Approximate Counting, the Lovász Local Lemma, and Inference in Graphical Models

ANKUR MOITRA, Massachusetts Institute of Technology, USA

In this article, we introduce a new approach to approximate counting in bounded degree systems with higher-order constraints. Our main result is an algorithm to approximately count the number of solutions to a CNF formula Φ when the width is logarithmic in the maximum degree. This closes an exponential gap between the known upper and lower bounds.

Moreover, our algorithm extends straightforwardly to approximate sampling, which shows that under Lovász Local Lemma-like conditions it is not only possible to find a satisfying assignment, it is also possible to generate one approximately uniformly at random from the set of all satisfying assignments. Our approach is a significant departure from earlier techniques in approximate counting, and is based on a framework to bootstrap an oracle for computing marginal probabilities on individual variables. Finally, we give an application of our results to show that it is algorithmically possible to sample from the posterior distribution in an interesting class of graphical models.

CCS Concepts: • **Mathematics of computing** → **Combinatorial algorithms**; Probabilistic algorithms; • **Theory of computation** → Random walks and Markov chains;

Additional Key Words and Phrases: Approximate counting, Lovász local lemma, graphical models

ACM Reference format:
Ankur Moitra. 2019. Approximate Counting, the Lovász Local Lemma, and Inference in Graphical Models. *J. ACM* 66, 2, Article 10 (April 2019), 25 pages.
https://doi.org/10.1145/3268930

1 INTRODUCTION

1.1 Background

In this article, we introduce a new approach to approximate counting in bounded degree systems with higher-order constraints. For example, if we are given a CNF formula Φ with n variables and m clauses with the property that each clause contains between k and $2k$ variables and each variable belongs to at most d clauses, we ask:

Question 1. *How does k need to relate to d for there to be algorithms to estimate the number of satisfying assignments to Φ within a $(1 \pm 1/n^c)$ multiplicative factor?*

This work was supported in part by an NSF CAREER Award CCF-1453261, NSF Large CCF-1565235, a David and Lucile Packard Fellowship and an Alfred P. Sloan Fellowship.
Author's address: A. Moitra, Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA, 02139, USA; email: moitra@mit.edu.

0004-5411/2019/04-ART10 $15.00
https://doi.org/10.1145/3268930

In the case of a monotone CNF formula where no variable appears negated, the problem is equivalent to the following: Suppose we are given a hypergraph on n nodes and m hyperedges with the property that each hyperedge contains between k and $2k$ nodes and each node belongs to at most d hyperedges. How does k need to relate to d to be able to approximately compute the number of independent sets? Here, an independent set is a subset of nodes for which there is no induced hyperedge. Bordewich, Dyer, and Karpinski [5] gave an MCMC algorithm for approximating the number of hypergraph independent sets (equivalently, the number of satisfying assignments in a monotone CNF formula) that succeeds whenever $k \geq d + 2$. Bezáková et al. [4] gave a deterministic algorithm that succeeds whenever $k \geq d \geq 200$ and proved that when $d \geq 5 \cdot 2^{k/2}$ it is NP-hard to approximate the number of hypergraph independent sets even within an exponential factor.

More broadly, there is a rich literature on approximate counting problems. In a seminal work, Weitz [30] gave an algorithm to approximately count in the hardcore model with parameter λ in graphs of degree at most d whenever

$$\lambda \leq \frac{(d-1)^{d-1}}{(d-2)^d}.$$

And in another seminal work, Sly [28] showed a matching hardness result that was later improved in various respects by Sly and Sun [29] and Galanis, Štefankovič, and Vigoda [11]. These results show that approximate counting is algorithmically possible if and only if there is spatial mixing. Moreover, Weitz's result can be thought of as a comparison theorem that spatial mixing holds on a bounded degree graph if and only if it holds on an infinite tree with the same degree bound. There have been a number of attempts to generalize these results to hypergraphs, many of which follow the approach of defining analogues of the self-avoiding walk trees used in Weitz's algorithm [30]. However, what makes hypergraph versions of these problems more challenging is that spatial mixing fails, even on trees. And we can see that there are *exponential* gaps between the upper and lower bounds, since the algorithms above require k to be linear in d while the lower bounds only rule out $k \leq 2 \log d - O(1)$. (Here and throughout this article, we will take log to be base 2.)

We can take another vantage point to study these problems. Bounded degree CNF formulae are also one of the principal objects of study in the Lovász Local Lemma [10], which is a celebrated result in combinatorics that guarantees when $k \geq \log d + \log k + O(1)$ that Φ has at least one satisfying assignment. The original proof of the Lovász Local Lemma was non-constructive and did not yield a polynomial time algorithm for finding such an assignment, even though it was guaranteed to exist. Beck [3] gave an algorithm followed by a parallel version due to Alon [2] that can find a satisfying assignment whenever $k \geq 8 \log d + \log k + O(1)$. And in a celebrated recent result, Moser and Tardos [24] gave an algorithm exactly matching the existential result. This was followed by a number of works giving constructive proofs of various other settings and generalizations of the Lovász Local Lemma [1, 16, 18, 22]. However, these works leave open the following question:

Question 2. *Under the conditions of the Lovász Local Lemma (i.e., when k is logarithmic in d, but perhaps with a larger constant), is it possible to approximately sample from the uniform distribution on satisfying assignments?*

Approximate counting and approximate sampling problems are well known to be related. When the problem is self-reducible, they are in fact algorithmically equivalent [20, 26]. However, in our setting the problem is not self-reducible, because as we fix variables, we could violate the assumption that k is at least logarithmic in d. It is natural to hope that under exactly the same conditions as the Lovász Local Lemma, there is an algorithm for approximate sampling that matches the limits of the existential and now algorithmic results. However, the hardness results of Bezáková et al. [4]

imply that we need at least another factor of two, and that it is *NP*-hard to approximately sample when $k \leq 2 \log d - O(1)$.[1]

In fact, there is another connection between the Lovász Local Lemma and approximate counting. Scott and Sokal [29] showed that given the dependency graph of events in the local lemma, the best lower bound on the probability of an event guaranteed to exist by the Lovász Local Lemma (i.e., the fraction of satisfying assignments) is exactly the solution to some counting problem. Harvey, Srivastava, and Vondrák [17] recently adapted techniques of Weitz to complex polydisks and gave an algorithm for approximately computing this lower bound. This yields a lower bound on the fraction of satisfying assignments; however, the actual number could be exponentially larger.

1.2 Our Results

Our main result is an algorithm to approximately count the number of solutions when k is at least logarithmic in d. In what follows, let c, k, and d be constants. We prove[2]:

Theorem 1.1 (Informal). *Suppose Φ is a CNF formula where each clause contains between k and $2k$ variables and at most d clauses contain any one variable. For $k \gtrsim 60 \log d$, there is a deterministic polynomial time algorithm for approximating the number of satisfying assignments to Φ within a multiplicative $(1 \pm 1/n^c)$ factor. Moreover, there is a randomized polynomial time algorithm to sample from a distribution that is $1/n^c$-close in total variation distance to the uniform distribution on satisfying assignments.*

See Theorems 6.2 and 6.4 for the corresponding formal theorem statements.

This algorithm closes an exponential gap between the known upper bounds [4, 5] and lower bounds [4]. It also shows that under Lovász Local Lemma-like conditions not only is it possible to efficiently find a satisfying assignment, it is possible to find a random one. **In this regime of parameters, the solution space need not even be connected.** Our approach is a significant departure from earlier techniques based either on path coupling [5] or adapting Weitz's approach to non-binary models and hypergraphs [4, 12, 23, 25, 27]. The results above appear in Theorems 6.2 and 6.4. Moreover, our techniques seem to extend to many non-binary counting problems, as we explain in Section 7.

Our approach starts from a thought experiment about what we could do if we had access to a very powerful oracle that could answer questions about the marginal distributions of individual variables under the uniform distribution on satisfying assignments. We use this oracle and properties of the Lovász Local Lemma (namely, bounds it gives on the marginal distribution of individual variables) to construct a coupling between two random satisfying assignments so that both agree outside some logarithmic sized component. If we could construct this coupling, then we could use brute-force search to find the ratio of the number of satisfying assignments with $x = T$ to the number with $x = F$ to compute marginals at x. However, the distribution of what component the coupling produces intimately depends on the powerful oracle we have assumed that we have access to.

Instead, we abstract the coupling procedure as a random root-to-leaf path in a tree that represents the state of the coupling. We show that at the leaves of this tree, there is a way to fractionally charge assignments where $x = T$ against assignments where $x = F$. Crucially, doing so requires only brute-force search on a logarithmic sized component. Finally, we show that there is a polynomial sized linear program to find a flow through the tree that produces an approximately valid way

[1] The hardness results in Reference [4] are formulated for approximate counting but carry over to approximate sampling. In particular, an oracle for approximately sampling from the set of satisfying assignments yields an oracle for approximating the marginal at any variable. Then one can invoke Lemma 7 in Reference [4].

[2] We have not made an attempt to optimize the constant in this theorem.

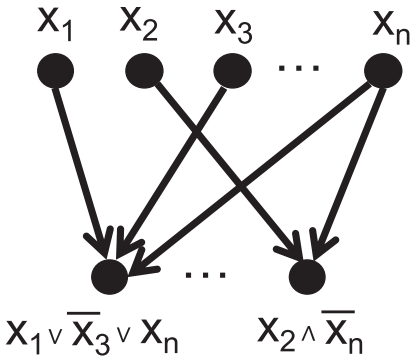

Fig. 1. An example cause network with n hidden variables. A sample is generated by choosing each hidden variable to be T/F independently with equal probability and observing the truth values of each clause.

to fractionally charge assignments with $x = T$ against ones with $x = F$, and that any such solution *certifies* the correct marginal distribution. From these steps, we have thus bootstrapped an oracle for answering queries about the marginal distribution. Our main results then follow from utilizing this oracle. In settings where the problem is self-reducible [26] it is well known how to go from knowing the marginal to approximate counting and sampling. In our setting, the problem is not self-reducible, because setting variables could result in clauses becoming too small, in which case k would not be large enough as a function of d. We are able to get around this by using the Lovász Local Lemma once more to find a safe ordering in which to set the variables.

In an exciting development and just after the initial posting of this article, Hermon, Sly, and Zhang [19] gave an algorithm for approximately counting the number of independent sets in a k-uniform hypergraph of maximum degree d provided that $k \geq 2 \log d + O(1)$. Their results are incomparable to ours, because they correspond to approximately counting in a monotone CNF formula. The techniques are entirely different and their algorithm matches the hardness result in Reference [4] up to an additive constant! It remains an interesting question to find similarly sharp phase transitions for the approximate counting problems studied here, namely for non-monotone CNFs. And in yet another interesting direction, Guo et al. [13] gave an algorithm based on connections to "cycle popping" that can uniformly sample from Φ under weaker conditions on the degree but by imposing conditions on intersection properties of bad events. In a related direction and building on our techniques, Guo et al. [14] gave an algorithm for approximately counting the number of q-colorings in a k-uniform hypergraph of maximum degree d, provided that $k \geq 28$ and $q \geq 315 d^{\frac{14}{k-14}}$. This is the first algorithm that can handle q being much smaller than d without any additional assumptions. While their overall algorithm follows a similar structure to ours, an important innovation is in circumventing our marking procedure and instead deciding which variables to set next in the coupling procedure in an adaptive manner.

1.3 Further Applications

Our algorithms have interesting applications in graphical models. Directed graphical models are a rich language for describing distributions by the conditional relationships of their variables. However, very little is known algorithmically about learning their parameters or performing basic tasks such as inference [8, 9]. In most settings, these problems are computationally hard. However, we can study an interesting class of directed graphical models that we call *cause networks*. See Figure 1.

Definition 1.2. In a cause network, there is a collection of hidden variables $x_1, x_2, \ldots, x_n$ that are chosen independently to be T or F with equal probability. There is a collection of m observed variables, each of which is either an OR or an AND of several variables or their negations.

Our goal is: Given a random sample $x_1, x_2, \ldots, x_n$ from the model where we observe the truth value of each of the m clauses, sample from the posterior distribution on the hidden variables. This generalizes graphical models such as the symptom-disease network where the hidden variables represent diseases that a patient may have, and the clauses represent observed symptoms. We will require the following regularity condition on our observations:

Definition 1.3. A collection of observations is k-regular if for every observed variable, the corresponding clause is adjacent to (i.e., shares a variable with) at most $15k/16$ OR clauses that are false and at most $15k/16$ AND clauses that are true.

Now, as an immediate corollary, we have:

Corollary 1.4. *Given a cause network where each observed variable depends on between k and $2k$ hidden variables, each hidden variable affects at most d observed variables and $k \gtrsim 70 \log d$, there is a polynomial time algorithm for sampling from the posterior distribution for any k-regular collection of observations.*

This is a rare setting where there is an algorithm to solve an inference problem in graphical models but (i) the underlying graph does not have bounded treewidth and (ii) correlation decay fails. We believe that our techniques may eventually be applicable to settings where the observed variables are noisy functions of the hidden variables and where the hidden variables are not distributed uniformly.

2 PRELIMINARIES

In this article, we will be interested in approximately counting the number of satisfying assignments to a CNF formula. For example, we could be given

$$\Phi = (\overline{x}_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee \overline{x}_8) \wedge \cdots \wedge (x_4 \vee \overline{x}_5 \vee x_9).$$

Let us fix some parameters. We will assume that there are n variables and there are m clauses, each of which is an OR of between k and Ck literals on distinct variables. The constant C will take values either 2 or 6, because of the way our algorithm will be built on various subroutines. Finally, we will require a degree bound that each variable appears in at most d clauses. We will be interested in the relationships between k and d that allow us to approximately count the number of satisfying assignments in polynomial time.

The celebrated Lovász Local Lemma tells us conditions on k and d, where we are guaranteed that there is at least one satisfying assignment. Let D be an upper bound on the degree of the dependency graph. We can take $D = 2dk$ or $D = 6dk$, depending on whether we are in a situation where there are at most $2k$ or at most $6k$ variables per clause.

Theorem 2.1. *[10] If $e(D+1) \leq 2^k$, then Φ has at least one satisfying assignment.*

Moser and Tardos [24] gave an algorithm to find a satisfying assignment under these same conditions. However, the assignment that their randomized algorithm finds is fundamentally not uniform from the set of all satisfying assignments. Our goal is to be able to both approximately count and uniformly sample when k is logarithmic in d.

There are many more related results, but we will not review them all here. Instead, we state a version of the asymmetric local lemma given in Reference [15], which gives us some control on the uniform distribution on satisfying assignments. Let $\mathcal{C}$ be the collection of clauses in Φ. Let

$\Pr[\cdot]$ denote the uniform distribution on all assignments—i.e., uniform on $\{T, F\}^n$. Finally, for a clause b, let $\Gamma(b)$ denote all the clauses that intersect b. We will abuse notation, and for any event a that depends on some set of the variables, let $\Gamma(a)$ denote all the clauses that contain any of the variables on which a depends.

Theorem 2.2. *Suppose there is a function $x : C \to (0, 1)$ such that for all $c \in C$ we have*

$$Pr[c \text{ is unsatisfied}] \leq x(c) \prod_{b \in \Gamma(c)} (1 - x(b)),$$

then there is at least one satisfying assignment. Moreover, the uniform distribution $\mathcal{D}$ on satisfying assignments satisfies that for any event a

$$Pr_{\mathcal{D}}[a] \leq Pr[a] \prod_{b \in \Gamma(a)} (1 - x(b))^{-1}.$$

Notice that this inequality is one-sided, as it ought to be. After all, if we take b to be some clause and a to be the event that b is not satisfied, then we know that $\Pr_{\mathcal{D}}[a] = 0$ even though $\Pr[a]$ is nonzero. However, what this theorem does tell us is that the marginal distribution of $\mathcal{D}$ on any variable is close to uniform. We will establish a quantitative version of this statement in the following corollary:

Corollary 2.3. *Suppose that $eDs \leq 2^k$. Then, for every variable x_i, we have*

$$\frac{1}{2} - \frac{2}{s} \leq Pr_{\mathcal{D}}[x_i = T] \leq \frac{1}{2} + \frac{2}{s}.$$

Proof. We will assume $s \geq 2$, since otherwise the corollary is trivial. Set $x(c) = \frac{1}{Ds}$ for each clause c, and consider the event a that $x_i = T$. Now invoking Theorem 2.2, we calculate

$$\begin{aligned}\Pr_{\mathcal{D}}[x_i = T] &\leq \Pr[x_i = T] \prod_{b \in \Gamma(a)} (1 - x(b))^{-1} \\ &\leq \left(\frac{1}{2}\right)\left(1 - \frac{1}{Ds}\right)^{-D} \leq \frac{1}{2} + \frac{2}{s},\end{aligned}$$

where the last inequality follows, because $(1 - \frac{1}{Ds})^{-D} \leq e^{\frac{2}{s}} \leq 1 + \frac{4}{s}$. An identical calculation works for the event $x_i = F$. All that remains is to check that the condition in Theorem 2.2 holds, which is a standard calculation: If c is a clause, then

$$\Pr[c \text{ is unsatisfied}] \leq \left(\frac{1}{Ds}\right)\left(1 - \frac{1}{Ds}\right)^{D}.$$

The left-hand side is at most 2^{-k}, because each clause has at least k distinct variables, and the right-hand side is at least $(\frac{1}{Ds})(\frac{1}{e})$. Rearranging completes the proof. □

Notice that k is still only logarithmic in d but with a larger constant, and by increasing this constant, we get some useful facts about the marginals of the uniform distribution on satisfying assignments.

3 A COUPLING PROCEDURE

3.1 Marked Variables

Throughout this section, we will assume that the number of variables per clause is between k and $6k$. Now, we are almost ready to define a coupling procedure. The basic strategy that we will employ is to start from either $x = T$ and $x = F$, and then sample from the corresponding marginal distribution on satisfying assignments. If we sample a variable y next, then Corollary 2.3 tells us

that regardless of whether $x = T$ or $x = F$, each clause has at least $k - 1$ variables remaining and so the marginal distribution on y is still close to uniform.

Thus, we will try to couple the conditional distributions, when starting from $x = T$ or $x = F$ as well as we can, to show that the marginal distribution on variables that are all at least some distance Δ away must converge in total variation distance. There is, however, an important catch that motivates the need for a fix. Imagine that we continue in this fashion, sampling variables from the appropriate conditional distribution. We can reach a situation where a clause c has all of its variables except y set and yet the clause is still unsatisfied. The marginal distribution on y is no longer close to uniform. Hence, reaching small clauses is problematic, because then we cannot say much about the marginal distribution on the remaining variables and it would be difficult to construct a good coupling.

Instead, our strategy is to use the Lovász Local Lemma once more, but to decide on a set of variables in advance that we call *marked*.

LEMMA 3.1. *Set* $c_0 = e^{(\frac{1}{2})(\frac{1}{4})^2}$. *Suppose that* $2e(D+1) \leq c_0^k$. *Then there is an assignment*

$$\mathcal{M} : \{x_i\}_{i=1}^n \to \{\textit{marked}, \textit{unmarked}\}$$

such that every clause c *has at least* $\frac{k}{4}$ *marked and at least* $\frac{k}{4}$ *unmarked variables.*

PROOF. We will choose each variable to be marked or unmarked with equal probability, and independently. Consider the m bad events, one for each clause c, that c does not have enough marked or enough unmarked variables. Then, we have

$$\Pr[c \text{ is bad}] \leq 2e^{-(\frac{1}{2})(\frac{1}{4})^2 k} = 2c_0^{-k},$$

which follows from the Chernoff bound. Now, we can appeal to the Lovász Local Lemma to get the desired conclusion. □

Only the variables that are marked will be allowed to be set to either T or F by the coupling procedure. Lemma 3.1 guarantees that every clause c always has enough remaining variables that can make it true so that the marginal distribution on any marked variable is always close to uniform.

3.2 Factorizing Formulas

Now fix a variable x. We will build up two partial assignments $\mathcal{A}_1$ and $\mathcal{A}_2$, and will use the notation

$$\mathcal{A}_1(x) = T \text{ and } \mathcal{A}_2(x) = F$$

to indicate that the first partial assignment sets x to T, and the second one sets x to F. Furthermore, we will refer to the conditional distribution that is uniform on all satisfying assignments consistent with the decisions made so far in $\mathcal{A}_1$ as $\mathcal{D}_1$. Similarly, we will refer to the other conditional distribution as $\mathcal{D}_2$. Note that these distributions are updated as more variables are set.

We can now state our goal. Suppose we have partial assignments $\mathcal{A}_1$ and $\mathcal{A}_2$. Then, we will want to write

$$\Phi_{\mathcal{A}_1} = \Phi_{I_1} \wedge \Phi_{O_1},$$

where $\Phi_{\mathcal{A}_1}$ is the subformula we get after making the assignments in $\mathcal{A}_1$ and simplifying—i.e., removing literals (a variable or its negation) that are set to F, and deleting clauses that already have a literal set to T. Similarly, we will want to write

$$\Phi_{\mathcal{A}_2} = \Phi_{I_2} \wedge \Phi_{O_2}.$$

Finally, we want the following conditions to be met:

(1) $\Phi_{O_1} = \Phi_{O_2} (:= \Phi_O)$
(2) Φ_{I_1} and Φ_O share no variables, and similarly for Φ_{I_2} and Φ_O.

The crucial point is that if we can find partial assignments $\mathcal{A}_1$ and $\mathcal{A}_2$ where $\Phi_{\mathcal{A}_1}$ and $\Phi_{\mathcal{A}_2}$ meet the above conditions, then the conditional distribution on all variables in Φ_O is exactly the same. We will use the notation

$$\mathcal{D}_1\Big|_{\text{vars}(\Phi_O)}$$

to denote the conditional distribution of $\mathcal{D}_1$ projected onto just the variables in Φ_O, and similarly for $\mathcal{D}_2$. Then, we have

Lemma 3.2. *If the above factorization conditions are met, then*

$$\mathcal{D}_1\Big|_{vars(\Phi_O)} = \mathcal{D}_2\Big|_{vars(\Phi_O)}.$$

Proof. From the assumption that $\Phi_{\mathcal{A}_1} = \Phi_{I_1} \wedge \Phi_O$ and because Φ_{I_1} and Φ_O share no variables, it means that there are no clauses that contain variables from both the subformulas Φ_{I_1} and Φ_O. Any such clause would prevent us from writing the formula $\Phi_{\mathcal{A}_1}$ in such a factorized form. Thus, the distribution $\mathcal{D}_1$ is simply the cross product of the uniform distributions on satisfying assignments to Φ_{I_1} and Φ_O. An identical statement holds for $\mathcal{D}_2$, which completes the proof. □

Note that meeting the factorization conditions does *not* mean that the *numbers* of satisfying assignments to $\Phi_{\mathcal{A}_1}$ and $\Phi_{\mathcal{A}_2}$ are the same.

3.3 Factorization Via Coupling

Our goal in this subsection is to give a coupling procedure (Algorithm 1) to generate partial assignments $\mathcal{A}_1$ and $\mathcal{A}_2$ starting from $x = T$ and $x = F$, respectively, that result in a factorized formula. In fact, we will set exactly the same set S of variables in both, although not all variables will be set to the same value in the two partial assignments and this set S will also be random.

There are two important constraints that we will impose on how we construct the partial assignments, which will make it somewhat tricky. First, suppose we have only set the variable x and next we choose to set the variable y in both $\mathcal{A}_1$ and $\mathcal{A}_2$. We will want the distribution on how we set y in the coupling procedure in $\mathcal{A}_1$ to match the conditional distribution $\mathcal{D}_1$ and similarly for $\mathcal{A}_2$. Now, suppose we terminate with some set S having been set. We can continue sampling the variables in $\bar{S}$ from $\mathcal{D}_1$, and we are now guaranteed that the full assignment we generate is uniform from the set of assignments with $x = T$. An identical statement holds when starting with $x = F$. Second, we will want that with very high probability, the coupling procedure terminates with not too many variables in the formula Φ_{I_1} or Φ_{I_2}. Finally, we will assume that we are given access to a powerful oracle:

Definition 3.3. We will call the following a *conditional distribution oracle*: Given a CNF formula Φ, a partial assignment $\mathcal{A}$, and a variable y, it returns the probability that $y = T$ in a uniformly random satisfying assignment that is consistent with $\mathcal{A}$.

Such an oracle is obviously very powerful, and it is well known that if we had access to it we could compute the number of satisfying assignments to Φ exactly with a polynomial number of queries. However, one should think of the coupling procedure as a thought experiment, which will be useful in an indirect way to build up towards our algorithm for approximate counting[3].

Here, we record some important properties of Algorithm 1. Notice that a clause c can only trigger the WHILE loop at most once. If it ends up in Case # 1, then it is deleted from the formula. If it ends up in Case # 2, then all its variables are included in V_I and once a variable is included in V_I

[3]Here, by "best coupling at each step," we mean that sequentially for each variable we maximize the probability that $\Pr[\mathcal{A}_1(y) = \mathcal{A}_2(y)]$ while preserving the fact that y is set in $\mathcal{A}_1$ and $\mathcal{A}_2$ according to $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively.

ALGORITHM 1: Coupling Procedure
Input: Monotone CNF Φ, variable x and conditional distribution oracle $\mathcal{Z}$

(1) Using Lemma 3.1, label variables as marked or unmarked
(2) Initialize $\mathcal{A}_1(x) = T$ and $\mathcal{A}_2(x) = F$
(3) Initialize $V_I = \{x\}$ and $V_O = \{x_i\}_{i=1}^n \setminus \{x\}$
(4) While there is a clause c with variables in both V_I and V_O
(5) Sequentially sample its marked variables (if any) from $\mathcal{D}_1$ and $\mathcal{D}_2$, using $\mathcal{Z}$ to construct best coupling at each step[3]
(6) Case # 1: c is already satisfied in both $\mathcal{A}_1$ and $\mathcal{A}_2$
(7) Let S be the variables in c that have different truth values in $\mathcal{A}_1$ and $\mathcal{A}_2$.
(8) Update $V_I \leftarrow V_I \cup S$, $V_O \leftarrow V_O \setminus S$
(9) Delete c
(10) Case # 2: c is not already satisfied by both $\mathcal{A}_1$ and $\mathcal{A}_2$
(11) Let S be all variables in c (marked or unmarked)
(12) Update $V_I \leftarrow V_I \cup S$, $V_O \leftarrow V_O \setminus S$
(13) End

it is never removed. Thus, Algorithm 1 clearly terminates. Our first step is to show that when it does terminate, the formula factorizes. Let C_I be the set of remaining clauses that have all of their variables in V_I. Similarly, let C_O be the set of remaining clauses that have all of their variables in V_O. Then, set

$$\Phi'_I = \wedge_{c \in C_I} c$$

and let Φ_{I_1} and Φ_{I_2} be the simplifications of Φ'_I with respect to the partial assignments $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Similarly, set

$$\Phi'_O = \wedge_{c \in C_O} c$$

and let Φ_{O_1} and Φ_{O_2} be the simplifications of Φ'_O with respect to the partial assignments $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Claim 1. *All variables with different truth assignments in $\mathcal{A}_1$ and $\mathcal{A}_2$ are in V_I.*

Proof. A variable is set in response to it being contained in some clause c that triggers the WHILE loop. Any such variable having different truth assignments is moved into V_I in both Case # 1 and Case # 2. □

Now, we have an immediate corollary that will help us in proving that Algorithm 1 finds a pair of partial assignments for which Φ factorizes:

Corollary 3.4. $\Phi_{O_1} = \Phi_{O_2}$.

Proof. Recall that Φ_{O_1} and Φ_{O_2} come from simplifying Φ'_O (which contains only variables in V_O) according to $\mathcal{A}_1$ and $\mathcal{A}_2$. From Claim 1, we know that $\mathcal{A}_1$ and $\mathcal{A}_2$ are equal when restricted to V_O and, thus, we get the same formula in both cases. □

Now that we know they are equal, we can define $\Phi_O = \Phi_{O_1} = \Phi_{O_2}$. What remains is to show that the subformulas we have are actually factorizations of the original formula Φ:

Lemma 3.5. $\Phi_{\mathcal{A}_1} = \Phi_{I_1} \wedge \Phi_O$ *and* $\Phi_{\mathcal{A}_2} = \Phi_{I_2} \wedge \Phi_O$.

Proof. When the WHILE loop in Algorithm 1 terminates, every clause c in the original formula Φ either has all of its variables in V_I or in V_O or was deleted, because it already contains at least one variable in both $\mathcal{A}_1$ and $\mathcal{A}_2$ that satisfies it (although it need not be the same variable). Hence, every clause in Φ that is not already satisfied in both $\mathcal{A}_1$ and $\mathcal{A}_2$ shows up in $\Phi'_I \wedge \Phi'_O$. Some clauses that

are already satisfied in both may show up as well. In any case, this completes the proof, because the remaining operation just simplifies the formulas according to the partial assignments. □

3.4 How Many Steps Does the Coupling Procedure Take to Terminate?

What remains is to bound the probability that the number of variables included in V_I is at most t. First, we need an elementary definition:

Definition 3.6. When a variable x_i is given different truth assignments in $\mathcal{A}_1$ and $\mathcal{A}_2$, we call it a *type 1* error. When a clause c has all of its marked variables set in both $\mathcal{A}_1$ and $\mathcal{A}_2$, but in at least one of them is not yet satisfied, we call it a *type 2* error.

Note that it is possible for a variable to participate in both a type 1 and type 2 error. In any case, these are the *only* reasons that a variable is included in V_I in an execution of the coupling procedure:

Observation 1. *All variables in V_I are included either due to a type 1 error or a type 2 error, or both.*

Now our approach to showing that V_I contains not too many variables with high probability is to show that if it did, there would be a large collection of disjoint errors. First, we construct a useful graph underlying the process:

Definition 3.7. Let G be the graph on vertices V_I where we connect variables if and only if they appear in the same clause together (*any* clause from the original formula Φ).

The crucial property is that this graph is connected:

Observation 2. *G is connected.*

Proof. This property holds by induction. Assume that at the start of the WHILE loop, the property holds. Then, at the end of the loop, any variable x_i added to V_I must have been contained in a clause c that at the start of the loop had one of its variables in V_I. This completes the proof. □

Now, by Observation 1, for every variable in V_I, we can blame it on either a type 1 or a type 2 error. Both of these types of errors are unlikely. But for each variable, charging it to an error is problematic because of overlaps in the events. In particular, suppose we have two variables x_i and x_j that are both included in V_I. It could be that both variables are in the same clause c, which resulted in a type 2 error, in which case we could only charge one of the variables to it. This turns out not to be a major issue.

The more challenging type of overlap is when two clauses c and c' both experience type 2 errors and overlap. In isolation, each clause would be unlikely to experience a type 2 error. But it could be that c and c' share all but one of their marked variables, in which case once we know that c experiences a type 2 error, then c' has a reasonable chance of experiencing one as well. We will get around this issue by building what we call a 3-tree. This approach is inspired by Noga Alon's parallel algorithmic local lemma [2], where he uses a 2,3-tree.

Definition 3.8. We call a graph T on subset of V_I a 3-tree if each vertex is distance at least 3 from all the others, and when we add edges between vertices at distance exactly 3 the tree is connected.

Next, we show that G contains a large 3-tree:

Lemma 3.9. *Suppose that any clause contains between k and $6k$ variables. Then, any maximal 3-tree contains at least $\frac{|V_I|}{2(6dk)^2}$ vertices.*

Proof. Consider a maximal 3-tree T. We claim that every vertex $x_i \in V_I$ must be distance at most 2 from some x_j in T. If not, then we could take the shortest path from x_i to T and move along it, and at some point we would encounter a vertex that is also not in T whose distance from T is exactly 3, at which point we could add it, contradicting T's maximality. Now, for every x_i in T, we remove from consideration at most $(6dk)^2 + (6dk)$ other variables (all those at distance at most 2 from x_i in G). This completes the proof. □

Now, we can indeed charge every variable in T to a disjoint error:

Claim 2. *If two variables x_i and x_j in T are the result of type 2 errors for c and c', then*

$$vars(c_i) \cap vars(c_j) = \emptyset.$$

Proof. For the sake of contradiction, suppose that $\text{vars}(c_i) \cap \text{vars}(c_j) \neq \emptyset$. Then, since c and c' experience type 2 errors, all of their variables are included in V_I. This gives a length 2 path from x_i to x_j in G, which if they were both included in T, would contradict the assumption that T is a 3-tree. □

We are now ready to prove the main theorem of this section:

Theorem 3.10. *Suppose that every clause contains between k and $6k$ variables and that $\log d \geq 5 \log k + 20$ and $k \geq 50 \log d + 50 \log k + 250$. Then,*

$$Pr[|V_I| \geq 2(6dk)^2 t] \leq \left(\frac{1}{2}\right)^t.$$

Proof. First, note that the conditions on k and d imply that the condition in Lemma 3.1 holds. Now, suppose that $|V_I| \geq 2(6dk)^2 t$. Then, by Lemma 3.9, we can find a 3-tree T with at least t vertices. First, we will work towards bounding the probability of any particular 3-tree on t vertices. We note that, since each clause has at least $k/4$ marked variables and has at most $6k$ total variables, we can bound the probability of a type 1 error as

$$\Pr[x \text{ has type 1 error}] \leq \frac{4}{d^{11}}.$$

This uses the assumption $ed^{11}(6dk) \leq 2^{k/4}$, which allows us to choose $s = d^{11}$ in Corollary 2.3. Also, we can bound the probability of a variable participating in a type 2 error as

$$\Pr[x \text{ participates in a type 2 error}] \leq d\left(\frac{49}{100}\right)^{k/4},$$

which follows from Corollary 2.3 using the condition that $s \geq d^3 \geq 100$ and that each variable belongs to at most d clauses and each clause has at least $k/4$ marked variables. Now, by Claim 2, we know that clauses that cause the type 2 errors for each vertex in T are disjoint. Thus, putting it all together, we can bound the probability of any particular 3-tree on t vertices as

$$\left(\frac{4}{d^{11}} + d\left(\frac{49}{100}\right)^{k/4}\right)^t.$$

Now, it is well known (see References [2, 21]) that the number of trees of size t containing a fixed vertex in a graph of degree at most Δ is at most $(e\Delta)^t$. Moreover, if we connect pairs of vertices in G that are distance exactly 3 from each other, then we get a new graph H whose maximum degree is at most $\Delta = 3(6dk)^3$. Thus, putting it all together, we have that the probability that $|V_I| > 2(6dk)^2 t$ can be bounded by

$$\left(\frac{4}{d^{11}} + d\left(\frac{49}{100}\right)^{k/4}\right)^t (3e(6dk)^3)^t \leq \left(\frac{1}{2}\right)^t,$$

where the last inequality follows from the assumptions that $\log d \geq 5 \log k + 20$ and $k \geq 50 \log d + 50 \log k + 250$. □

Thus, we can conclude that with probability at least $1 - \frac{1}{n^c}$ the number of variables in V_I is at most τ where $\tau = 2c(6dk)^2 \log n$. In our description of the coupling procedure, we relied on access to a conditional distribution oracle. But suppose for the moment that we could get around this and run the coupling procedure without such an oracle. Then, whenever the coupling procedure terminates with $|V_I| \leq \tau$, we could use brute force to count the number of satisfying assignments to Φ_{I_1} and Φ_{I_2}. The number of candidate assignments we would need to search over is polynomial in $n^{cd^2k^2}$ and, thus, the algorithm would run in polynomial time for any fixed c, d, and k.

The important point is that, because the formula factorizes upon termination of the coupling procedure, the ratio between the number of satisfying assignments in Φ_{I_1} and Φ_{I_2} is the same as the ratio of the number of satisfying assignments in $\Phi_{\mathcal{A}_1}$ and $\Phi_{\mathcal{A}_2}$. In the next section, we show how we can use these ratios to approximate the marginal probability that x is set to true under a random satisfying assignment of Φ. Then, eventually we will need to find a way around our use of the conditional distribution oracle.

4 IMPLICATIONS OF THE COUPLING PROCEDURE

In this section, we give an abstraction that allows us to think about the coupling procedure as a randomly chosen root-to-leaf path in a certain tree whose nodes represent states. First, we make an elementary observation that will be useful in discussing how this tree is constructed. Recall that the coupling procedure chooses *any* clause that contains variables in both V_I and V_O and then samples *all* marked variables in it. We will assume without loss of generality that the choices it makes are done in lexicographic order. So, if the clauses in Φ are ordered arbitrarily as $c_1, c_2, \ldots, c_m$ and the variables are ordered as $x_1, x_2, \ldots, x_n$, if when executing the WHILE loop the algorithm has a choice of more than one clause, it selects among them the clause c_i with the lowest subscript i. Similarly, given a choice of which marked variable to sample next, it selects the x_j with the lowest subscript j.

The important point is that now we can think of a state associated with the coupling procedure, which we will denote by σ.

Definition 4.1. The state σ of the coupling procedure specifies the following:

(1) The set of remaining clauses C'—i.e., clauses that have not yet been deleted;
(2) The current partition of the variables into V_I and V_O;
(3) The set S of variables whose values have been set, along with their values in both $\mathcal{A}_1$ and $\mathcal{A}_2$;
(4) The current clause c^* being operated on in the while loop, if any.

We will assume that the set $\mathcal{M}$ of marked variables is fixed once and for all. Now, the transition rules from state σ are as follows:

(1) If c^* has any marked variables that are unset, then the algorithm chooses the lexicographically first and sets it.
(2) If c^* has no remaining marked variables to set, then the algorithm updates C', V_I, and V_O, according to whether c^* falls into Case # 1 or Case #2 in Algorithm 1 and sets the current clause to empty.
(3) If the current clause is empty, then the algorithm chooses the lexicographically first clause from C', which has at least one variable in each of V_I and V_O to be c^*.

Finally, we can define the next variable operation:

ALGORITHM 2: Decision Tree Sampling
Input: Monotone CNF Φ, stochastic decision tree S

(1) Choose a random root-to-leaf path in S
(2) Choose a uniformly random assignment A_1 consistent with $\mathcal{A}_1$
(3) Choose a uniformly random assignment A_2 consistent with $\mathcal{A}_2$
(4) Output A_1 with probability $q = \Pr_{\mathcal{D}}[x = T]$, and otherwise output A_2

Definition 4.2. Let $\mathcal{R} : \Sigma \to \{x_i\}_{i=1}^n \cup \{\emptyset\} \times \Sigma$ be the function that takes in a state σ, transitions to the next state σ' that sets some variable y and outputs (y, σ').

Note that some states σ do not immediately set a variable—e.g., if the next operation is to choose the next clause, or update C', V_I, and V_O. These latter transitions are deterministic, so we let σ' be the end resulting state and y be the variable that it sets. Now, we can define the stochastic decision tree underlying the coupling procedure:

Definition 4.3. Given a conditional distribution oracle $\mathcal{Z}$, the function $\mathcal{R}$ and a stopping threshold s, the associated *stochastic decision tree* is the following:

(1) The root node corresponds to the state where only x is set, $\mathcal{A}_1(x) = T, \mathcal{A}_2(x) = F, V_I = \{x\}$ and $V_O = \{x_i\}_{i=1}^n \setminus \{x\}$.
(2) Each node has either zero or four descendants. If the current node corresponds to state σ, let $(y, \sigma') = \mathcal{R}(\sigma)$. Then, if $y = \emptyset$ or if $|V_I| = \tau$, there are no descendants and the current node is a leaf corresponding to the termination of the coupling procedure or $|V_I|$ being too large. Otherwise, the four descendants correspond to the four choices for how to set y in $\mathcal{A}_1$ and $\mathcal{A}_2$, and are marked with the state σ'', which incorporates their respective choices into σ'.
(3) Moreover, the probability on an edge from a state σ' to a state σ'' where y has been set as $\mathcal{A}_1(y) = T$ and $\mathcal{A}_2(y) = T$ is equal to

$$\min(\mathcal{D}_1(y), \mathcal{D}_2(y))$$

and the transition to the state where $\mathcal{A}_1(y) = F$ and $\mathcal{A}_2(y) = F$ has probability

$$\min(1 - \mathcal{D}_1(y), 1 - \mathcal{D}_2(y)).$$

Finally, if $\mathcal{D}_1(y) > \mathcal{D}_2(y)$, then the transition to $\mathcal{A}_1(y) = T$ and $\mathcal{A}_2(y) = F$ is non-zero and is assigned all the remaining probability. Otherwise, the transition to $\mathcal{A}_1(y) = F$ and $\mathcal{A}_2(y) = T$ is non-zero and is assigned all the remaining probability.

Definition 4.4. We will refer to a leaf node where the coupling has terminated as a *coupled leaf node*. Otherwise, we will refer to it as an *uncoupled leaf node*.

Now, we can use the stochastic decision tree to give an alternative procedure to sample a uniformly random satisfying assignment of Φ. We will refer to the process of starting from the root, and choosing a descendant with the corresponding transition probability, until a leaf node is reached as "choosing a random root-to-leaf path."

Claim 3. *Algorithm 2 outputs a uniformly random satisfying assignment of* Φ.

Proof. We could alternatively think of Algorithm 2 as deciding on whether (at the end) to output A_1 or A_2 with probability q vs. $1 - q$ at the outset. Then, if we choose to output A_1 and we only keep track of the choices made for $\mathcal{A}_1$, marginally these correspond to sequentially sampling the assignment of variables from $\mathcal{D}_1$. And when we reach a leaf node in S, we can interpret the

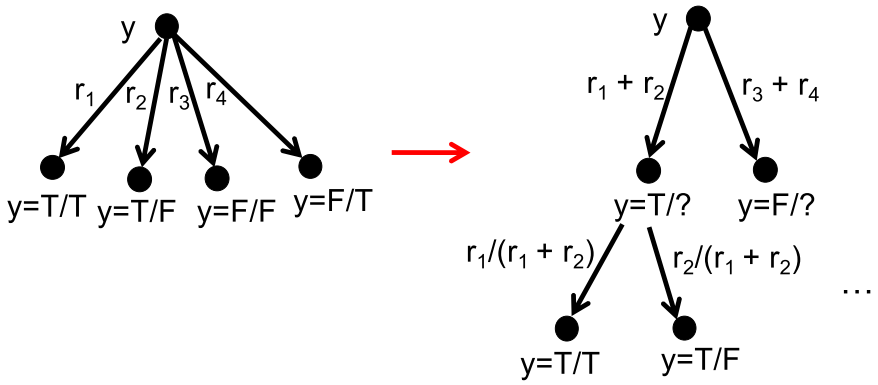

Fig. 2. A transformation on the stochastic decision tree that makes it easier to understand what happens when we condition on the assignment A that is output by Algorithm 2. It will also be used in Section 5.2.

remaining choices to A_1 as sampling all unset variables from $\mathcal{D}_1$. Thus, the output in this case is a uniformly random satisfying assignment with $x = T$. An identical statement holds for when we choose to output A_2, and because we decided between them at the outset with the correct probability, this completes the proof of the claim. □

Now, let σ be the state of a leaf node u and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the resulting partial assignments. Let p_1 be the product of certain probabilities along the root-to-leaf path. In particular, suppose along the path there is a transition with y being set. Let q_1 be the probability of the transition to $(\mathcal{A}_1(y), \mathcal{A}_2(y))$—i.e., along the branch that it actually went down. And let q_2 be the probability of the transition to $(\mathcal{A}_1(y), \overline{\mathcal{A}_2(y)})$—i.e., where y is set the same in $\mathcal{A}_1$ but is set to the opposite value as it was in $\mathcal{A}_2$. We let p_1 be the product of all $\frac{q_1}{q_1+q_2}$ over all such decisions on the root-to-leaf path.

Lemma 4.5. *Let A be an assignment that agrees with $\mathcal{A}_1$. Then, for Algorithm 2,*

$$Pr[\textit{terminates at leaf } u | \textit{outputs assignment } A] = p_1.$$

Proof. The idea behind this proof is to think of the random choice of which of the four descendants to transition to as being broken down into two separate random choices where we first choose $\mathcal{A}_1(y)$ and then we choose $\mathcal{A}_2(y)$. See Figure 2. Now, we can make the random choices in Algorithm 2 in an entirely different order. Instead of choosing the transition in the first layer, then the second layer, and so on, we instead make *all* of the choices in the odd layers. Moreover, at each leaf, we choose which assignment consistent with $\mathcal{A}_1$ we would output. This is the first phase. Next, we choose whether to output the assignment consistent with $\mathcal{A}_1$ or with $\mathcal{A}_2$. Finally, we make all the choices in the even layers, which fixes the root-to-leaf path, and then we choose an assignment consistent with $\mathcal{A}_2$. This is the second phase.

The key point is that once the output A is fixed, all of the choices in the first phase are determined, because every time a variable y is set, it must agree with its setting in A. Moreover, each leaf node must choose A for its assignment consistent with $\mathcal{A}_1$. And finally, we know that the sampling procedure must output the assignment consistent with $\mathcal{A}_1$, because A agrees with $\mathcal{A}_1$ and not $\mathcal{A}_2$ (because they differ on how they set x). Thus, conditioned on outputting A, the only random choices left are those in the second phase. Now, the lemma follows, because the probability of reaching leaf node u is exactly the probability along the path of all of the even layer choices, which is how we defined p_1. □

We can define p_2 in an analogous way to how we defined p_1, and the lemma above shows that p_2 is exactly the probability of all the decisions made along the root-to-leaf path conditioned on the output being A where A agrees with $\mathcal{A}_2$. The key lemma is the following:

Lemma 4.6. *Let N_1 be the number of satisfying assignments consistent with $\mathcal{A}_1$ and let N_2 be the number of satisfying assignments consistent with $\mathcal{A}_2$. Then,*

$$\frac{p_1 N_1}{p_2 N_2} = \frac{q}{1-q}.$$

Proof. Let u be a leaf node. Consider a random variable Z_u that when we run Algorithm 2 is non-zero if and only if we end at u. Moreover, let $Z_u = (1-q)$ if an assignment with $x = T$ is output, and $Z_u = -q$ if an assignment with $x = F$ is output. Then, clearly $\mathbf{E}[Z_u] = 0$. Now, alternatively, we can write

$$\mathbf{E}[Z_u] = \mathop{\mathbf{E}}_{A}[\mathbf{E}[Z_u | A \text{ is output}]],$$

where A is a uniformly random satisfying assignment of Φ, precisely because of Claim 3. Let N be the total number of such assignments. Then,

$$\mathop{\mathbf{E}}_{A}[\mathbf{E}[Z_u|A]] = \left(\frac{N_1}{N}\right)(p_1)(1-q) + \left(\frac{N_2}{N}\right)(p_2)(-q).$$

This follows because the only assignments A that can be output at u must be consistent with either $\mathcal{A}_1$ or $\mathcal{A}_2$. Note that these are disjoint events, because in one of them $x = T$ while in the other $x = F$. Then, once we know that A is consistent with $\mathcal{A}_1$ (which happens with probability $\frac{N_1}{N}$), the probability for the decisions made in $\mathcal{A}_2$ being such that we reach u is exactly p_1, as this was how it was defined. The final term in the product of three terms is just the value of Z_u. An identical argument justifies the second term. Now, using the fact that the above expression evaluates to zero and rearranging completes the proof. □

5 CERTIFYING THE MARGINAL DISTRIBUTION

5.1 Overview

The stochastic decision tree that we defined in the previous section is a natural representation of the trajectory of the coupling procedure. Its crucial property is captured in Lemma 4.6, which gives a relation between

(1) q—the probability that a satisfying assignment to Φ sets x to true
(2) p_i—the conditional probability of a satisfying assignment consistent with $\mathcal{A}_i$ reaching a particular leaf node u and
(3) N_i—the number of satisfying assignments consistent with $\mathcal{A}_i$

for $i = 1, 2$. In particular, at every leaf node u (whether it is coupled or uncoupled), the equation

$$\frac{p_1 N_1}{p_2 N_2} = \frac{q}{1-q}$$

is satisfied.

Why is this property useful? Imagine that we did not know q but that someone gave use a stochastic decision tree satisfying the equation above at every leaf node. Furthermore, suppose that every leaf node is coupled. We claim that we could convince ourselves, in polynomial time, that q is the correct marginal probability. First, we could compute $\frac{p_1 N_1}{p_2 N_2}$. It is straightforward to compute p_1 and p_2, and we can use the fact that the coupling terminated to compute $\frac{N_1}{N_2}$. More precisely,

because the coupling terminated, we know that when we substitute in the partial assignments, the simplified formulas factorize as

$$\Phi_{\mathcal{A}_i} = \Phi_{I_1} \wedge \Phi_O$$

Thus, the ratio $\frac{N_1}{N_2}$ is the same as the ratio between the numbers of satisfying assignments to Φ_{I_1} and Φ_{I_2}. Finally, these are formulas on a logarithmic number of variables so we can compute the number of satisfying assignments in polynomial time by brute-force search.

Now, we could check that at every leaf node u, the ratio $\frac{p_1 N_1}{p_2 N_2}$ is the same and solve for q in the equation in Lemma 4.6. But why does this guarantee that q is the correct marginal probability? Consider any satisfying assignment to Φ that sets x to true. This assignment can contribute to the count N_1 at many different leaf nodes, but when we multiply by p_1 it contributes one total unit to the sum of $p_1 N_1$ over all leaf nodes. This is true because the values p_1 represent the conditional distribution of which leaf node the coupling procedure maps the assignment to. The same is true for any satisfying assignment to Φ that sets x to false—it contributes one total unit to the sum of $p_2 N_2$ over all leaf nodes. Now, if every satisfying assignment contributes one unit, and we have broken up the set of all satisfying assignments into fractional groups where the ratio between the contributions of the satisfying assignments with x set to true and those where x is set to false is always $\frac{q}{1-q}$, it follows that this is also the the ratio between the total numbers of satisfying assignments with x set to true and those where x is set to false.

What happens if there are uncoupled leaf nodes? We can no longer compute the ratio $\frac{N_1}{N_2}$. But it turns out that we can ignore uncoupled leaf nodes and still be able convince ourselves that q is close to the correct marginal probability. More precisely, the same sorts of arguments that we used to show that the coupling procedure terminates after a logarithmic number of steps with high probability will also imply that when we ignore trajectories in the coupling procedure that have not terminated in a logarithmic number of steps, we have not changed the total contribution of a typical satisfying assignment to the sum of $p_1 N_1$ by much.

Finally, what if we are not given the stochastic decision tree? Our approach is to use linear programming to search for a stochastic decision tree that satisfies the equation $\frac{p_1 N_1}{p_2 N_2} = \frac{q}{1-q}$ at every coupled leaf node. The natural approach would be to think of the probabilities of its transitions as flows along the tree. However, p_1 turns out to be a complicated and non-linear function of this encoding. Nevertheless, there is an alternate encoding where

$$p_1 \frac{N_1}{N_2} = p_2 \frac{q}{1-q}$$

is a linear equation. We will describe this alternate encoding, which is the last ingredient in our overall algorithm, in the next subsection.

5.2 One-Sided Stochastic Decision Trees

In this subsection, we will transform a stochastic decision tree into two separate trees, that we call *one-sided stochastic decision trees*. The idea is to consider the conditional distribution on trajectories of the coupling procedure, when we fix what $\mathcal{A}_1$ will be at termination. Unlike stochastic decision trees, one-sided decision trees will have the crucial property that

$$p_1 \frac{N_1}{N_2} = p_2 \frac{q}{1-q}$$

is a linear equation in the transition probabilities. First, we explain the transformation from a stochastic decision tree to a one-sided stochastic decision tree. See Figure 2 for an illustration. We will then formally define its properties and what we require of it.

Now, suppose we are given a stochastic decision tree S. Let us construct the one-sided stochastic decision tree S_1 that represents the trajectory of the partial assignment $\mathcal{A}_1$. When we start from the starting state σ (see Definition 4.1), the four descendants of it in S will now be four grandchildren. Its immediate descendants will be two nodes u and u', one representing the choice $\mathcal{A}_1(y) = T$ and one representing $\mathcal{A}_1(y) = F$, where y is the next variable set (see Definition 4.2). The two children of σ in S that correspond to $\mathcal{A}_1(y) = T$ will now be the children of u and the other two children will now be the children of u'. We will continue in this way so that alternate layers represent nodes present in S and new nodes.

This alone does not change much the semantics of the trajectory. All we are doing is breaking up the decision of which of the four children to proceed to, into two separate decisions. The first decision is based on just $\mathcal{A}_1$ and the second is based on $\mathcal{A}_2$. However, we will change the semantics of what probabilities we associate with different transitions. For starters, we will work with total probabilities. So the total probability incoming into the starting node is 1. Let us see how this works inductively. Let us now suppose that σ represents the state of some node in S (not necessarily the starting state) and u and u' are its descendants in S_1. Then, if the total probability into σ in S_1 is z, we place z along both the edges to u and to u'. This is because the decision tree is now from the perspective of $\mathcal{A}_1$, who perhaps has already chosen his assignment uniformly at random from the satisfying assignments with $x = T$ but has not set all of those values in $\mathcal{A}_1$. Hence, his decision is not a random variable, since given the option of transition to u or u' he must go to whichever one is consistent with his hidden values.

However, from this perspective, the choices corresponding to $\mathcal{A}_2$ are random, because he has no knowledge of the assignment that the other player is working with. If we have z total probability coming into u, then the total probability into its two descendants will be $(\frac{q_1}{q_1+q_2})z$ and $(\frac{q_2}{q_1+q_2})z$, respectively, where q_1 and q_2 were the probabilities on the transitions in S into the two corresponding descendants. In particular, if q_1 is the probability of setting $\mathcal{A}_1(y) = T$ and $\mathcal{A}_2(y) = T$ and q_2 is the probability of setting $\mathcal{A}_1(y) = T$ and $\mathcal{A}_2(y) = F$, then $(\frac{q_1}{q_1+q_2})z$ is the total probability on the transition from u to the descendant where $\mathcal{A}_2(y) = T$ and $(\frac{q_2}{q_1+q_2})z$ is the total probability on the transition from u to the descendant where $\mathcal{A}_2(y) = F$. Note that from Corollary 2.3, we have that

$$\left(\frac{q_2}{q_1 + q_2}\right)z \leq \left(\frac{4}{s}\right)z.$$

This is an important property that we will make crucial use of later. Notice that it is a linear constraint in the total probability. Now, we are ready to define a one-sided stochastic decision tree, which closely mirrors Definition 4.3.

Definition 5.1. Given the function $\mathcal{R}$ and a stopping threshold τ, the associated *one-sided stochastic decision tree* for $\mathcal{A}_1$ is the following:

(1) The root node corresponds to the state where only x is set, $\mathcal{A}_1(x) = T, \mathcal{A}_2(x) = F, V_I = \{x\}$ and $V_O = \{x_i\}_{i=1}^n \setminus \{x\}$.
(2) Each node has either two descendants and four *grand*-descendants or zero descendants. If the current node a corresponds to state σ, let $(y, \sigma') = \mathcal{R}(\sigma)$. Then, if $y = \emptyset$ or if $|V_I| = \tau$, there are no descendants and the current node is a leaf corresponding to the termination of the coupling procedure or $|V_I|$ being too large. Otherwise, the two descendants correspond to the two choices for how to set y in $\mathcal{A}_1$. Each of their two descendants correspond to the two choices for how to set y in $\mathcal{A}_2$. Each grand-descendant is marked with the state σ', which incorporates their respective choices.
(3) Let z be the total probability into a. Then, the total probability into each descendant is z. Moreover, let the total probability into the grand-descendants with states $\mathcal{A}_1(y) = T$ and

$\mathcal{A}_2(y) = T$ and $\mathcal{A}_1(y) = T$ and $\mathcal{A}_2(y) = F$ be z_1 and z_2, respectively. Then, z_1 and z_2 are nonnegative, sum to z and satisfy $z_2 \leq (\frac{4}{s})z$. Similarly, let the total probability into the grand-descendants with states $\mathcal{A}_1(y) = F$ and $\mathcal{A}_2(y) = F$ and $\mathcal{A}_1(y) = F$ and $\mathcal{A}_2(y) = T$ be z_3 and z_4, respectively. Then, z_3 and z_4 are nonnegative, sum to z and satisfy $z_4 \leq (\frac{4}{s})z$.

The one-sided stochastic decision tree for $\mathcal{A}_2$ is defined analogously, in the obvious way. Finally, we record an elementary fact:

Claim 4. *There is a perfect matching between the root-to-leaf paths in S_1 and S_2, so that any pair of assignments A_1 and A_2 that takes a root-to-leaf path p in S_1, must also take the root-to-leaf path in S_2 to which p is matched.*

Proof. Recall that the odd levels in S_1 and S_2 correspond to the nodes in S. Therefore, from a root-to-leaf path p in S_1, we can construct the root-to-leaf path in S, which in turn uniquely defines a root-to-leaf path in S_2 (because it specifies which nodes are visited in odd layers, and all paths end on a node in an odd layer). □

5.3 An Algorithm for Finding a Valid S_1 and S_2

We are now ready to prove one of the two main theorems of this section:

Theorem 5.2. *Let $q = Pr_{\mathcal{D}}[x = T]$ and $q' \leq q \leq q''$. Then there are two one-sided stochastic decision trees S_1 and S_2 that, for any pair of matched root-to-leaf paths terminating in u and u', respectively, satisfy*

$$\left(\frac{q'}{1-q'}\right)p_2N_2 \leq p_1N_1 \leq \left(\frac{q''}{1-q''}\right)p_2N_2,$$

where N_1 and N_2 are number of satisfying assignments consistent with $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, and p_1 and p_2 are the total probabilities into u and u', respectively.

Moreover, given q' and q'' that satisfy $q' \leq q \leq q''$ there is an algorithm to construct two one-sided stochastic decision trees S_1 and S_2 that satisfy the above condition on all matched coupled leaf nodes, which runs in time polynomial in m and 2^τ, where τ is the stopping threshold.

Proof. The first part of the theorem follows from the transformation we gave from a stochastic decision tree to two one-sided stochastic decision trees. Then, Claim 4 combined with Lemma 4.6 implies $\frac{q}{1-q} = \frac{p_1N_1}{p_2N_2}$, which then necessarily satisfies $\frac{q'}{1-q'} \leq \frac{p_1N_1}{p_2N_2} \leq \frac{q''}{1-q''}$. Rearranging completes the proof of the first part.

To prove the second part of the theorem, notice that if τ is the stopping threshold, then the number of leaf nodes in S_1 and in S_2 is bounded by 4^τ. At each coupled leaf node, from Lemma 3.5, we can compute the ratio of N_1 to N_2 as the ratio of the number of satisfying assignments to Φ_{I_1} to the number of satisfying assignments to Φ_{I_2}. This can be done in time polynomial in m and 2^τ by brute force. Finally, the constraints in Definition 5.1 are all linear in the variables that represent total probability (if we treat $\frac{4}{s}$, $\frac{q'}{1-q'}$, $\frac{q''}{1-q''}$ and all ratios $\frac{N_1}{N_2}$ as given constants). Thus, we can find a valid choice of the total probability variables by linear programming. This completes the proof of the second part. □

Recall that we will be able to choose $\tau = 2c(6dk)^2 \log n$ and Theorem 3.10 will imply that at most a $1/n^c$ fraction of the distribution fails to couple. Thus, the algorithm above runs in polynomial time for any constants d and k. What remains is to show that any valid choice of total probabilities *certifies* that $q' \leq \Pr_{\mathcal{D}}[x = T] \leq q''$.

5.4 A Fractional Matching to Certify q

We are now ready to prove the second main theorem of this section. We will show that having any two one-sided stochastic decision trees that meet the constraints on the leaves imposed by Theorem 5.2 is enough to certify that $\Pr_{\mathcal{D}}[x = T]$ is approximately between q' and q''. This result will rest on two facts. Fix any assignment A. Then, either

(1) The assignment has too many clauses that restricted to marked variables are all set to F or
(2) The total probability of A reaching a leaf node u where the coupling procedure failed to terminate before reaching size τ is at most $O(\frac{1}{n^c})$.

Theorem 5.3. *Suppose that every clause contains between k and $6k$ variables and that $\log d \geq 5 \log k + 20$ and $k \geq 50 \log d + 50 \log k + 250$. Then any two one-sided stochastic decision trees S_1 and S_2 that meet the constraints on the leaves imposed by Theorem 5.2 and satisfy $\tau = 20c(6dk)^2 \log n$ imply that*

$$q' - O\left(\frac{1}{n^c}\right) \leq Pr_{\mathcal{D}}[x = T] \leq q'' + O\left(\frac{1}{n^c}\right).$$

The proof of this theorem will use many of the same tools that appeared in the proof of Theorem 3.10, since in essence we are performing a one-sided charging argument.

Proof. The proof will proceed by constructing a complete bipartite graph $H = (U, V, E)$ and finding a fractional approximate matching as follows. The nodes in U represent the satisfying assignments of Φ with $x = T$. The nodes in V represent the satisfying assignments of Φ with $x = F$. Moreover, all but a $O(\frac{1}{n^c})$ fraction of the nodes on the left will send between $1 - q'' - O(\frac{1}{n^c})$ and $1 - q' + O(\frac{1}{n^c})$ flow along their outgoing edges. Finally, all but a $O(\frac{1}{n^c})$ fraction of the nodes on the right will receive between $q' - O(\frac{1}{n^c})$ and $q'' + O(\frac{1}{n^c})$ flow along their incoming edges.

First, notice that any assignment A (say with $x = T$) is mapped by S_1 to a distribution over leaf nodes (both coupled and uncoupled). Now consider matched pairs of leaf nodes (according to Claim 4) that correspond to a coupling. Let p_1 and p_2 be the total probability of the leaf nodes in S_1 and S_2, respectively. Let N_1 and N_2 be the total number of assignments that are consistent with $\mathcal{A}_1$ and $\mathcal{A}_2$, and let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the corresponding sets of assignments. From the assumption that

$$\left(\frac{q'}{1-q'}\right)p_2N_2 \leq p_1N_1 \leq \left(\frac{q''}{1-q''}\right)p_2N_2$$

and the intermediate value theorem, it follows that there is a $q' \leq q^* \leq q''$, which satisfies

$$\left(\frac{q'^*}{1-q^*}\right)p_2N_2 = p_1N_1.$$

Hence, there is a flow that sends exactly $(1 - q^*)p_1$ units of flow out of each node in $\mathcal{N}_1$ and exactly q^*p_2 units of flow into each node in $\mathcal{N}_2$.

If every leaf node is coupled, then we would indeed have the fractional matching we are looking for, just by summing these flows over all leaf nodes. What remains is to handle uncoupled leaf nodes. Consider any such leaf node u in S_1 and the corresponding leaf node v in S_2. From Lemma 3.9, we have that there is a 3-tree T of size at least $10c \log n$. For each node in T, from Claim 2 we have there are at least $10c \log n$ disjoint type 1 or type 2 errors.

Case # 1: Suppose that there are at least $3.5c \log n$ disjoint type 1 errors. Fix the 3-tree T, and look at all root-to-leaf paths that are consistent with just the type 1 errors. Then the sum of their total probabilities is at most

$$\left(\frac{4}{d^{11}}\right)^{3.5c \log n}.$$

This follows, because the constraint that $z_2 \leq (\frac{4}{s})z$ (and similarly for z_4) in Definition 5.1 implies that for each path, we can factor out the above term corresponding to just the decisions where there are type 1 errors. Moreover, we chose $s = d^{11}$ exactly as we did in the proof of Theorem 3.10. The remaining probabilities are conditional distributions on the paths (after having taken into account the type 1 errors) and sum to at most one. Finally, the total number of 3-trees of size $10c \log n$ containing x is at most $(3e(6dk)^3)^{10c \log n}$. Thus, for any assignment A, if we ignore what happens to it when it ends up at a leaf node that did not couple and that has at least $3.5c \log n$ disjoint type 1 errors, in total we have ignored at most

$$\left(\frac{4}{d^{11}}\right)^{3.5c \log n} (3e(6dk)^3)^{10c \log n} \leq 1/n^c$$

of its probability, where the last inequality uses the fact that $\log d \geq 5 \log k + 20$.

Case # 2: Suppose that there are at least $6.5c \log n$ disjoint type 2 errors. Each type 2 error can be blamed on either $\mathcal{A}_1$ or $\mathcal{A}_2$ or both (e.g., it could be that the clause c might only have all of its marked variables set to F in $\mathcal{A}_1$). Let us suppose that the assignment A contributes at least $2.5c \log n$ disjoint type 2 errors. In this case, we will completely ignore A in the constraints imposed by our flow. How many such assignments can there be? The probability of getting any such assignment is bounded by

$$\left(\left(\frac{49}{100}\right)^{k/4}\right)^{2.5c \log n} \left(3e(6dk)^3\right)^{10c \log n} \leq 1/n^c,$$

where the last inequality has used the fact that $k \geq 50 \log d + 50 \log k + 250$.

Thus, if we ignore the flow constraints for all such assignments, then we will be ignoring at most a $1/n^c$ fraction of the nodes in U and the nodes in V. The only remaining case is when the assignment A ends up at a leaf node u that has at least $6.5c \log n$ disjoint type 2 errors, but it contributes less than $2.5c \log n$ itself. For each type 2 error that it does not contribute to, it contributes to another type 1 error. The only minor complication is that the node responsible might not be in the 3-tree T. However, it is distance at most 1 from the 3-tree, because it is contained in a clause that results in type 2 error that does contain a node in T. Now, by analogous reasoning to Case #1, if we fix the pattern of these type 1 errors—i.e., we fix the 3-tree and the extra nodes at distance 1 from it that contribute the missing type 1 errors—then the sum of the total probability of all consistent root-to-leaf paths is at most

$$\left(\frac{4}{d^{11}}\right)^{4c \log n}.$$

Now, the number of patterns can be bounded by $(4e(6kd)^4)^{10c \log n}$, which accounts for the inclusion of extra nodes that are not in T. Once again, for such an assignment A, if we ignore what happens to it when it ends up at a leaf node that did not couple and has at least $6.5c \log n$ disjoint type 2 errors but it contributes less than $2.5c \log n$ itself, then in total we have ignored at most

$$\left(\frac{4}{d^{11}}\right)^{4c \log n} (4e(6kd)^4)^{10c \log n} \leq 1/n^c$$

of its probability, where the last inequality uses the fact that $\log d \geq 5 \log k + 20$.

Now returning to the beginning of the proof and letting N_1 and N_2 be the total number of satisfying assignments with $x = T$ and $x = F$, respectively. We have that the flow in the bipartite graph implies

$$(1 - q'')N_1 - O\left(\frac{1}{n^c}\right) \leq \text{flowout}_U = \text{flowin}_V \leq q''N_2 + O\left(\frac{1}{n^c}\right)$$

and the further condition

$$q'N_2 - O\left(\frac{1}{n^c}\right) \leq \text{flowin}_V = \text{flowout}_U \leq (1-q')N_1 + O\left(\frac{1}{n^c}\right),$$

which gives $\frac{q'}{1-q'} - O(\frac{1}{n^c}) \leq \frac{q}{1-q} \leq \frac{q''}{1-q''} + O(\frac{1}{n^c})$, which completes the proof of the theorem. □

6 APPLICATIONS

Here, we show how to use our algorithm for computing marginal probabilities when k is logarithmic in d for approximate counting and sampling from the uniform distribution on satisfying assignments. Throughout this section, we will assume that each clause contains between k and $2k$ variables and make use of Theorem 5.3 as a subroutine, which makes a weaker assumption that the number of variables per clause is between k and $6k$.

6.1 Approximate Counting

There is a standard approach for how to use an algorithm for computing marginal probabilities to do approximate counting in a monotone CNF, where no variable is negated (see, e.g., Reference [4]). Essentially, we fix an ordering of the variables $x_1, x_2, \ldots, x_n$ and a sequence of formulas $\Phi_1, \Phi_2, \ldots, \Phi_n$. Let $\Phi_1 = \Phi$ and let Φ_i be the subformula we get when substituting $x_1 = T$, $x_2 = T, \ldots, x_{i-1} = T$ into Φ and simplifying. Notice that each such formula is a monotone CNF and inherits the properties we need from Φ. In particular, each clause has at least k variables, because the only clauses left in Φ_i (i.e., not already satisfied) are the ones that have all of their variables unset.

However, such approaches crucially use monotonicity to ensure that no clause becomes too small (i.e., contains few variables but is still unsatisfied). This is a similar issue to what happened with the coupling procedure, which necessitated using *marked* and *unmarked* variables, the latter being variables that are never set and are used to make sure no clause becomes too small. We can take a similar approach here. In what follows, we will no longer assume Φ is monotone.

Lemma 6.1. *Suppose that $e(D+1) \leq (32/31)^k$. Then, there is a partial assignment $\mathcal{A}$ so that every clause is satisfied and each clause has at least $7k/8$ unset variables. Moreover, there is a deterministic algorithm to find such a partial assignment that runs in time polynomial in m, n, k, and d.*

Proof. Independently for each variable, we will set it to T with probability $1/32$, set it to F with probability $1/32$ and to leave it unset with probability $15/16$. Now consider the m bad events, one for each clause c, that c is either unsatisfied or has not enough unset variables (or both). Let $D(p||q)$ be the Kullback-Leibler divergence between Bernoulli random variables with parameters p and q, respectively. Then, we have

$$\Pr[c \text{ is bad}] \leq e^{-D(\frac{1}{8}||\frac{1}{16})k} + \left(\frac{31}{32}\right)^k \leq 2\left(\frac{31}{32}\right)^k.$$

Here, the first term follows from the Chernoff bound and represents the probability that there are not enough unset variables and the second term is the probability that the clause is unsatisfied. Moreover, using the fact that $D(\frac{1}{8}||\frac{1}{16}) \geq \frac{1}{11}$, we conclude that the second term is larger than the first. Now, we can once again appeal to the Lovász Local Lemma to show the existence of a partial assignment where none of the bad events occur. Finally, we can use the deterministic algorithm of Chandrasekaran, Goyal, and Haeupler [7] to find such a partial assignment in polynomial time. □

Theorem 6.2. *Suppose we are given a CNF formula Φ on n variables where every clause contains between k and $2k$ variables. Moreover, suppose that $\log d \geq 5 \log k + 20$ and $k \geq 60 \log d + 60 \log k + 300$. Let OPT be the number of satisfying assignments. Then there is a deterministic algorithm that*

outputs a quantity count that satisfies

$$\left(1 - \frac{1}{n^c}\right)OPT \leq count \leq \left(1 + \frac{1}{n^c}\right)OPT$$

and runs in time polynomial in m and $n^{cd^2k^2}$.

Proof. First, we (deterministically) find a partial assignment that satisfies the properties guaranteed by Lemma 6.1. Note that the conditions on k and d imply that the condition in Lemma 6.1 holds. Suppose t total variables are assigned a truth value in the partial assignment. Let $x_1, x_2, \ldots, x_t$ be an ordering of these variables. As above, we construct a sequence of subformulas $\Phi_1, \Phi_2, \ldots, \Phi_t$, where Φ_i is obtained by substituting into Φ the assignments for $x_1, x_2, \ldots, x_{i-1}$ in the partial assignment and simplifying the formula. Again let q_i be our estimate for the marginal probabilities that a random satisfying assignment of Φ_i sets x_i to true.

The key point is that the next subformula in the sequence, Φ_{t+1}, is empty, because all the clauses are satisfied by the partial assignment. Moreover, each clause that appears in any formula Φ_i for $1 \leq i \leq t$ has at most $2k$ variables and has at least $7k/8$ variables, because it has at least that many unset variables in the partial assignment. Note that the upper and lower bounds on clause sizes differ by less than a factor of 6. Moreover, we can now output

$$\text{count} \triangleq 2^{n-t} \prod_{i=1}^{n} \left(\frac{1}{q_i}\right) = \left(1 \pm \frac{1}{n^c}\right)\text{OPT},$$

because Φ_{t+1} has exactly 2^{n-t}, satisfying assignments (every choice of the unset variables) and we have used the same telescoping product, but now to compute the ratio of the number of satisfying assignments to Φ_{t+1} divided by the number of satisfying assignments to Φ. □

6.2 Approximate Sampling

Here, we give an algorithm to generate an assignment approximately uniformly from the set of all satisfying assignments. Again, the complication is that our oracle for approximating the marginals works only if k is at least logarithmic in d, so we need some care in the order we choose to sample variables. In Algorithm 3, we give the algorithm.

First, we prove that the output is close to uniform.

Lemma 6.3. *If the oracle* $\mathcal{Y}$ *outputs a marginal probability that is* $1/n^{c+1}$ *close to the true marginal distribution for each variable queried, then the output of the* Sampling Procedure *is a random assignment whose distribution is* $1/n^c$*-close in total variation distance to the uniform distribution on all satisfying assignments.*

Proof. The proof of this lemma is in two parts. First, imagine we were instead given access to an oracle $\mathcal{W}$ that answered each query for a marginal distribution with the exact value. Then, each variable set using the oracle is chosen from the correct marginal distribution. And in the last step, the set of satisfying assignments is a cross-product of the satisfying assignments for each component. Thus, the procedure would output a uniformly random assignment from the set of all satisfying assignments. Second, since at most n variables are queried, we have that with probability at least $1 - 1/n^c$ all of the random decision of the procedure would be the same if we had given it answers from $\mathcal{W}$ instead of from $\mathcal{Y}$. This now completes the proof. □

The key step in the analysis of this algorithm rests on showing that with high probability each connected component is of logarithmic size.

ALGORITHM 3: Sampling Procedure,
Input: CNF Φ, oracle $\mathcal{Y}$ for approximating marginals of variables

```
(1)  Using Lemma 3.1, label variables as marked or unmarked
(2)  While there is a marked variable x that is unset
(3)      Sample x using 𝒴
(4)      Initialize V_I = {x} and V_O to be all unset variables (x is already set)
(5)      While there is a clause c with variables in both V_I and V_O
(6)          Sequentially sample its marked variables (if any) using 𝒴
(7)          Case # 1: c is satisfied
(8)              Delete c
(9)          Case # 2: c is unsatisfied
(10)             Let S be all variables in c (marked or unmarked)
(11)             Update V_I ← V_I ∪ S, V_O ← V_O \ S
(12)     End
(13) End
(14) For each connected component of the remaining clauses
(15)     Enumerate and uniformly choose a satisfying assignment of the unset variables
(16) End
```

Theorem 6.4. *Suppose we are given a CNF formula Φ on n variables where each clause contains between k and $2k$ variables. Moreover, suppose that $\log d \geq 5 \log k + 20$ and $k \geq 60 \log d + 60 \log k + 300$. There is an algorithm that outputs a random assignment whose distribution is $1/n^c$-close in total variation distance to the uniform distribution on all satisfying assignments. Moreover, the algorithm runs in time polynomial in m and $n^{cd^2k^2}$.*

Proof. The proof of this theorem uses many ideas from the coupling procedure as analyzed in Section 3. Let Φ' be the formula at the start of some iteration of the inner WHILE loop. Then, at the end of the inner WHILE loop, using Lemma 3.5, we can write

$$\Phi' = \Phi'_I \wedge \Phi'_O,$$

where Φ'_I is a formula on the variables in V_I and Φ'_O is a formula on the variables in V_O. In particular, no clause has variables in both because the inner WHILE loop terminated. Now, we can appeal to the analysis in Theorem 3.10, which gives a high probability bound on the size of V_I. The analysis presented in its proof is nominally for a different procedure, the Coupling Procedure, but the inner WHILE loop of the Sampling Procedure is identical except for the fact that there are no type 1 errors, because we are building up just one assignment. Thus,

$$\Pr[|V_I| \geq 2(6dk)^2 t] \leq \left(\frac{1}{2}\right)^t.$$

The inner WHILE loop is run at most n times, so if we choose $t \geq c \log n$, we get that with probability at least $1 - 1/n^c$ no component has size larger than $2c(6dk)^2 \log n$. Now, the brute-force search in the last step can be implemented in time polynomial in m and $n^{cd^2k^2}$, which, combined with Lemma 6.3, completes the proof. □

We can also now prove Corollary 1.4.

Proof. Recall, we are given a cause network and the truth assignment of each observed variable. First, we do some preprocessing. If an observed variable is an OR of several hidden variables or their negation and the observed variable is set to F, then we know the assignment of each hidden variable on which it depends. Similarly, if an observed variable is an AND and it is set to T, again

we know the assignment of each of its variables. For all the remaining observed variables, we know there is exactly one configuration of its variables that is prohibited so each yields a clause in a CNF formula Φ. Moreover, each clause depends on at least $7k/8$ variables whose truth value has not been set, because the collection of observations is k-regular. Finally, each variable is contained in at most d clauses. The posterior distribution on the remaining hidden variables (whose value has not already been set) is uniform on the set of satisfying assignments to Φ and, thus, we can appeal to Theorem 6.4 to complete the proof. □

7 CONCLUSION

In this article, we presented a new approach for approximate counting in bounded degree systems based on bootstrapping an oracle for the marginal distribution. In fact, our approach seems to extend to non-binary approximate counting problems as well. For example, suppose we are given a set of hyperedges and our goal is to color the vertices `red`, `green`, or `blue` with the constraint that every hyperedge has at least one of each color. It is still true that the uniform distribution on satisfying colorings is locally close to the uniform distribution on all colorings in the sense of Corollary 2.3. This once again allows us to construct a coupling procedure, but now between a triple of partial colorings. The coupling can be used to give alternative ways to sample a satisfying coloring uniformly at random, which in turn yields a method to certify the marginals on any vertex by solving a polynomial number of counting problems on logarithmic sized hypergraphs. We chose to work with only binary counting problems to simplify the exposition, but it remains an interesting question to understand the limits of the techniques that we introduced here.

ACKNOWLEDGMENTS

We thank Elchanan Mossel and David Rolnick for many helpful discussions at an earlier stage of this work.

REFERENCES

[1] D. Achlioptas and F. Iliopoulos. 2016. Random walks that find perfect objects and the Lovász Local Lemma. *J. ACM* 63, 3 (2016), 22.
[2] N. Alon. 1991. A parallel algorithmic version of the local lemma. *Rand. Struct. Alg.* 2, 4 (1991), 367–378.
[3] J. Beck, 1991. An algorithmic approach to the Lovász Local Lemma. *Rand. Struct. Alg.* 2, 4 (1991), 343–365.
[4] I. Bezáková, A. Galanis, L. A. Goldberg, H. Guo, and D. Štefankovič. 2016. Approximation via correlation decay when strong spatial mixing fails. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP'16)*, 1–13.
[5] M. Bordewich, M. Dyer, and M. Karpinski. 2006. Stopping times, metrics, and approximate counting. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP'06)*, 108–119.
[6] G. Bresler, E. Mossel, and A. Sly. 2013. Reconstruction of Markov random fields from samples: Some observations and algorithms. *SIAM J. Comput.* 42, 2 (2013), 563–578.
[7] K. Chandrasekaran, N. Goyal, and B. Haeupler. 2013. Deterministic algorithms for the Lovász Local Lemma. *SIAM J. Comput.* 42, 6 (2013), 2132–2155.
[8] P. Dagum and M. Luby. 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artific. Intel.* 60 (1993), 141–153.
[9] P. Dagum and M. Luby. 1997. An optimal approximation algorithm for Bayesian inference. *Artific. Intel.* 93, 1-2 (1997), 1–27.
[10] P. Erdös and L. Lovász. 1975. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and Finite Sets*. North Holland, 609–627.
[11] A. Galanis, D. Štefankovič, and E. Vigoda. 2016. Inapproximability of the partition function for the antiferromagnetic Ising and hard-core models. *Combin., Prob. Comput.* 25, 4 (2016), 500–559.
[12] D. Gamarnik and D. Katz. 2012. Correlation decay and deterministic FPTAS for counting colorings of a graph. *J. Disc. Alg.* 12 (2012), 29–47.
[13] H. Guo, M. Jerrum, and J. Liu. 2017. Uniform sampling through the Lovász Local Lemma. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'17)*, 342–355.

[14] H. Guo, C. Liao, P. Lu, and C. Zhang. 2018. Counting hypergraph colourings in the local lemma regime. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'18)*, 926–939.
[15] B. Haeupler, B. Saha, and A. Srinivasan. 2011. New constructive aspects of the Lovász Local Lemma. In *J. ACM* 58, 6 (2011), 28.
[16] D. Harris and A. Srinivasan. 2017. A constructive algorithm for the Lovász Local Lemma on permutations. In *Theor . Comput.* 13, 1 (2017), 1–41.
[17] N. Harvey, P. Srivastava, and J. Vondrák. 2018. Computing the independence polynomial in Shearer's region for the LLL. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, 1557–1576.
[18] N. Harvey and J. Vondrák. 2015. An algorithmic proof of the Lovász Local Lemma via resampling oracles. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'15)*, 1327–1346.
[19] J. Hermon, A. Sly, and Y. Zhang. 2016. Rapid mixing of hypergraph independent set. *ArXiv:1610.07999.*
[20] M. Jerrum, L. Valiant, and V. Vazirani. 1986. Random generation of combinatorial structures from a uniform distribution. *Theoret. Comput. Sci.* 43 (1986), 169–188.
[21] D. Knuth. 1969. *The Art of Computer Programming*, Vol I. Addison Wesley, London, p. 396 (exercise 11).
[22] V. Kolmogorov. 2016. Commutativity in the algorithmic Lovász Local Lemma. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'16)*, to appear.
[23] J. Liu and P. Lu. 2015. FPTAS for counting monotone CNF. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, 1531–1548.
[24] R. Moser and G. Tardos. 2010. A constructive proof of the Lovász Local Lemma. *J. ACM* 57, 2 (2010), 1–15.
[25] C. Nair and P. Tetali. 2007. The correlation decay (CD) tree and strong spatial mixing in multi-spin systems. *ArXiv:0701494.*
[26] A. Sinclair and M. Jerrum. 1989. Approximately counting, uniform generation and rapidly mixing Markov chains. *Inform. Comput.* 82 (1989), 93–133.
[27] A. Sinclair, P. Srivastava, and M. Thurley. 2014. Approximation algorithms for two-state anti-ferromagnetic spin systems. *J. Stat. Phys.* 155, 4 (2014), 666–686.
[28] A. Sly. 2010. Computational transition at the uniqueness threshold. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'10)*, 287–296.
[29] A. Sly and N. Sun. 2014. Counting in two-spin models on d-regular graphs. *Ann. Prob.* 42, 6 (2014), 2383–2416.
[30] D. Weitz. 2006. Counting independent sets up to the tree threshold. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'06)*, 140–149.

Received November 2017; revised October 2018; accepted November 2018