



# Uniform Sampling through the Lovász Local Lemma

Heng Guo

Queen Mary, University of London  
School of Mathematical Sciences  
London E1 4NS, UK  
h.guo@qmul.ac.uk

Mark Jerrum

Queen Mary, University of London  
School of Mathematical Sciences  
London E1 4NS, UK  
mj@qmul.ac.uk

Jingcheng Liu

University of California, Berkeley  
Department of EECS  
Berkeley, CA, USA  
liuexp@berkeley.edu

## ABSTRACT

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 (perhaps surprising) 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 by Wilson. 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*;

## KEYWORDS

Exact sampling, Lovász Local Lemma, #SAT

### ACM Reference format:

Heng Guo, Mark Jerrum, and Jingcheng Liu. 2017. Uniform Sampling through the Lovász Local Lemma. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC’17)*, 14 pages.  
DOI: 10.1145/3055399.3055410

## 1 INTRODUCTION

The Lovász Local Lemma [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, 23, 29], the celebrated result by Moser and Tardos [24] gives efficient algorithms to find such a perfect object under conditions that match the Lovász Local Lemma 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 [24] 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. showed [4], sampling can be **NP**-hard even under conditions that are stronger than those of the local lemma. On the one hand, the symmetric Lovász Local Lemma only requires  $ep\Delta \leq 1$ , where  $p$  is the probability of bad events and  $\Delta$  is the maximum degree of the dependency graph. On the other hand, translating the result of [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 2.5) under which we show that the output of the Moser-Tardos algorithm is in fact uniform (see Theorem 2.7). Intuitively, the condition requires any two dependent bad events to be disjoint. Indeed, instances satisfying this condition are called “extremal” in the study of Lovász Local Lemma. 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”,<sup>1</sup> 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 [31] for sampling rooted spanning trees, and the “sink-popping” algorithm by Cohn, Pemantle, and Propp [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 Lovász Local Lemma, 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 2.8), which matches the running time upper bound given by Kolipaka and Szegedy [20] under Shearer’s condition [27].

The next natural question is thus whether we can go beyond extremal instances. Indeed, our main technical contribution is a general uniform sampler (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, in order 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

STOC’17, Montreal, Canada

© 2017 ACM. 978-1-4503-4528-6/17/06...\$15.00

DOI: 10.1145/3055399.3055410

<sup>1</sup>Despite the apparent similarity in names, our algorithm is different from “partial resampling” in [15, 16]. We resample *all* variables in certain sets of events whereas “partial resampling” only resamples a subset of variables from some bad event.

finds all the connected components of size  $\geq 2$ . It marks these vertices and their boundaries (which are unoccupied) as “to be resampled”. Then it resamples all marked vertices, and 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 6.5, 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 mark the set of events to be resampled, denoted by  $\text{Res}$ , iteratively. We start from the set of occurring bad events. Then we mark neighbouring events of the current set  $\text{Res}$ , until there is no event  $A$  on the boundary of  $\text{Res}$  such that the current assignment, projected on the common variables of  $A$  and  $\text{Res}$ , 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 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 4.5).

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 from the desired distribution. Prior to our work, most exact sampling algorithms were obtained by coupling from the past [25]. We also note that previous work on the Moser-Tardos output distribution, such as [14], is not strong enough to guarantee a uniform sample (or  $\epsilon$ -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 5.1). 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 [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 6.2 and Theorem 6.3):

- 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, 21]). Although our result requires an additional lower bound on intersections, not only does it improve the dependency of  $k$  and  $d$  *exponentially*, but 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, e.g., sampling configurations in the hard-core model [11, 28, 30].

Simultaneous to our work, Hermon, Sly, and Zhang [19] 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 [22] 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 while ours is exact. Moreover, ours does not require monotonicity (unlike [19]), and allows larger  $d$  than [22] 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. [19], Markov chains mix rapidly for  $d \leq c2^k/k^2$  when  $s = 1$ .

While the study of algorithmic Lovász Local Lemma has progressed beyond the variable framework [1, 17, 18], 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.

## 2 PARTIAL REJECTION SAMPLING

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

The dependency graph  $G = (V, E)$  has  $m$  vertices, identified with the integers  $\{1, 2, \dots, m\}$ , 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. In other words,  $(i, j) \in E$  if  $\text{var}(A_i) \cap \text{var}(A_j) \neq \emptyset$ . We write  $A_i \sim A_j$  if the vertices  $i$  and  $j$  are adjacent in  $G$ . The asymmetric Lovász Local Lemma [9] states the following.

**THEOREM 2.1.** *If there exist non-negative real numbers  $x_i < 1$  ( $1 \leq i \leq m$ ) such that  $\forall i$ ,*

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

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

Theorem 2.1 has a condition that is easy to verify, but not necessarily optimal. Shearer [27] 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  $1 \leq i \leq m$ . 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  $\mathbf{p} = (p_1, \dots, p_m)$ . When there is no confusion we also simply write  $q_I$  instead of the more cumbersome  $q_I(\mathbf{p})$ . Note that if  $I \notin \mathcal{I}$ ,  $q_I = 0$ .

**THEOREM 2.2 (SHEARER [27]).** *If  $q_I \geq 0$  for all  $I \subseteq V$ , then  $\Pr\left(\bigwedge_{i=1}^m \overline{A_i}\right) \geq q_\emptyset$ .*

In particular, if the condition holds with  $q_\emptyset > 0$ , then

$$\Pr\left(\bigwedge_{i=1}^m \overline{A_i}\right) > 0.$$

Neither Theorem 2.1 nor Theorem 2.2 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 devoting to an algorithmic version of LLL, culminating in Moser and Tardos [24] with essentially the same condition as in Theorem 2.1. The RESAMPLE algorithm of Moser and Tardos is very simple, described in Algorithm 1.

---

**Algorithm 1** The RESAMPLE algorithm

---

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

In [24], Moser and Tardos showed that Algorithm 1 finds a good assignment very efficiently.

**THEOREM 2.3 (MOSER AND TARDOS [24]).** *Under the condition of Theorem 2.1, the expected number of resampling steps in Algorithm 1 is at most  $\sum_{i=1}^m \frac{x_i}{1-x_i}$ .*

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 [20] showed that up to the condition of Shearer, Algorithm 1 is efficient. To simplify the notation, let  $q_i := q_{\{i\}}$  for  $1 \leq i \leq m$ .

**THEOREM 2.4 (KOLIPAKA AND SZEGEDY [20]).** *If  $q_I \geq 0$  for all  $I \in \mathcal{I}$  and  $q_\emptyset > 0$ , then the expected number of resampling steps in Algorithm 1 is at most  $\sum_{i=1}^m \frac{q_i}{q_\emptyset}$ .*

On the other hand, Wilson's cycle-popping algorithm [31] 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, Pemantle, and Propp [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 2.5.** *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 Lovász Local Lemma. They are the ones that minimize  $\Pr\left(\bigwedge_{i=1}^m \overline{A_i}\right)$  given Shearer's condition (namely  $\Pr\left(\bigwedge_{i=1}^m \overline{A_i}\right) = q_\emptyset$ ). Instances satisfying Condition 2.5 have been named *extremal* [20].

We will show that, given Condition 2.5, the final output of the RESAMPLE algorithm is a sample from a conditional product distribution (Theorem 2.7). Moreover, we will show that under Condition 2.5, the running time upper bound  $\sum_{i=1}^m \frac{q_i}{q_\emptyset}$  given by Kolipaka and Szegedy (Theorem 2.4) is indeed the exact expected running time. See Theorem 2.8.

In fact, when Condition 2.5 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 2.5,  $\text{var}(A_i) \cap \text{var}(A_j) = \emptyset$  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.

We call Algorithm 2 the PARTIAL REJECTION SAMPLING algorithm. This name was coined by Cohn, Pemantle, and Propp [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 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 4, which is a uniform sampler for all instances.

---

**Algorithm 2** PARTIAL REJECTION SAMPLING for extremal instances

---

- (1) Draw independent samples of all variables  $X_1, \dots, X_n$  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  $\bigcup_{i \in I} \text{var}(A_i)$ .
  - (3) Output the current assignment.
- 

In fact, Algorithm 2 is the same as the parallel version of Algorithm 1 by Moser and Tardos [24]. 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 2.5.

In the following analysis, we will use the resampling table idea, which has appeared in both the analysis of Moser and Tardos [24] and Wilson [31]. 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  $\{X_{i,1}, X_{i,2}, \dots\}$ . This forms the *resampling table* where each row represents a variable and there are infinitely many columns. 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  $\sigma_t = \{X_{i,j_{i,t}} \mid 1 \leq i \leq n\}$  is the current assignment of variables at round  $t$ . This  $\sigma_t$  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 above,  $I_t$  is an independent set of  $G$  due to Condition 2.5. Then variables that are involved in any of the events in  $I_t$  are resampled. In other words, if  $\exists \ell \in I_t$  such that  $X_i \in \text{var}(A_\ell)$ , then  $j_{i,t+1} = j_{i,t} + 1$ ; otherwise  $j_{i,t+1} = j_{i,t}$ .

Let  $M$  be a resampling table. Suppose running Algorithm 2 on  $M$  does not terminate up to some integer  $\ell \geq 1$  rounds. Define the *log* of running Algorithm 2 on  $M$  up to round  $\ell$  as the sequence of independent sets  $I_1, I_2, \dots, I_\ell$  created by this run. Thus, for any  $M$  and  $1 \leq t \leq \ell$ ,  $I_{t+1} \subseteq \Gamma^+(I_t)$ .

**LEMMA 2.6.** *Suppose Condition 2.5 holds. Given any log  $\mathcal{S} = S_1, S_2, \dots, S_\ell$  of length  $\ell \geq 1$ ,  $\sigma_{\ell+1}$  is a random sample from the product distribution conditioned on none of the events  $A_i$  occurring, where  $i \notin \Gamma^+(S_\ell)$ .*

**PROOF.** The set of occurring events at round  $\ell$  is  $S_\ell$ . Hence  $\sigma_{\ell+1}$  does not make any of the  $A_i$ 's happen where  $i \notin \Gamma^+(S_\ell)$ . Call an assignment  $\sigma$  *valid* if none of  $A_i$ '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  $\sigma$  and  $\sigma'$  are proportional to their probabilities of occurrence in the product distribution.

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.

The lemma follows if the algorithm running on  $M'$  generates the same log  $\mathcal{S}$ . Since if this is the case, then conditioned on the log  $\mathcal{S}$ , every possible table  $M$  is one-to-one corresponding to another

table  $M'$  where  $\sigma_{\ell+1} = \sigma'$ . Hence the probability of getting  $\sigma$  is proportional to its weight in the product distribution.

Suppose otherwise 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. Since  $X'_{i,j} = X_{i,j}$  if  $j < j_{i,\ell+1}$  for any  $i \in [n]$ , any event  $A$  that occurs in  $\mathcal{S}$  should still occur when running on  $M'$ . (If this is not the case, then  $A$  must depend on values at the final round of the resampling table. When the variables of  $A$  are resampled, the algorithm will attempt to access values beyond the table, a contradiction.) There must be an occurring event, say  $A$ , that happens at  $t_0$  on  $M'$  but not on  $M$ . Moreover, there must be a set of variables in  $\text{var}(A)$  that have values  $(X_{i,j_{i,\ell+1}})$ , as otherwise the two runs should be identical. Let us call this set of variables  $Y$ . Since resampling 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  $Y = \text{var}(A)$ . If the claim does not hold, then  $Z := \text{var}(A) \setminus Y \neq \emptyset$ . 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, thus there must be  $A' \neq A$  causing such a resampling of  $X_j$ . Then  $\text{var}(A) \cap \text{var}(A') \neq \emptyset$ . Consider any variable  $X_k \in \text{var}(A) \cap \text{var}(A')$ . 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  $Z$ . On the other hand, in the  $M'$  run,  $X_k$ 's value causes  $A$  to happen at time  $t_0$ . Hence, there exists an assignment on variables in  $\text{var}(A) \cap \text{var}(A')$  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 2.5,  $A \cap A' = \emptyset$ . Contradiction! Therefore the claim holds.

We argue that the remaining case,  $Y = \text{var}(A)$ , is also not possible. Since  $A$  occurs in the  $M'$  run, we know, by the definition of  $\sigma'$ , 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.  $\square$

We remark that Lemma 2.6 is not true for non-extremal instances (that is, if Condition 2.5 fails). In particular, Lemma 2.6 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 4 we show how to instead achieve uniformity by resampling an “unblocking” set of bad events.

Using Lemma 2.6 it is not hard to see the final output of Algorithm 2 is uniform.

**THEOREM 2.7.** *When Condition 2.5 holds and Algorithm 2 halts, its output is a product distribution conditioned on avoiding all bad events.*

The running time of Algorithm 2 actually matches exactly the upper bound given by Kolipaka and Szegedy [20] (see Theorem 2.4).

We omit the proof here and it can be found in Section 3 of the full version [13].

**THEOREM 2.8.** *If  $q_0 > 0$ , then the expected number of resampling steps in Algorithm 2 is given by  $\sum_{i=1}^m \frac{q_i}{q_0}$ .*

The quantity  $\sum_{i=1}^m \frac{q_i}{q_0}$  has a simple combinatorial explanation under Condition 2.5. Call an assignment is perfect if it avoids all bad events, and near-perfect if it avoids all but one bad events. Then  $\sum_{i=1}^m \frac{q_i}{q_0}$  is the ratio between the number of near-perfect assignments and the number of perfect assignments. However it is not always easy to bound. Kolipaka and Szegedy [20] showed 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. Let  $p = \max_{i \in [m]} \{p_i\}$ .

**COROLLARY 2.9** ([20, THEOREM 5]). *Let  $d \geq 2$  be a positive integer and  $p_c = \frac{(d-1)^{(d-1)}}{d^d}$ . If  $G$  has maximum degree  $d$  and  $p < p_c$ , then Algorithm 2 resamples at most  $\frac{p}{p_c - p} \cdot m$  events in expectation.*

### 3 APPLICATIONS OF ALGORITHM 2

#### 3.1 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  $\sigma$  so that  $\sigma(e) = (u, v)$  or  $(v, u)$  where  $e = (u, v) \in E$ . A *sink* under orientation  $\sigma$  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  $v_i$ , we associate it with a bad event  $A_i$ , which happens when  $v_i$  is a sink. Thus the graph  $G$  itself is also the dependency graph. Condition 2.5 is satisfied. This follows because if a vertex is a sink, then none of its neighbours can be a sink. Thus we may apply Algorithm 2, which yields Algorithm 3. This is the "sink-popping" algorithm given by Cohn, Pemantle, and Propp [7].

---

#### Algorithm 3 Sample Sink-free Orientations

---

- (1) Orient each edge independently and uniformly at random.
  - (2) While there is at least one sink, re-orient 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 2.8 specializes into the following.

**THEOREM 3.1.** *The expected number of resampled sinks in Algorithm 3 is  $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}}$ .*

The next theorem gives an explicit bound on  $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}}$ .

**THEOREM 3.2.** *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  $\Pi$  in  $G$  of the form  $v = v_0, v_1, \dots, v_{\ell-1}, v_\ell = v_k$ , where the vertices  $v_0, v_1, \dots, v_{\ell-1}$  are all distinct and  $0 \leq k \leq \ell-2$ . In other words,  $\Pi$  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  $\Pi$  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  $\Pi$  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  $v_\ell$ . To see this, note that if we reach the vertex  $v_{\ell-1}$  then the final edge of  $\Pi$  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 above procedure is to reverse the orientation of edges on some initial segment  $v_0, \dots, v_i$  of  $\Pi$ . 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.  $\square$

**REMARK.** *The bound in Theorem 3.2 is optimal up to a factor of 2. Consider a cycle of length  $n$ . Then there are 2 sink-free orientations, and  $n(n-1)$  single-sink orientations.*

*Theorem 3.2 and Theorem 3.1 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  $n$ .*

*On the other hand, if  $G$  is a regular graph of degree  $\Delta \geq 3$ , then we get a much better linear bound from Corollary 2.9. In the case  $\Delta = 3$ , we have  $p_c = 4/27$ ,  $p = 1/8$  and  $p/(p_c - p) = 27/5$ . So the expected number of resamplings is bounded by  $27n/5$ . The constant in the bound improves as  $\Delta$  increases. Conversely, since the expected running time is exact, we can also apply Corollary 2.9 to give an upper bound of  $\frac{Z_{\text{sink},1}}{Z_{\text{sink},0}}$  when  $G$  is a regular graph.*

#### 3.2 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 neighbours. This is the random variable associated with  $v$ . We will think of this random variable as an arrow  $v \rightarrow s(v)$  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 2.5 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  $v \rightarrow v'$ . 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 2.5 is met, we may apply Algorithm 2, yielding Algorithm 4. This is exactly the “cycle-popping” algorithm by Wilson [31], as described in [26].

---

**Algorithm 4** Sample Rooted Spanning Trees

---

- (1) Let  $T$  be an empty set. For each vertex  $v \neq r$ , randomly choose a neighbour  $u \in \Gamma(v)$  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  $G$ , that yield a (directed) tree with root  $r$ , and  $Z_{\text{tree},1}$  be the number of assignments that yield a unicyclic subgraph. A 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. The next theorem gives an explicit bound on  $\frac{Z_{\text{tree},1}}{Z_{\text{tree},0}}$ .

**THEOREM 3.3.** *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. As previously observed, this graph has two components. 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  $v_0, v_1, \dots, v_\ell$ , by following arrows until we reach the cycle. Thus,  $v_1 \rightarrow v_2, v_2 \rightarrow v_3, \dots, v_{\ell-1} \rightarrow v_\ell$  are all arrows,

and  $v_\ell$  is the first vertex that lies on the cycle. (It may happen that  $\ell = 1$ .) Let  $v_\ell \rightarrow v_{\ell+1}$  be the arrow out of  $v_\ell$ . Now reassign the arrows from vertices  $v_1, \dots, v_\ell$  thus:  $v_\ell \rightarrow v_{\ell-1}, \dots, v_2 \rightarrow v_1, v_1 \rightarrow v_0$ . 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  $v_\ell$  is the vertex closer to the root  $r$ . The vertices  $v_{\ell-1}, \dots, v_1$  are easy to recover, as they are the vertices on the unique path in the tree from  $v_\ell$  to  $v_0$ . To recover the unicyclic subgraph, we just need to reassign the arrows for vertices  $v_1, \dots, v_\ell$  as follows:  $v_1 \rightarrow v_2, \dots, v_\ell \rightarrow v_{\ell+1}$ .

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  $mn$ .  $\square$

Theorem 3.3 combined with Theorem 2.8 yields an  $mn$  upper bound on the expected number of “popped cycles” during a run of Algorithm 4.

On the other hand, the bound of Theorem 3.3 is tight up to constant factors. For example, take a cycle of length  $n$ . Then 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  $m = n$ .

### 3.3 Extremal CNF formulas

A classical setting in the study of algorithmic Lovász Local Lemma is to find satisfying assignments in  $k$ -CNF formulas<sup>2</sup>, when the number of appearances of every variable is bounded by  $d$ . Theorem 2.1 guarantees the existence of a satisfying assignment as long as  $d \leq \frac{2^k}{ek} + 1$ . On the other hand, sampling is apparently harder than searching in this setting. As shown in [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 2.5. Such formulas are defined as follows.

**Definition 3.4.** 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  $C_1, \dots, C_m$  be the clauses of a formula  $\phi$ . 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 2.5. This is because if  $A_i \sim A_j$ , then by Definition 3.4, there exists a variable

<sup>2</sup>As usual in the study of Lovász Local Lemma, by “ $k$ -CNF” we mean that every clause has exactly size  $k$ .

$x \in \text{var}(A_i) \cap \text{var}(A_j)$  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  $p_i = \Pr(A_i) = 2^{-k}$ . 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 Corollary 2.9,  $p_c = \frac{(\Delta-1)^{(\Delta-1)}}{\Delta^\Delta} \geq \frac{1}{e^\Delta}$ . Thus if  $d \leq \frac{2^k}{ek} + 1$ , then  $p_i = 2^{-k} < p_c$  and we may apply Corollary 2.9 to obtain a polynomial time sampling algorithm.

**COROLLARY 3.5.** *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 3.5 essentially matches the condition of Theorem 2.1. On the other hand, if we only require Shearer's condition as in Theorem 2.2, the algorithm may no longer be efficient. More precisely, let  $Z_{\text{CNF},0}$  be the number of satisfying assignments, and  $Z_{\text{CNF},1}$  be the number of assignments satisfying all but one clause. then the expected number of resamplings  $\frac{Z_{\text{CNF},1}}{Z_{\text{CNF},0}}$  can be exponential, as shown in the next example.

*Example 3.6.* Construct an extremal CNF formula  $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_{4m}$  as follows. Let  $C_1 := x_1$ . Then the variable  $x_1$  is pinned to 1 to satisfy  $C_1$ . Let  $C_2 := \bar{x}_1 \vee y_1 \vee y_2$ ,  $C_3 := \bar{x}_1 \vee y_1 \vee \bar{y}_2$ , and  $C_4 := \bar{x}_1 \vee \bar{y}_1 \vee y_2$ . Then  $y_1$  and  $y_2$  are also pinned to 1 to satisfy all  $C_1 - C_4$ .

We continue this construction by letting

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

for all  $1 \leq k \leq m-1$ . 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  $Z_{\text{CNF},0} = 1$ .

On the other hand, if we are allowed to ignore  $C_1$ , then  $x_1$  can be 0. In that case, there are 3 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  $x_1 = x_2 = \dots = x_m = 0$ . It implies that  $Z_{\text{CNF},1} \geq 3^m$ . Hence we see that  $\frac{Z_{\text{CNF},1}}{Z_{\text{CNF},0}} \geq 3^m$ . Due to Theorem 2.8, the expected running time of Algorithm 2 on this formula  $\varphi$  is exponential in  $m$ .

We will discuss more on sampling satisfying assignments of a  $k$ -CNF formula in Section 6.1.

## 4 GENERAL PARTIAL REJECTION SAMPLING

In this section we give a general version of Algorithm 2 which can be applied to arbitrary instances in the variable framework, even without Condition 2.5.

Recall the notation introduced at the beginning of Section 2. So,  $\{X_1, \dots, X_n\}$  is a set of random variables, each with its own distribution and range  $D_i$ , and  $\{A_1, \dots, A_m\}$  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  $\sigma = \sigma_t = \{X_{i,j,t} \mid 1 \leq i \leq n\}$

denotes the current assignment of variables at round  $t$ , i.e., the elements of the resampling table that are active at time  $t$ . Given  $\sigma$ , let  $\text{Bad}(\sigma)$  be the set of occurring bad events; that is,  $\text{Bad}(\sigma) = \{i \mid \sigma \in A_i\}$ . For a subset  $S \subset V$ , let  $\partial S$  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  $\text{var}(S) := \bigcup_{i \in S} \text{var}(A_i)$ . Let  $\sigma|_S$  (or simply  $\sigma_S$  when there is no confusion) be the partial assignment of  $\sigma$  restricted to  $\text{var}(S)$ . For an event  $A_i$  and  $S \subseteq V$ , we write  $A_i \cap \sigma_S = \emptyset$  if  $\text{var}(A_i) \cap \text{var}(S) = \emptyset$ , or  $A_i$  is disjoint from the partial assignment of  $\sigma$ , when both are restricted to  $\text{var}(A_i) \cap \text{var}(S)$ . Informally,  $A_i$  cannot occur given partial assignment  $\sigma_S$ . Otherwise we write  $A_i \cap \sigma_S \neq \emptyset$ .

**Definition 4.1.** A set  $S \subseteq V$  is *unblocking* under  $\sigma$  if for every  $i \in \partial S$ ,  $A_i \cap \sigma_S = \emptyset$ .

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

---

### Algorithm 5 Select the resampling set $\text{Res}(\sigma)$

---

- (1) Let  $R = \text{Bad}(\sigma)$ , which is the set of events that will be resampled. Let  $N = \emptyset$ , which is the set of events that will not be resampled.
  - (2) While  $\partial R \setminus N \neq \emptyset$ , go through  $i \in \partial R \setminus N$ ; if  $A_i \cap \sigma_R \neq \emptyset$ , add  $i$  into  $R$ , otherwise add  $i$  into  $N$ .
  - (3) Output  $R$ .
- 

Intuitively, we start by setting the resampling set  $R_0$  as the set of bad events  $\text{Bad}(\sigma)$ . We mark resampling events in rounds, similar to a breadth first search. Let  $R_t$  be the resampling set of round  $t \geq 0$ . In round  $t+1$ , let  $A_i$  be an event on the boundary of  $R_t$  that hasn't 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.

It is easy to see that the output of Algorithm 5 is deterministic under  $\sigma$ . Call it  $\text{Res}(\sigma)$ .

**LEMMA 4.2.** *Let  $\sigma$  be an assignment. For any  $i \in \partial \text{Res}(\sigma)$ ,  $A_i \cap \sigma_{\text{Res}(\sigma)} = \emptyset$ .*

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

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

as  $R \subseteq \text{Res}(\sigma)$ . We have that  $A_i \cap \sigma_{\text{Res}(\sigma)} = \emptyset$ .  $\square$

If Condition 2.5 is met, then  $\text{Res}(\sigma) = \text{Bad}(\sigma)$ . This is because at the first step,  $R = \text{Bad}(\sigma)$ . By Condition 2.5, for any  $i \in \partial \text{Bad}(\sigma)$ ,

$A_i$  is disjoint from all  $A_j$ 's where  $j \in \text{Bad}(\sigma)$  and  $A_i \sim A_j$ . Since  $A_j$  occurs under  $\sigma$ ,  $A_i \cap \sigma_R = \emptyset$ . Algorithm 5 halts at the first step. 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 to  $\text{Res}(\sigma)$  is that if we change the assignment outside of  $\text{Res}(\sigma)$ , then  $\text{Res}(\sigma)$  does not change, unless the new assignment introduces a new bad event outside of  $\text{Res}(\sigma)$ .

**LEMMA 4.3.** *Let  $\sigma$  be an assignment. Let  $\sigma'$  be another assignment such that  $\text{Bad}(\sigma') \subseteq \text{Res}(\sigma)$  and such that  $\sigma$  and  $\sigma'$  agree on all variables in  $\text{var}(\text{Res}(\sigma)) = \bigcup_{i \in \text{Res}(\sigma)} \text{var}(A_i)$ . Then,  $\text{Res}(\sigma') = \text{Res}(\sigma)$ .*

**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  $\sigma$ . Thus  $R_0(\sigma) = \text{Bad}(\sigma)$  and  $R_0(\sigma) \subseteq R_1(\sigma) \subseteq \dots \subseteq \text{Res}(\sigma)$ . Moreover,  $N_0(\sigma) \subseteq N_1(\sigma) \subseteq \dots$ . 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  $i \in \text{Bad}(\sigma) \subseteq \text{Res}(\sigma)$ , the assignments  $\sigma$  and  $\sigma'$  agree on  $\text{var}(A_i)$ ; or equivalently  $\sigma_{\text{Res}(\sigma)} = \sigma'_{\text{Res}(\sigma)}$ . Together with  $\text{Bad}(\sigma') \subseteq \text{Res}(\sigma)$ , it implies that  $\text{Bad}(\sigma) = \text{Bad}(\sigma')$  and  $R_0(\sigma) = R_0(\sigma')$ . Moreover,  $N_0(\sigma) = N_0(\sigma') = \emptyset$ .

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

---

#### Algorithm 6 GENERAL PARTIAL REJECTION SAMPLING

---

- (1) Draw independent samples of all variables  $X_1, \dots, X_n$  from their respective distributions.
  - (2) While at least one bad event occurs under the current assignment  $\sigma$ , use Algorithm 5 to find  $\text{Res}(\sigma)$ . Resample all variables in  $\bigcup_{i \in \text{Res}(\sigma)} \text{var}(A_i)$ .
  - (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  $\text{Res}(\sigma)$ , which are intuitively summarized as follows:

- (1)  $\text{Bad}(\sigma) \subseteq \text{Res}(\sigma)$ ;
- (2) For any  $i \in \partial \text{Res}(\sigma)$ ,  $A_i$  is disjoint from the partial assignment of  $\sigma$  projected on  $\text{var}(A_i) \cap \text{var}(\text{Res}(\sigma))$  (Lemma 4.2);
- (3) If we fix the partial assignment of  $\sigma$  projected on  $\text{var}(\text{Res}(\sigma))$ , then the output of Algorithm 5 is fixed, unless there are new bad events occurring outside of  $\text{Res}(\sigma)$  (Lemma 4.3).

Similarly to the analysis of Algorithm 2, we call  $\mathcal{S} = S_1, \dots, S_\ell$  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. Also, recall that  $\sigma_i$  is the assignment of variables in step  $i$ .

**LEMMA 4.4.** *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  $i \notin \Gamma^+(S_\ell)$ .*

**PROOF.** Suppose  $i \notin \Gamma^+(S_\ell)$ . By construction,  $S_\ell$  contains all occurring bad events of  $\sigma_\ell$ , and hence  $A_i$  does not occur under  $\sigma_\ell$ . In step  $\ell$ , we only resample variables that are involved in  $S_\ell$ , so  $\sigma_{\ell+1}$  and  $\sigma_\ell$  agree on  $\text{var}(A_i)$ . Hence  $A_i$  cannot occur under  $\sigma_{\ell+1}$ . Call an assignment  $\sigma$  *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  $\sigma$  and  $\sigma'$  are proportional to their probabilities of occurrence in the product distribution.

Let  $M$  be the resampling table so that the log of Algorithm 6 is  $\mathcal{S}$  up to round  $\ell$ , 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  $i \in [n]$ , and  $\sigma' = (X'_{i,j_{i,\ell+1}} : 1 \leq i \leq n)$  is a valid assignment. In other words, we only change the last assignment  $(X_{i,j_{i,\ell+1}})$  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  $\ell$ , 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  $\sigma$  is proportional to its weight in the product distribution.

Suppose otherwise and the log of running Algorithm 6 on  $M$  and  $M'$  differ. Let  $t_0 \leq \ell$  be the first round where resampling changes, by which we mean that  $\text{Res}(\sigma_{t_0}) \neq \text{Res}(\sigma'_{t_0})$ . By Lemma 4.3, there must be a variable  $X_i$  such that  $j_{i,t_0} = j_{i,\ell+1}$  (otherwise  $X_{i,j_{i,t_0}} = X'_{i,j_{i,t_0}}$ ) and  $X_i$  is involved in some event of  $\text{Res}(\sigma_{t_0})$  or  $X_i$  is involved in  $\text{Bad}(\sigma'_{t_0}) \setminus \text{Res}(\sigma_{t_0})$ .

- (1) If  $X_i$  is involved in some event in  $\text{Res}(\sigma_{t_0})$ , then  $X_i$  is resampled once more in the original run on  $M$ , and its index goes up to at least  $j_{i,\ell+1} + 1$  at round  $\ell + 1$ . Contradiction!

This in particular implies that restricted to variables of  $\text{Res}(\sigma_{t_0})$ ,  $\sigma$  and  $\sigma'$  should agree; that is,  $\sigma_{t_0}|_{\text{Res}(\sigma_{t_0})} = \sigma'_{t_0}|_{\text{Res}(\sigma_{t_0})}$ .

- (2) Otherwise there exists  $j$  such that  $X_i \in \text{var}(A_j), j \in \text{Bad}(\sigma'_{t_0})$  but  $j \notin \text{Res}(\sigma_{t_0})$ .

Suppose first that  $\forall k \in \text{var}(A_j), j_{k,t_0} = j_{k,\ell+1}$ , which means that all variables of  $A_j$  have reached their final values in the  $M$  run at time  $t_0$ . This implies that  $j \notin \Gamma^+(S_\ell)$  for any  $t \geq t_0$  as otherwise some of the variables in  $\text{var}(A_j)$  would be resampled at least once after round  $t_0$ . In particular,  $j \notin \Gamma^+(S_\ell)$ . This contradicts with  $\sigma'$  being valid.

Otherwise there are some variables in  $\text{var}(A_j)$  that get resampled after time  $t_0$  in the  $M$  run. Let  $t_1$  be the first such time and  $Y \subseteq \text{var}(A_j)$  be the set of variables resampled at round  $t_1$ ; namely,  $Y = \text{var}(A_j) \cap \text{var}(\text{Res}(\sigma_{t_1}))$ . 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  $j \in \text{Res}(\sigma_{t_1})$  would contradict the fact that  $X_i$  has reached its final value in the  $M$  run. Hence  $j \notin \text{Res}(\sigma_{t_1})$ , but nevertheless variables in  $Y \subset \text{var}(A_j)$  are resampled. This implies that  $j \in \partial \text{Res}(\sigma_{t_1})$ . By Lemma 4.2,  $A_j \cap \sigma_{t_1} |_{\text{Res}(\sigma_{t_1})} = \emptyset$ . This means that  $A_j$  is disjoint from the partial assignment of  $\sigma_{t_1}$  restricted to  $\text{var}(A_j) \cap \text{var}(\text{Res}(\sigma_{t_1})) = Y$ . Equivalently,  $A_j \cap \sigma_{t_1} |_Y = \emptyset$ . However we know that  $\sigma_{t_1} |_Y = \sigma'_{t_0} |_Y$ , so  $A_j \cap \sigma'_{t_0} |_Y = \emptyset$ , contradicting  $j \in \text{Bad}(\sigma'_{t_0})$ .  $\square$

**THEOREM 4.5.** *If Algorithm 6 halts, then its output has the product distribution conditioned on none of  $A_i$ 's occurring.*

**PROOF.** Clearly the output of Algorithm 6 avoids all bad events, since otherwise it does not halt.

Let a sequence  $\mathcal{S}$  of sets of events be the log of any successful run. By Lemma 4.4, conditioned on  $\mathcal{S}$ , the output assignment  $\sigma$  has the product distribution conditioned on  $\bigwedge_{i \notin \Gamma^+(\mathcal{S}_t)} \overline{A_i}$ . Since the algorithm terminates we further condition on none of the other bad events occurring either. The resulting output distribution still has the property that valid assignments occur with probability proportional to those in the product distribution. Since the above argument is valid for any possible log, the theorem follows.  $\square$

## 5 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 2.2 is quite difficult to work under. Instead, in this section we will be working under the assumption that Theorem 2.1's condition (1) holds. In fact, to make the presentation clean, we will mostly work with the simpler symmetric case.

However, as mentioned in Section 3.3, [4, Corollary 30] showed that even under the condition (1), sampling can still be **NP-hard**. We thus in turn look for some further condition to make Algorithm 6 efficient.

Let  $\mu(\cdot)$  be 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  $\text{var}(A_i) \cap \text{var}(A_j)$  can be extended to an assignment making  $A_j$  true. Thus, if  $A_j$  is added by Algorithm 5 at round  $t \geq 1$ , then for any event  $A_j$  added in round  $t - 1$  such that  $A_i \sim A_j$ , the event  $R_{ji}$  has to be true. Conversely, suppose  $A_i$  is unmarked and is under examination at round  $t$ . Then even if  $R_{ji}$  is true for all  $j \in \Gamma(i)$  where  $A_j$  is added in round  $t - 1$ ,  $A_i$  is not necessarily marked "resampling" in round  $t$  by Algorithm 5. (It is possible for  $A_i \cap \sigma_R = \emptyset$  even if all the  $R_{ji}$  are true.) 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  $\Delta$  be the maximum degree of the dependency graph  $G$ . The main result of the section is the following theorem.

**THEOREM 5.1.** *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, in this case, the expected number of rounds is  $O(\log m)$  and thus the expected number of variable resamples is  $O(n \log m)$ .*

The first condition  $6ep\Delta^2 \leq 1$  is stronger than the condition of the symmetric Lovász Local Lemma, but this seems necessary since [4, Corollary 30] implies 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 5.1 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 4.4 implies that if we resample  $S$  at some step  $t$  of Algorithm 6, then at step  $t + 1$  the distribution is the product measure  $\mu$  conditioned on none of the events in the exterior of  $S$  holds; namely  $\Pr_\mu(\cdot | B(S^e))$ .

Let  $E$  be an event (not necessarily one of  $A_i$ ) depending on variables in  $\text{var}(E)$ . Let  $\Gamma(E) := \{i \in [m], \text{var}(A_i) \cap \text{var}(E) \neq \emptyset\}$  if  $E$  is not one of  $A_i$ , and  $\Gamma(A_i) := \{j \in [m], j \neq i \text{ and } \text{var}(A_j) \cap \text{var}(A_i) \neq \emptyset\}$  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 [14].

**LEMMA 5.2** ([14, THEOREM 2.1]). *Suppose (1) holds. For an event  $E$  and any set  $S \subseteq [m]$ ,*

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

where  $x_i$ 's are from (1).

Typically we set  $x_i = \frac{1}{\Delta+1}$  in the symmetric setting. Then (1) holds if  $ep(\Delta + 1) \leq 1$ . In this setting, Lemma 5.2 is specialized into the following.

**COROLLARY 5.3.** *If  $ep(\Delta + 1) \leq 1$ , then*

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

In particular, if  $ep(\Delta + 1) \leq 1$ , for any event  $A_i$  where  $i \notin S$ , by Corollary 5.3,

$$\Pr_\mu(A_i | B(S)) \leq p_i \left(\frac{\Delta + 1}{\Delta}\right)^\Delta \leq ep. \quad (2)$$

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

Define a random variable  $Z_t := C^{-t} |\text{Res}_t|$  for  $C := 1 - p < 1$  and  $t \geq 0$ . Note that  $Z_0 = C^0 |[m]| = m$ .

**LEMMA 5.4.** *For any  $\Delta \geq 2$ , if  $6ep\Delta^2 \leq 1$  and  $3er\Delta \leq 1$ , then  $Z_t$  is a supermartingale.*

**PROOF.** Clearly for any  $\Delta \geq 2$ , the condition  $6ep\Delta^2 \leq 1$  implies that  $ep(\Delta + 1) \leq 1$ . Therefore the prerequisite of Corollary 5.3 is met. We will show that  $Z_t$  is a supermartingale with respect to the sequence  $(\text{Res}_t)$  [12, §7.6(3)]. Since  $|Z_t| < \infty$ , and  $Z_t$  is clearly determined by  $\text{Res}_t$ , we just need to show that

$$\mathbb{E}(Z_{t+1} | \text{Res}_0, \dots, \text{Res}_t) \leq Z_t.$$

Notice that, by Lemma 4.4, we have that

$$\mathbb{E}(Z_{t+1} \mid \text{Res}_0, \dots, \text{Res}_t) = \mathbb{E}(Z_{t+1} \mid \text{Res}_t).$$

We will show in the following that conditioned on the set of resampling events at round  $t$  is (exactly)  $\text{Res}_t$ , the expectation of  $|\text{Res}_{t+1}|$  at round  $t+1$  is at most  $C|\text{Res}_t|$ , where  $C = 1-p$ . This implies the lemma.

Let us consider how an event  $i \in [m]$  is added into  $\text{Res}_{t+1}$ . For any  $i \in \text{Bad}_{t+1}$ , it is always resampled. If  $i \in \text{Res}_{t+1} \setminus \text{Bad}_{t+1}$ , then it is added due to Algorithm 5 at some round  $\ell \geq 1$ . We handle the more complicated latter case first.

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

- (1)  $i_0 \in \text{Bad}_{t+1}$ ;
- (2) the event  $R_{i_{k-1}i_k}$  holds for every  $1 \leq k \leq \ell$ ;
- (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  $G$ . We claim that for any  $i \in \text{Res}_{t+1} \setminus \text{Bad}_{t+1}$  added in round  $\ell \geq 1$  by Algorithm 5, there exists at least one bad path  $i_0, i_1, \dots, i_\ell = i$ . We show the claim by an induction on  $\ell$ .

- If  $\ell = 1$ , then  $i = i_1 \in \partial\text{Bad}_{t+1}$  and there must be an  $i_0 \in \text{Bad}_{t+1}$  such that  $(i_0, i_1)$  is an edge. Since  $i$  is marked “resampling” by Algorithm 5,  $A_i \cap \sigma_{\text{Bad}_{t+1}} \neq \emptyset$ . This implies that  $R_{i_0i_1}$  occurs and the claim holds.
- For the induction step  $\ell \geq 2$ , due to Algorithm 5, there must exist  $i_{\ell-1}$  adjacent to  $i_\ell = i$  such that it is marked “resampling” at round  $\ell-1$ , and  $R_{i_{\ell-1}i_\ell}$  occurs. By the induction hypothesis, there exists a bad path  $i_0, \dots, i_{\ell-1}$ . Since  $i$  is not marked at the round  $\ell-1$ ,  $i$  is not adjacent to any vertices that has been marked up to round  $\ell-2$ . Thus  $i_\ell$  is not adjacent to any  $i_k$  where  $k \leq \ell-2$ , and the path  $i_0, \dots, i_{\ell-1}, i_\ell$  is bad.

Let  $P = i_0, \dots, i_\ell$  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. Let  $D_P$  be the event that  $P$  is bad. In other words,  $D_P := A_{i_0} \wedge R_{i_0i_1} \wedge \dots \wedge R_{i_{\ell-1}i_\ell}$ . By Lemma 4.4, we have that

$$\begin{aligned} & \Pr(P \text{ is bad at round } t+1 \mid \\ & \quad \text{the set of resampling events at round } t \text{ is } \text{Res}_t) \\ &= \Pr_\mu(D_P \mid B(\text{Res}_t^e)), \end{aligned} \quad (3)$$

where we recall that we denote  $\text{Res}_t^e = [m] \setminus \Gamma^+(\text{Res}_t)$ . Applying Corollary 5.3 with  $S = \text{Res}_t^e$ , we have that

$$\Pr_\mu(D_P \mid B(\text{Res}_t^e)) \leq \Pr_\mu(D_P) \left(1 + \frac{1}{\Delta}\right)^{|\Gamma(D_P)|}. \quad (4)$$

Note that  $\Gamma(R_{i_0i_1}) \subseteq \Gamma^+(A_{i_0})$ . By the definition of  $D_P$ ,

$$\begin{aligned} \Gamma(D_P) &\subseteq \Gamma(A_{i_0}) \cup \Gamma(R_{i_0i_1}) \cup \dots \cup \Gamma(R_{i_{\ell-1}i_\ell}) \\ &= \Gamma^+(A_{i_0}) \cup \Gamma(R_{i_1i_2}) \cup \dots \cup \Gamma(R_{i_{\ell-1}i_\ell}), \end{aligned}$$

implying that

$$\begin{aligned} |\Gamma(D_P)| &\leq |\Gamma^+(A_{i_0})| + |R_{i_1i_2}| + \dots + |R_{i_{\ell-1}i_\ell}| \\ &\leq \ell(\Delta + 1), \end{aligned} \quad (5)$$

as  $|\Gamma(R_{ij})| \leq |\Gamma^+(A_i)| \leq \Delta + 1$  for any  $(i, j) \in E$ .

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

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

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  $|k' - (k-1)| \geq 2$  and  $i_{k'}$  is not adjacent to  $i_{k-1}$ . It implies that

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

The consequence of these independences is

$$\begin{aligned} \Pr_\mu(D_P) &\leq \Pr_\mu(A_{i_0} \wedge R_{i_1i_2} \wedge \dots \wedge R_{i_{\ell-1}i_\ell}) \\ &= \Pr_\mu(A_{i_0}) \prod_{k=2}^{\ell} \Pr_\mu(R_{i_{k-1}i_k}) \\ &\leq pr^{\ell-1}. \end{aligned} \quad (6)$$

Note that in the calculation above we ignore  $R_{i_0i_1}$  as it can be positively correlated to  $A_{i_0}$ .

Combining (3), (4), (5), and (6), we have that

$$\begin{aligned} & \Pr(D_P \mid \text{the resampling set at round } t \text{ is (exactly) } \text{Res}_t) \\ &\leq pr^{\ell-1} \left(1 + \frac{1}{\Delta}\right)^{\ell(\Delta+1)} \leq \frac{p}{r} \left(\left(1 + \frac{1}{\Delta}\right)er\right)^\ell. \end{aligned} \quad (7)$$

In order to apply a union bound on all bad paths, we need to bound their number. The first vertex  $i_0$  must be in  $\text{Bad}_{t+1}$ , implying that  $i_0 \in \Gamma^+(\text{Res}_t)$ . Hence there are at most  $(\Delta + 1)|\text{Res}_t|$  choices. Then there are at most  $\Delta$  choices of  $i_1$  and  $(\Delta - 1)$  choices of every subsequent  $i_k$  where  $k \geq 2$ . Hence, there are at most  $\Delta(\Delta - 1)^{\ell-1}$  induced paths of length  $\ell \geq 1$ , originating from a particular  $i_0 \in \Gamma^+(\text{Res}_t)$ . Thus, by a union bound on all potentially bad paths and (7),

$$\begin{aligned} & \mathbb{E}(|\text{Res}_{t+1} \setminus \text{Bad}_{t+1}| \mid \\ & \quad \text{the set of resampling events at round } t \text{ is (exactly) } \text{Res}_t) \\ &\leq \sum_{\ell=1}^{\infty} (\Delta + 1)|\text{Res}_t| \Delta(\Delta - 1)^{\ell-1} p/r \left(\left(1 + \frac{1}{\Delta}\right)er\right)^\ell \\ &= \frac{(\Delta + 1)\Delta p}{(\Delta - 1)r} |\text{Res}_t| \sum_{\ell=1}^{\infty} \left(\left(\frac{\Delta^2 - 1}{\Delta}\right)er\right)^\ell \\ &\leq \frac{(\Delta + 1)\Delta p}{(\Delta - 1)r} |\text{Res}_t| \sum_{\ell=1}^{\infty} (er\Delta)^\ell = \frac{(\Delta + 1)\Delta p}{(\Delta - 1)r} \cdot \frac{er\Delta}{1 - er\Delta} |\text{Res}_t| \\ &= \frac{\Delta + 1}{\Delta - 1} \cdot \frac{3}{2} \cdot ep\Delta^2 |\text{Res}_t|, \end{aligned} \quad (8)$$

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

On the other hand, it is straightforward to bound the size of  $\text{Bad}_{t+1} \subseteq \Gamma^+(\text{Res}_t)$ . If  $i \in \text{Bad}_{t+1}$ , then there are two possibilities. The first scenario is that  $i \in \text{Res}_t$  and then all of its random variables are fresh. In this case it occurs with probability  $p_i \leq p$ . Otherwise

$i \in \partial \text{Res}_t$ . Recall that by Lemma 4.4, the distribution at round  $t + 1$  is  $\Pr_\mu(\cdot \mid B(\text{Res}_t^e))$ . By Corollary 5.3, for any  $i \in \partial \text{Res}_t$ ,

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

This implies that

$$\begin{aligned} & \mathbb{E} \left( |\text{Bad}_{t+1}| \mid \right. \\ & \quad \left. \text{the set of resampling events at round } t \text{ is (exactly) } \text{Res}_t \right) \\ & \leq p |\text{Res}_t| + ep |\partial \text{Res}_t| \leq p(1 + e\Delta) |\text{Res}_t|. \end{aligned} \quad (9)$$

Combining (8) and (9), we have that

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

All that is left is to verify that

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

This is straightforward by the condition  $6ep\Delta^2 \leq 1$  and  $\Delta \geq 2$ , as

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

Since  $Z_t$  is a supermartingale by Lemma 5.4, we have, by the tower property of expectations [12, §7.7(16)],

$$\mathbb{E} Z_t \leq \mathbb{E} Z_0 = m.$$

In other words,  $\mathbb{E} |\text{Res}_t| \leq C^t m$ . As  $C < 1$ , the expected number of resampling events is

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

This implies the first part of Theorem 5.1. For the second part, just observe that within  $O(\log m)$  rounds, the expected number of bad events is less than 1.

The first condition of Theorem 5.1 requires  $p$  to be roughly  $O(\Delta^{-2})$ . This is necessary, due to the hardness result in [4] (see also Theorem 6.3). Also, in the analysis, it is possible to always add all of  $\partial \text{Bad}_t$  into  $\text{Res}_t$ . Consider a monotone CNF formula. If a clause is unsatisfied, then all of its neighbours need to be added into the resampling set. Such behaviours would eventually lead to the  $O(\Delta^{-2})$  bound. This situation is in contrast to the resampling algorithm of Moser and Tardos [24], which only requires  $p = O(\Delta^{-1})$  as in the symmetric Lovász Local Lemma.

Also, we 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 5.1 is necessary for Algorithm 6. In contrast, Hermon et al. [19] 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.

## 6 APPLICATIONS OF ALGORITHM 6

### 6.1 $k$ -CNF Formulas

Consider a  $k$ -CNF formula where every variable appears in at most  $d$  clauses. Then Theorem 2.1 says that if  $d \leq 2^k/(ek) + 1$ , then there exists a satisfying assignment. However, [4, Corollary 30] showed that when  $d \geq 5 \cdot 2^{k/2}$ , 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 5.1. 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 6.1.* 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  $r = 2^{-s}$ . A simple double counting argument indicates that the maximum degree  $\Delta$  in the dependency graph satisfies  $\Delta \leq \frac{dk}{s}$ . It is easy to check 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 5.1, namely,  $6ep\Delta^2 \leq 1$  and  $3er\Delta \leq 1$ . Thus by Theorem 5.1 we have the following result.

**COROLLARY 6.2.** *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 expected  $O(n)$  time where  $n$  is the number of variables.*

We remark that the lower bound on  $s$  in Corollary 6.2 is never larger than  $k/2$ . This lower bound on intersections does not make the problem trivial. Indeed, the bad instance in the proof of [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 [4, Corollary 30]. The case of odd  $k$  can be similarly handled but with larger constants.

**THEOREM 6.3.** *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.*

Theorem 6.3 is based on the inapproximability result of Sly and Sun [28] (or equivalently, of Galanis et al. [11]) for the hard-core model. The proof is omitted and can be found in the full version [13].

Due to Theorem 6.3, we see that the dependence between  $k$  and  $d$  in Corollary 6.2 is tight in the exponent, even with the further assumption on intersection  $s$ .

## 6.2 Independent Sets

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

Let the graph be  $G = (V, E)$  with maximum degree  $d$ . Given a configuration  $\sigma : V \rightarrow \{0, 1\}$ , we denote by  $\text{BadVtx}(\sigma)$  the vertices in any occupied component of size at least 2. Then the output of Algorithm 5 is  $\text{ResVtx}(\sigma) := \text{BadVtx}(\sigma) \cup \partial\text{BadVtx}(\sigma)$ . This is because first, all of  $\partial\text{BadVtx}(\sigma)$  would be resampled, since any of them has at least one occupied neighbour in  $\text{BadVtx}(\sigma)$ . Secondly,  $v \in \partial\text{BadVtx}(\sigma)$  is unoccupied (otherwise  $v \in \text{BadVtx}(\sigma)$ ), and Algorithm 5 stops after adding all of  $\partial\text{BadVtx}(\sigma)$ . This explains Algorithm 7.

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

LEMMA 6.4. *If  $4epd^2 - 1 < 1$ , then  $\mathbb{E}|\text{Bad}_{t+1}| \leq (4epd^2 - p) \cdot \mathbb{E}|\text{Bad}_t|$ .*

Lemma 6.4 implies that, if  $4epd^2 \leq 1$ , then the number of bad edges shrinks with a constant factor, and Algorithm 7 resamples  $O(n)$  vertices in expectation. Since  $p = \left(\frac{\lambda}{1+\lambda}\right)^2$ , the condition  $4epd^2 \leq 1$  is equivalent to  $\lambda \leq \frac{1}{2\sqrt{ed}-1}$ . Thus we have the following theorem, where the constants are slightly better than directly applying Theorem 5.1.

THEOREM 6.5. *If  $\lambda \leq \frac{1}{2\sqrt{ed}-1}$ , then Algorithm 7 draws a uniform hard-core configuration with parameter  $\lambda$  from a graph with maximum degree  $d$  in expected  $O(n)$  time.*

The optimal bound of sampling hard-core configurations is  $\lambda < \lambda_c \approx \frac{e}{d}$  where  $\lambda_c := (d-1)^{d-1}/(d-2)^d$ . The algorithm is due to Weitz [30] and the hardness is shown in [11, 28]. The condition of our Theorem 6.5 is more restricted than correlation decay based algorithms [30] 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.

## 7 DISTRIBUTED ALGORITHMS FOR SAMPLING

An interesting feature of Algorithm 6 is that it is distributed.<sup>3</sup> For concreteness, consider the algorithm applied to sampling hard-core configurations on a graph  $G$  (i.e. Algorithm 7), assumed to be 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. 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 in expectation by a logarithmic function of the size of the input. We show that this is optimal. (Although the argument is presented in the context of the hard-core model, it ought to generalise 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  $\Pi_1, \dots, \Pi_{n/L}$  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  $\hat{\sigma} : V(G) \rightarrow \{0, 1\}$ , 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  $\Pi$  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  $\Pi$  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  $\sigma : V(G) \rightarrow \{0, 1\}$  be a sample from the hard-core distribution on a path  $\Pi$  on  $k$  vertices with endpoints  $u$  and  $v$ , and let  $I_k$  denote the corresponding hard-core partition function. Define the matrix  $W_k = \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix}$ , 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  $\Pi$ ,  $I_{k-2}$  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

<sup>3</sup>See [10] for a very recent work by Feng, Sun, and Yin on distributed sampling algorithms. In particular, they show a similar lower bound in [10, Section 5].

so on. Also note that  $I_k$  satisfies the recurrence

$$I_0 = 1, \quad I_1 = \lambda + 1, \quad \text{and} \quad I_k = I_{k-1} + \lambda I_{k-2}, \quad \text{for } k \geq 2. \quad (10)$$

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  $\det W_k = \lambda^2 I_{k-2}^{-2} \det W'_k$ . Applying recurrence (10) 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

$$\det W_k = \frac{(-1)^{k-1} \lambda^k}{I_k^2}, \quad (11)$$

for all  $k \geq 4$ .

Solving the recurrence (10) gives the following asymptotic formula for  $I_k$ :

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

Combined with (11), this yields  $|\det W_k| = \Theta(\alpha^k)$  where

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

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

Now let the matrix  $\widehat{W}_k = \begin{pmatrix} \widehat{w}_{00} & \widehat{w}_{01} \\ \widehat{w}_{10} & \widehat{w}_{11} \end{pmatrix}$  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 that  $i = j = 0$  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  $\sigma$  and  $\hat{\sigma}$  is  $1 - o(1)$ .

The above 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  $\Pr(T \leq t) \geq \frac{2}{3}$ . Then

$$\begin{aligned} \|\hat{\sigma} - \sigma\|_{\text{TV}} &= \max_A |\Pr(\hat{\sigma} \in A) - \Pr(\sigma \in A)| \\ &= \max_A \left( \Pr(\hat{\sigma} \in A \mid T \leq t) - \Pr(\sigma \in A) \right) \Pr(T \leq t) \\ &\quad + \left( \Pr(\hat{\sigma} \in A \mid T > t) - \Pr(\sigma \in A) \right) \Pr(T > t) \\ &\geq \frac{2}{3}(1 - o(1)) - \frac{1}{3} \times 1, \end{aligned}$$

Where  $\|\cdot\|_{\text{TV}}$  denotes variation distance, and  $A$  ranges over events  $A \subseteq \{0, 1\}^{|V(G)|}$ . Thus  $\|\sigma - \hat{\sigma}\|_{\text{TV}} \geq \frac{1}{3} - o(1)$ , which is a contradiction. It follows that  $\Pr(T \leq t) < \frac{2}{3}$  and hence  $\mathbb{E}(T) \geq \frac{1}{3}t$ . 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  $n^\alpha$ , 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 Theorem 6.2. We also thank Dimitris Achlioptas, Fotis Iliopoulos, Pinyan Lu, Alistair Sinclair, and Yitong Yin for their helpful comments.

HG and MJ are supported by the EPSRC grant EP/N004221/1. JL is 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, and HG was supported by a Google research fellowship during the visit.

## 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 *ICALP*, 45:1–13.
- [5] Magnus Bordewich, Martin E. Dyer, and Marek Karpinski. 2006. Stopping Times, Metrics and Approximate Counting. In *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 *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).
- [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. *Infinite and finite sets, volume 10 of Colloquia Mathematica Societatis János Bolyai* (1975), 609–628.
- [10] Weiming Feng, Yuxin Sun, and Yitong Yin. 2017. What can be sampled locally? *CoRR* abs/1702.00142 (2017).
- [11] 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.
- [12] Geoffrey R. Grimmett and David R. Stirzaker. 2001. *Probability and random processes* (third ed.). Oxford University Press, New York. xii+596 pages.

- [13] Heng Guo, Mark Jerrum, and Jingcheng Liu. 2016. Uniform Sampling through the Lovász Local Lemma. *CoRR* abs/1611.01647 (2016).
- [14] 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.
- [15] David G. Harris and Aravind Srinivasan. 2013. Constraint Satisfaction, Packet Routing, and the Lovász Local Lemma. In *STOC*. 685–694.
- [16] David G. Harris and Aravind Srinivasan. 2013. The Moser-Tardos Framework with Partial Resampling. In *FOCS*. 469–478.
- [17] David G. Harris and Aravind Srinivasan. 2014. A constructive algorithm for the Lovász Local Lemma on permutations. In *SODA*. 907–925.
- [18] Nicholas J. A. Harvey and Jan Vondrák. 2015. An Algorithmic Proof of the Lovász Local Lemma via Resampling Oracles. In *FOCS*. 1327–1346. Full version available at abs/1504.02044.
- [19] Jonathan Hermon, Allan Sly, and Yumeng Zhang. 2016. Rapid Mixing of Hypergraph Independent Set. *CoRR* abs/1610.07999 (2016).
- [20] Kashyap Babu Rao Kolipaka and Mario Szegedy. 2011. Moser and Tardos meet Lovász. In *STOC*. 235–244.
- [21] Jingcheng Liu and Pinyan Lu. 2015. FPTAS for Counting Monotone CNF. In *SODA*. 1531–1548.
- [22] Ankur Moitra. 2016. Approximate Counting, the Lovász Local Lemma and Inference in Graphical Models. *CoRR* abs/1610.04317 (2016). *STOC* 2017, to appear.
- [23] Michael Molloy and Bruce A. Reed. 1998. Further Algorithmic Aspects of the Local Lemma. In *STOC*. 524–529.
- [24] Robin A. Moser and Gábor Tardos. 2010. A constructive proof of the general Lovász Local Lemma. *J. ACM* 57, 2 (2010).
- [25] 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.
- [26] 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.
- [27] James B. Shearer. 1985. On a problem of Spencer. *Combinatorica* 5, 3 (1985), 241–245.
- [28] 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.
- [29] Aravind Srinivasan. 2008. Improved algorithmic versions of the Lovász Local Lemma. In *SODA*. 611–620.
- [30] Dror Weitz. 2006. Counting independent sets up to the tree threshold. In *STOC*. 140–149.
- [31] David B. Wilson. 1996. Generating Random Spanning Trees More Quickly than the Cover Time. In *STOC*. 296–303.