sigma)$ the set of vertices in any component of $G[\sigma]$ of size at least 2. Then the output of Algorithm 5 is

$$\mathsf{ResVtx}(\sigma) := \mathsf{BadVtx}(\sigma) \cup \partial\mathsf{BadVtx}(\sigma).$$

This is because first, all of $\partial\mathsf{BadVtx}(\sigma)$ will be resampled, since any of them has at least one occupied neighbor in $\mathsf{BadVtx}(\sigma)$. Second, $v \in \partial\mathsf{BadVtx}(\sigma)$ is unoccupied (otherwise, $v \in \mathsf{BadVtx}(\sigma)$), and Algorithm 5 stops after adding all of $\partial\mathsf{BadVtx}(\sigma)$. This explains Algorithm 7.

Moreover, let $\mathsf{Bad}(\sigma)$ be the set of edges whose both endpoints are occupied under σ. Let $\mathsf{Res}(\sigma)$ be the set of edges whose both endpoints are in $\mathsf{ResVtx}(\sigma)$. Let σ_t be the random configuration of Algorithm 7 at round t if it has not halted, and $\mathsf{Bad}_t = \mathsf{Bad}(\sigma_t)$, $\mathsf{Res}_t = \mathsf{Res}(\sigma_t)$.

Lemma 34. *If $ep(2d-1) < 1$, then $\mathbb{E}\,|\mathsf{Bad}_{t+1}| \le (4ed^2 - 1)p\,\mathbb{E}\,|\mathsf{Bad}_t|$.*

Proof. First note that the dependency graph is the line graph of G and $\Delta = 2d-2$ is the maximum degree of the line graph of G. Thus, $ep(2d-1) < 1$ guarantees the prerequisite of Corollary 28 is met. It also implies that for any σ, $|\mathsf{Res}(\sigma)| \le (2d-1)\mathsf{Bad}(\sigma)$, and $\partial\,|\mathsf{Res}(\sigma)| \le (2d-2)\,|\mathsf{Res}(\sigma)|$. Similarly to the analysis in Lemma 29, conditioned on a fixed Bad_t, by Corollary 28 (or (15) in particular), we have that

$$\begin{aligned}\mathbb{E}\,|\mathsf{Bad}_{t+1}| &\le p\,|\mathsf{Res}(\sigma)| + ep\,|\partial\mathsf{Res}(\sigma)| \\ &\le (p(2d-1) + ep(2d-2)(2d-1))\,|\mathsf{Bad}_t| \\ &< (4ed^2-1)p\,|\mathsf{Bad}_t|\,.\end{aligned}$$

Since the preceding equality holds for any Bad_t, the lemma follows. □

Lemma 34 implies that if $4epd^2 \le 1$, then the number of bad edges shrinks with a constant factor, and Algorithm 7 resamples $O(m)$ edges in expectation and $O(m \log m)$ edges with high probability, where $m = |E|$. A bounded degree graph is sparse and thus $m = O(n)$, where n is the number of vertices. Since $p = (\frac{\lambda}{1+\lambda})^2$, the condition $4epd^2 \le 1$ is equivalent to

$$\lambda \le \frac{1}{2\sqrt{ed}-1}.$$

Thus, we have the following theorem, where the constants are slightly better than directly applying Theorem 26.

Theorem 35. *If $\lambda \le \frac{1}{2\sqrt{ed}-1}$, then Algorithm 7 draws a uniform hard-core configuration with parameter λ from a graph with maximum degree d in $O(n)$ time in expectation and in $O(n \log n)$ time with high probability, where n is the number of vertices.*

The optimal bound of sampling hard-core configurations is $\lambda < \lambda_c \approx \frac{e}{d}$, where λ_c is defined in Theorem 32. The algorithm is due to Weitz [38], and the hardness is shown in Galanis et al. [12] and Sly and Sun [36]. The condition of our Theorem 35 is more restricted than correlation decay–based algorithms [38] or traditional Markov chain–based algorithms. Nevertheless, our algorithm matches the correct order of magnitude $\lambda = O(d^{-1})$. Moreover, our algorithm has the advantage of being simple, exact, and running in linear time in expectation.

8 DISTRIBUTED ALGORITHMS FOR SAMPLING

An interesting feature of Algorithm 6 is that it is distributed.[4] For concreteness, consider the algorithm applied to sampling hard-core configurations on a graph G (i.e., Algorithm 7), assumed to be

[4]See Feng et al. [11] for a very recent work on distributed sampling algorithms. In particular, they show a similar lower bound in their work [11, Section 5].

of bounded maximum degree. Imagine that each vertex is assigned a processor that has access to a source of random bits. Communication is possible between adjacent processors and is assumed to take constant time. This is essentially the LOCAL model of Linial [27]. Then, in each parallel round of the algorithm, the processor at vertex v can update the value $\sigma(v)$ in constant time, as this requires access only to the values of $\sigma(u)$ for vertices $u \in V(G)$ within a bounded distance r of v. In the case of the hard-core model, we have $r = 2$, since the value $\sigma(v)$ at vertex v should be updated precisely if there are vertices u and u' such that $v \sim u$ and $u \sim u'$ and $\sigma(u) = \sigma(u') = 1$. Note that we allow $u' = v$ here.

In certain applications, including the hard-core model, Algorithm 6 runs in a number of rounds that is bounded by a logarithmic function of the input size with high probability. (Recall Theorem 26.) We show that this is optimal. (Although the argument is presented in the context of the hard-core model, it ought to generalize to many other applications.)

Set $L = \lceil c \log n \rceil$ for some constant $c > 0$ to be chosen later. The instance that establishes the lower bound is a graph G consisting of a collection of n/L disjoint paths $\Pi_1, \ldots, \Pi_{n/L}$ with L vertices each. (Assume that n is an exact multiple of L; this is not a significant restriction.) The high-level idea behind the lower bound is simple and consists of two observations. We assume first that the distributed algorithm we are considering always produces an output, say $\hat{\sigma} : V(G) \to \{0, 1\}$, within t rounds. It will be easy at the end to extend the argument to the situation where the running time is a possibly unbounded random variable with bounded expectation.

Focus attention on a particular path Π with endpoints u and v. The first observation is that if $rt < L/2$, then $\sigma(u)$ (respectively, $\sigma(v)$) depends only on the computations performed by processors in the half of Π containing u (respectively, v). Therefore, in the algorithm's output, $\hat{\sigma}(u)$ and $\hat{\sigma}(v)$ are probabilistically independent. The second observation is that if the constant c is sufficiently small, then in the hard-core distribution, $\sigma(u)$ and $\sigma(v)$ are significantly correlated. Since the algorithm operates independently on each of the n/L paths, these small but significant correlations combine to force to a large variation distance between the hard-core distribution and the output distribution of the algorithm.

We now quantify the second observation. Let $\sigma : V(G) \to \{0, 1\}$ be a sample from the hard-core distribution on a path Π on k vertices with endpoints u and v, and let $I_k = Z_\Pi(\lambda)$ denote the corresponding hard-core partition function (weighted sum over independent sets). Define the matrix $W_k = \left(\begin{smallmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{smallmatrix}\right)$, where $w_{ij} = \Pr(\sigma(u) = i \wedge \sigma(v) = j)$. Then

$$W_k = \frac{1}{I_k} \begin{pmatrix} I_{k-2} & \lambda I_{k-3} \\ \lambda I_{k-3} & \lambda^2 I_{k-4} \end{pmatrix},$$

since I_k is the total weight of independent sets in Π, I_{k-2} is the total weight of independent sets with $\sigma(u) = \sigma(v) = 0$, I_{k-3} is the total weight of independent sets with $\sigma(u) = 0$ and $\sigma(v) = 1$, and so on. Also note that I_k satisfies the recurrence

$$I_0 = 1, \quad I_1 = \lambda + 1, \quad \text{and} \quad I_k = I_{k-1} + \lambda I_{k-2}, \text{ for } k \geq 2. \tag{23}$$

We will use $\det W_k$ to measure the deviation of the distribution of $(\sigma(u), \sigma(v))$ from a product distribution. Write

$$W'_k = \begin{pmatrix} I_{k-2} & I_{k-3} \\ I_{k-3} & I_{k-4} \end{pmatrix},$$

and note that $\det W_k = \lambda^2 I_k^{-2} \det W'_k$. Applying recurrence (23) once to each of the four entries of W'_k, we have

$$\begin{aligned}
\det W'_k &= I_{k-2}I_{k-4} - I_{k-3}^2 \\
&= (I_{k-3} + \lambda I_{k-4})(I_{k-5} + \lambda I_{k-6}) - (I_{k-4} + \lambda I_{k-5})^2 \\
&= I_{k-3}(I_{k-5} + \lambda I_{k-6}) - I_{k-4}(I_{k-4} + \lambda I_{k-5}) + \lambda^2(I_{k-4}I_{k-6} - I_{k-5}^2) \\
&= I_{k-3}I_{k-4} - I_{k-4}I_{k-3} + \lambda^2 \det W'_{k-2} \\
&= \lambda^2 \det W'_{k-2},
\end{aligned}$$

for all $k \geq 6$. By direct calculation, $\det W'_4 = -\lambda^2$ and $\det W'_5 = \lambda^3$. Hence, by induction, $\det W'_k = (-1)^{k-1}\lambda^{k-2}$, and

$$\det W_k = \frac{(-1)^{k-1}\lambda^k}{I_k^2}, \tag{24}$$

for all $k \geq 4$.

Solving the recurrence (23) gives the following formula for I_k:

$$I_k = A_\lambda \left(\frac{1+\sqrt{4\lambda+1}}{2}\right)^k + B_\lambda \left(\frac{1-\sqrt{4\lambda+1}}{2}\right)^k,$$

where

$$A_\lambda = \left(\frac{1}{2} + \frac{2\lambda+1}{2\sqrt{4\lambda+1}}\right) \quad \text{and} \quad B_\lambda = \left(\frac{1}{2} - \frac{2\lambda+1}{2\sqrt{4\lambda+1}}\right).$$

Asymptotically,

$$I_k = (1+o(1))A_\lambda \left(\frac{1+\sqrt{4\lambda+1}}{2}\right)^k.$$

Substituting this estimate into (24) yields $|\det W_k| = (1+o(1))A_\lambda^{-2}\alpha^k$ where

$$\alpha = \frac{2\lambda}{2\lambda + \sqrt{4\lambda+1} + 1}.$$

Note that $0 < \alpha < 1$ and α depends only on λ.

Now let the matrix $\widehat{W}_k = \left(\begin{smallmatrix} \widehat{w}_{00} & \widehat{w}_{01} \\ \widehat{w}_{10} & \widehat{w}_{11} \end{smallmatrix}\right)$ be defined as for W_k, but with respect to the output distribution of the distributed sampling algorithm rather than the true hard-core distribution. Recall that we choose $L = \lceil c \log n \rceil > 2rt$, which implies that $\hat{\sigma}(u)$ and $\hat{\sigma}(v)$ are independent and $\det \widehat{W}_L = 0$. It is easy to check that if $\|\widehat{W}_k - W_k\|_\infty \leq \varepsilon$, where the matrix norm is entrywise, then $|\det W_k| \leq \varepsilon$. Thus, for c sufficiently small (and $L = \lceil c \log n \rceil$), we can ensure that $\|\widehat{W}_L - W_L\|_\infty \geq n^{-1/3}$. Thus, $|\widehat{w}_{ij} - w_{ij}| \geq n^{-1/3}$, for some i, j; for definiteness, suppose $i = j = 0$ and that $\widehat{w}_{00} > w_{00}$.

Let Z (respectively, $\widehat{Z}$) be the number of paths whose endpoints are both assigned 0 in the hard-core distribution (respectively, the algorithm's output distribution). Then Z (respectively, $\widehat{Z}$) is a binomial random variable with expectation $\mu = w_{00}n/L$ (respectively, $\hat{\mu} = \widehat{w}_{00}n/L$). Since $|\mathbb{E}\, Z - \mathbb{E}\, \widehat{Z}| > \Omega(n^{2/3}/\log n)$, a Chernoff bound gives that $\Pr(Z \geq (\mu + \hat{\mu})/2)$ and $\Pr(\widehat{Z} \leq (\mu + \hat{\mu})/2)$ both tend to zero exponentially fast with n. It follows that the variation distance between the distributions of σ and $\hat{\sigma}$ is $1 - o(1)$.

The preceding argument assumes an absolute bound on running time, whereas the running time of an exact sampling algorithm will in general be a random variable T. To bridge the gap, suppose

$\Pr(T \le t) \ge \frac{2}{3}$. Then

$$\begin{aligned}\|\hat{\sigma} - \sigma\|_{\mathrm{TV}} &= \max_A |\Pr(\hat{\sigma} \in A) - \Pr(\sigma \in A)| \\ &= \max_A \Big|\Big(\Pr(\hat{\sigma} \in A \mid T \le t) - \Pr(\sigma \in A)\Big)\Pr(T \le t) \\ &\quad + \Big(\Pr(\hat{\sigma} \in A \mid T > t) - \Pr(\sigma \in A)\Big)\Pr(T > t)\Big| \\ &\ge \tfrac{2}{3}(1 - o(1)) - \tfrac{1}{3} \times 1,\end{aligned}$$

where $\|\cdot\|_{\mathrm{TV}}$ denotes variation distance and A ranges over events $A \subseteq \{0,1\}^{|V(G)|}$. Thus, $\|\sigma - \hat{\sigma}\|_{\mathrm{TV}} \ge \frac{1}{3} - o(1)$, which is a contradiction. It follows that $\Pr(T \le t) < \frac{2}{3}$ and hence $\mathbb{E}(T) \ge \frac{1}{3}t$. Note that this argument places a lower bound on parallel time not just for exact samplers but even for (very) approximate ones.

With only a slight increase in work, one could take the instance G to be a path of length n, which might be considered more natural. Identify $O(n/L)$ subpaths within G, suitably spaced, and of length L. The only complication is that the hard-core distribution does not have independent marginals on distinct subpaths. However, by ensuring that the subpaths are separated by distance n^α, for some small $\alpha > 0$, the correlations can be controlled, and the argument proceeds, with only slight modification, as before.

ACKNOWLEDGMENTS

We would like to thank Yumeng Zhang for pointing out a factor k saving in Corollary 31. We thank Dimitris Achlioptas, Fotis Iliopoulos, Pinyan Lu, Alistair Sinclair, and Yitong Yin for their helpful comments. We also thank anonymous reviewers for their detailed comments.

REFERENCES

[1] Dimitris Achlioptas and Fotis Iliopoulos. 2016. Random walks that find perfect objects and the Lovász local lemma. *J. ACM* 63, 3 (2016), 22.
[2] Noga Alon. 1991. A parallel algorithmic version of the local lemma. *Random Struct. Algorithms* 2, 4 (1991), 367–378.
[3] József Beck. 1991. An algorithmic approach to the Lovász local lemma. I. *Random Struct. Algorithms* 2, 4 (1991), 343–366.
[4] Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, Heng Guo, and Daniel Štefankovič. 2016. Approximation via correlation decay when strong spatial mixing fails. In *Proceedings of ICALP. 45:1–13.*
[5] Magnus Bordewich, Martin E. Dyer, and Marek Karpinski. 2006. Stopping times, metrics and approximate counting. In *Proceedings of ICALP.* 108–119.
[6] Russ Bubley and Martin E. Dyer. 1997. Graph orientations with no sink and an approximation for a hard case of #SAT. In *Proceedings of SODA.* 248–257.
[7] Henry Cohn, Robin Pemantle, and James G. Propp. 2002. Generating a random sink-free orientation in quadratic time. *Electr. J. Comb.* 9, 1 (2002), 13 pages. Research Paper 10.
[8] Artur Czumaj and Christian Scheideler. 2000. Coloring nonuniform hypergraphs: A new algorithmic approach to the general Lovász local lemma. *Random Struct. Algorithms* 17, 3–4 (2000), 213–237.
[9] Paul Erdős and László Lovász. 1975. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and Finite Sets, Volume 10 of Colloquia Mathematica Societatis János Bolyai.* North Holland, 609–628.
[10] Weiming Feng, Yahui Liu, and Yitong Yin. 2018. Local rejection sampling with soft filters. arXiv:1807.06481.
[11] Weiming Feng, Yuxin Sun, and Yitong Yin. 2017. What can be sampled locally? In *Proceedings of PODC.* ACM, New York, NY, 121–130.
[12] Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. 2016. Inapproximability of the partition function for the antiferromagnetic ising and hard-core models. *Comb. Probab. Comput.* 25, 4 (2016), 500–559.
[13] Heidi Gebauer, Tibor Szabó, and Gábor Tardos. 2016. The local lemma is asymptotically tight for SAT. *J. ACM* 63, 5 (2016), 43:1–43:32.
[14] Heng Guo and Kun He. 2018. Tight bounds for popping algorithms. arXiv:1807.01680.
[15] Heng Guo and Mark Jerrum. 2018. Approximately counting bases of bicircular matroids. arXiv:1808.09548.

[16] Heng Guo and Mark Jerrum. 2018. Perfect simulation of the hard disks model by partial rejection sampling. In *ICALP (LIPIcs)*, Vol. 107. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 69:1–69:10.
[17] Heng Guo and Mark Jerrum. 2018. A polynomial-time approximation algorithm for all-terminal network reliability. In *ICALP (LIPIcs)*, Vol. 107. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 68:1–68:12.
[18] Heng Guo, Mark Jerrum, and Jingcheng Liu. 2017. Uniform sampling through the Lovász local lemma. In *Proceedings of STOC*. ACM, New York, NY, 342–355.
[19] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. 2011. New constructive aspects of the Lovász local lemma. *J. ACM* 58, 6 (2011), 28:1–28:28.
[20] David G. Harris and Aravind Srinivasan. 2013. Constraint satisfaction, packet routing, and the Lovász local lemma. In *Proceedings of STOC*. 685–694.
[21] David G. Harris and Aravind Srinivasan. 2013. The Moser-Tardos framework with partial resampling. In *Proceedings of FOCS*. 469–478.
[22] David G. Harris and Aravind Srinivasan. 2014. A constructive algorithm for the Lovász local lemma on permutations. In *Proceedings of SODA*. 907–925.
[23] Nicholas J. A. Harvey and Jan Vondrák. 2015. An algorithmic proof of the Lovász local lemma via resampling oracles. In *Proceedings of FOCS*. 1327–1346.
[24] Jonathan Hermon, Allan Sly, and Yumeng Zhang. 2016. Rapid mixing of hypergraph independent set. arXiv:1610.07999.
[25] Donald E. Knuth. 2015. The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability. Available at http://www-cs-faculty.stanford.edu/~uno/fasc6a.ps.gz.
[26] Kashyap Babu Rao Kolipaka and Mario Szegedy. 2011. Moser and Tardos meet Lovász. In *Proceedings of STOC*. 235–244.
[27] Nathan Linial. 1987. Distributive graph algorithms-global solutions from local data. In *Proceedings of FOCS*. 331–335.
[28] Jingcheng Liu and Pinyan Lu. 2015. FPTAS for counting monotone CNF. In *Proceedings of SODA*. 1531–1548.
[29] Ankur Moitra. 2016. Approximate counting, the Lovász local lemma and inference in graphical models. arXiv:1610.04317.
[30] Michael Molloy and Bruce A. Reed. 1998. Further algorithmic aspects of the local lemma. In *Proceedings of STOC*. 524–529.
[31] Robin A. Moser and Gábor Tardos. 2010. A constructive proof of the general Lovász local lemma. *J. ACM* 57, 2 (2010), Article 11.
[32] James G. Propp and David B. Wilson. 1996. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Struct. Algorithms* 9, 1–2 (1996), 223–252.
[33] James G. Propp and David B. Wilson. 1998. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *J. Algorithms* 27, 2 (1998), 170–217.
[34] Alexander D. Scott and Alan D. Sokal. 2005. The repulsive lattice gas, the independent-set polynomial, and the Lovász local lemma. *J. Stat. Phy.* 118, 5 (2005), 1151–1261.
[35] James B. Shearer. 1985. On a problem of spencer. *Combinatorica* 5, 3 (1985), 241–245.
[36] Allan Sly and Nike Sun. 2014. The computational hardness of counting in two-spin models on d-regular graphs. *Ann. Probab.* 42, 6 (2014), 2383–2416.
[37] Aravind Srinivasan. 2008. Improved algorithmic versions of the Lovász local lemma. In *Proceedings of SODA*. 611–620.
[38] Dror Weitz. 2006. Counting independent sets up to the tree threshold. In *Proceedings of STOC*. 140–149.
[39] David B. Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *Proceedings of STOC*. 296–303.

Received April 2017; revised January 2019; accepted January 2019