Adaptive Hashing for Model Counting

Jonathan Kuck[1], **Tri Dao**[1], **Shengjia Zhao**[1], **Burak Bartan**[1], **Ashish Sabharwal**[2], and **Stefano Ermon**[1]
[1]Stanford University [2]Allen Institute for Artificial Intelligence
{kuck,trid,sjzhao,bbartan,ermon}@stanford.edu, ashishs@allenai.org

Abstract

Randomized hashing algorithms have seen recent success in providing bounds on the model count of a propositional formula. These methods repeatedly check the satisfiability of a formula subject to increasingly stringent random constraints. Key to these approaches is the choice of a fixed family of hash functions that strikes a good balance between computational efficiency and statistical guarantees for a hypothetical worst case formula. In this paper we propose a scheme where the family of hash functions is chosen adaptively, based on properties of the specific input formula. Akin to adaptive importance sampling, we use solutions to the formula (found during the bounding procedure of current methods) to estimate properties of the solution set, which guides the construction of random constraints. Additionally, we introduce an orthogonal variance reduction technique that is broadly applicable to hashing based methods. We empirically show that, when combined, these approaches lead to better lower bounds on existing benchmarks, with a median improvement factor of 2^{13} over 1,198 propositional formulas.[1]

1 INTRODUCTION

Propositional model counting is the problem of counting the number of satisfying solutions to a Boolean expression. Exact model counting is computationally intractable, which has led to the development of approximation schemes. An effective line of work performs approximate model counting using randomized hashing schemes [Gomes et al., 2006, Chakraborty et al., 2013a, Ermon et al., 2014, Ivrii et al., 2015, Achlioptas and Jiang, 2015, Belle et al., 2015, Achlioptas et al., 2018, Soos and Meel, 2019]. These methods reduce the #P-complete model counting problem [Valiant, 1979] to a polynomial number of NP queries of Boolean satisfiability (SAT), which can benefit from decades of research in combinatorial reasoning and fast SAT solvers [Biere et al., 2009, Soos and Meel, 2019]. The fundamental idea is to approximate the number of elements in a set by recursively splitting the set in half using a hash function until only a single element is left. We focus on improving the hash functions at the heart of these methods.

As an illustrative example, consider a 100 by 100 grid of chess boards. Each board contains 8 by 8 cells, where each cell may or may not contain a piece. The maximum number of pieces present across all boards is 64×100^2. We want to know the number of pieces across all boards, but directly counting them is computationally expensive. To obtain a good estimate of the number of chess pieces it is sufficient to be able to (a) independently remove each piece with probability $1/2$, thus removing half the pieces in expectation, and (b) check whether at least one piece is still on any board. Note that in the problems we consider pieces are 'removed' implicitly, so this operation is much easier than counting. Suppose we apply operation (a) followed by operation (b) repeatedly, until operation (b) informs us after round $m + 1$ that there are no longer any pieces remaining. Then 2^m is a good estimate of the number of pieces across all boards. However, it would be very tedious to independently remove each piece with probability $1/2$ during operation (a), representing (a lack of) *computational efficiency*. Instead we might be tempted to perform our algorithm considering only pieces on black squares, black squares in even rows, or even a single randomly chosen square on each chess board. We would then scale our answer by the fraction of pieces considered, e.g. multiply it by 2 when only considering pieces on black squares. This would be more computationally efficient, reducing our work in expecta-

[1]Code at https://github.com/ermongroup/adaptive_hashing

tion by factors of 2, 4, and 64 respectively, but represents a degradation in *statistical efficiency*. These approaches will give good estimates if the pieces are distributed uniformly across all boards, but will perform very poorly on certain configurations. If pieces are placed only on white squares then the first scheme will completely fail. If pieces are only placed on a small fraction of the chess boards, then the third scheme's performance will degrade. When applied to an unknown configuration, these computationally efficient methods will yield weaker statistical bounds on the true number of pieces.

In the context of propositional model counting, we are given a boolean expression and each satisfying assignment is equivalent to a chess piece. Analogously, we remove satisfying assignments using hash functions. Improving hash function performance is challenging because of the inherent tension between computational and statistical efficiency. Hash functions are commonly implemented by appending parity constraints to the original boolean expression. It has been shown that reducing the length of parity constraints (i.e., the number of variables involved) improves computational efficiency at the cost of statistical efficiency, a trade-off that improves practical performance in many cases [Ermon et al., 2014, Zhao et al., 2016, Achlioptas and Theodoropoulos, 2017, Achlioptas et al., 2018]. In particular, regular constraints inspired by low-density parity check codes have recently been shown to perform extremely well in practice [Achlioptas and Theodoropoulos, 2017, Achlioptas et al., 2018]. Combined with improvements in SAT solving technology specifically tailored to parity constraints [Soos and Meel, 2019], these advances have led to dramatic scalability improvements.

A key limitation of all existing hashing-based approaches, however, is that the family of hash functions is defined *a priori*, and the same family is used for every input formula with the same number of variables. Theoretical guarantees are provided showing that the family will perform well regardless of the "shape" of the solution set of the input formula. The family of hash functions plays the same role as the proposal distribution in importance sampling [Hadjis and Ermon, 2015]. Existing hashing approaches analogously use a fixed proposal distribution with good worst-case guarantees, i.e., a proposal distribution that is expected to work well for inference on many target distributions.

Inspired by adaptive importance sampling, we explore the idea of *designing hash functions that are tailored to the properties of each input propositional formula*. To this end, we note that existing approaches solve a sequence of SAT instances obtained from the input formula by adding an increasing number of parity constraints. Many of these instances are satisfiable, and in this case a SAT solver will also return a solution. These solutions are currently discarded, however, they have unexploited value. Our main insight is that these solutions are approximately uniform samples from the set of all possible solutions [Gomes et al., 2007], and can therefore be used to infer important properties of the solution set (e.g., marginals) without additional overhead.

We propose an adaptive method to improve the construction of random parity constraints using approximate samples found during the search procedure used in Achlioptas and Theodoropoulos [2017] and Achlioptas et al. [2018]. Intuitively, we use estimated marginals to encourage the hash functions to partition the solution set into hash buckets as evenly as possible. In the chess board analogy, each solution is represented by a chess piece. We choose which computationally efficient method to use based on the chess pieces that have been previously found. If the original scheme is performed repeatedly, but pieces are only found on black squares, we could choose to only consider pieces on black squares and scale our result by a factor of 2. On the other hand, if pieces are only found on a small fraction of the boards, then we could choose to only consider these boards and appropriately scale our result. While our approach is in principle compatible with various hash families, we focus on regular/biregular matrices inspired by LDPC codes because of their excellent performance [Achlioptas and Theodoropoulos, 2017, Achlioptas et al., 2018].

Additionally, we propose an orthogonal variance reduction technique to improve accuracy at the cost of extra computation. This approach can be used with any hash function construction and SAT solver. The key idea is that by performing multiple repetitions and using a suitable aggregation strategy, one can provably reduce the variance of the estimate of the model count, and thus tighten bounds. The repetitions are independent, which allows them to be performed in parallel. We demonstrate empirically that the combination of our variance reduction technique and adaptive strategy can lead to dramatic improvements in both lower bound tightness and computation speed. On an example problem instance, which was selected due to its computational difficulty, we demonstrate dramatic improvements. We compute a lower bound using extremely sparse regular constraint matrices that is tighter, by over a factor of 2^{100}, than the bound computed using biregular constraint matrices with 1.5 times the density. In addition, our bound can be computed over 100x faster than the denser biregular bound, using a simple parallelization procedure across 10 cores.

Algorithm 1 Known Lower Bound

Inputs: s: Solution cutoff
Δ: Failure Probability
$\{\mathcal{A}^m\}_{m=1}^n$: Fixed distributions over parity matrices $\mathbb{F}_2^m$
$\mathcal{O}_S$: A SAT oracle
Output: A probabilistic lower bound on $|S|$

1: $T = \lceil 8 \log \frac{1}{\Delta} \rceil$
2: $m = 1$
3: **while** $m \leq n$ **do**
4: **for** $t = 1, \cdots, T$ **do**
5: Sample $A^m \sim \mathcal{A}^m$, denote $h^m(x) = A^m x$
6: Sample $b \sim \text{Uniform}(\mathbb{F}_2^m)$
// Invoke oracle $\mathcal{O}_S$ up to s times to check
// whether the input formula with additional
// constraints $A^m x = b$ has at least s distinct
// solutions
7: $w^t = \min\left\{s, |S \cap (h^m)^{-1}(b)|\right\}$
8: **if** $\sum_{t=1}^T w^t < sT/2$ **then**
9: break
10: $m = m + 1$
11: Output "$|S| \geq s\lfloor 2^{m-3} \rfloor$"

Algorithm 2 New, Adaptive Lower Bound

Inputs: s: Solution cutoff
Δ: Failure probability
K: Number of repetitions per trial
$\mathcal{O}_S$: A SAT oracle
Output: A probabilistic lower bound on $|S|$

1: $T = \lceil 8 \log \frac{1}{\Delta} \rceil$
2: $m = 1$
3: $\mathcal{D} = \emptyset$ // Solutions found so far
4: **while** $m \leq n$ **do**
5: **for** $t = 1, \cdots, T$ **do**
6: **for** $k = 1, \cdots, K$ **do**
7: $A^m = \text{SampleAdaptiveMatrix}(\mathcal{D}, m)$
8: Sample $b \sim \text{Uniform}(\mathbb{F}_2^m)$
// Invoke oracle $\mathcal{O}_S$ up to sK times to check
// whether the input formula with additional
// constraints $A^m x = b$ has at least sK
// distinct solutions
9: $w_k = \min\left\{sK, |S \cap (h^m)^{-1}(b)|\right\}$
10: $S' =$ Up to sK solutions obtained at step 9
11: $\mathcal{D} = \mathcal{D} \cup S'$
12: $w^t = \min\left\{s, \frac{1}{K}\sum_{k=1}^K w_k\right\}$
13: **if** $\sum_{t=1}^T w^t < sT/2$ **then**
14: break
15: $m = m + 1$
16: Output "$|S| \geq s\lfloor 2^{m-3} \rfloor$"

2 BACKGROUND

Model Counting by Hashing. Let $x_1, \cdots, x_n$ be n binary variables, and let $S \subseteq \mathbb{F}_2^n = \{0,1\}^n$ be a high-dimensional set with size up to 2^n. Addition and multiplication are defined modulo 2 for $\mathbb{F}_2 = \{0,1\}$. The set S is usually defined by conditions that its elements must satisfy, such as through a boolean expression. An NP oracle can be used to determine whether S is empty. Membership in S of a specific $x \in \mathbb{F}_2^n$ can be tested by evaluating the boolean expression. We would like to estimate $|S|$, the number of elements in S. When S is the set of solutions to a boolean expression over n binary variables, the problem of computing $|S|$ is known as model counting, which is the canonical #P complete problem [Valiant, 1979].

The hashing approach [Gomes et al., 2006, Chakraborty et al., 2013b, Ermon et al., 2013] reduces counting to solving a polynomial number of NP-complete SAT problems. As a primitive operation, the estimation procedure randomly partitions S into 2^m bins, selects one of these bins, and checks whether S has at least s elements in this bin. Checking whether s elements of S belong to this bin can be done with a query to an NP oracle, e.g., invoking a SAT solver. Performing this primitive operation over a range of m values gives an estimate of $|S|$. Repeating this estimation procedure a small number of times allows us to approximate $|S|$ well by giving both lower and upper bounds that hold with high probability. This result is notable because solving counting problems (in #P) is believed to be significantly harder than solving decision problems (in NP) [Valiant, 1979].

A Hashing Algorithm for Model Counting. In this section we formally define the model counting algorithm based on hashing. There are several variants of the algorithms [Gomes et al., 2006, Chakraborty et al., 2013a, Ermon et al., 2014, Ivrii et al., 2015, Achlioptas and Jiang, 2015, Belle et al., 2015, Achlioptas et al., 2018, Soos and Meel, 2019] based on the same core idea. For ease of exposition we present the simplest variant in Algorithm 1. We note that more efficient versions are possible, e.g., replacing the linear search in Line 3 with binary search [Chakraborty et al., 2016] or doubling binary search [Achlioptas and Theodoropoulos, 2017]. The methods we introduce can be applied to more efficient variants of Algorithm 1, as in our experiments.

We denote a hash function that maps all elements $x \in \mathbb{F}_2^n$ to one of 2^m bins as $h^m : \mathbb{F}_2^n \to \mathbb{F}_2^m$ (where $m \leq n$). It has m components $h^m = (h_1^m, \cdots, h_m^m)$, where each h_i^m maps x to one of two bins.

The hash function h^m is implemented using parity constraints. That is, $h^m(x) = A^m x$, where A^m is a binary matrix in $\mathbb{F}_2^{m \times n}$ and addition and multiplication are done

modulo 2. Each row of A is an individual parity constraint, with the i-th row corresponding to h_i^m, that maps every $x \in \mathbb{F}_2^n$ to one of 2 bins. Combined, the parity constraint matrix maps every $x \in \mathbb{F}_2^n$ to one of 2^m bins. At each iteration, the algorithm samples A^m from some distribution $\mathcal{A}^m$, which defines the hash function h^m (although this dependence is not explicit in our notation). We will return to the choice of $\mathcal{A}^m$ in the next section.

Given the hash function h^m we can inspect the 2^m hash bins. If most of the hash bins contain at least k elements from S, then it is likely that $|S| \geq k2^{m-1}$. More precisely, we uniformly sample b from $\mathbb{F}_2^m$ and invoke the SAT oracle $\mathcal{O}_S$ to exactly bound the size of the set

$$S(h^m) \equiv S \cap (h^m)^{-1}(b). \tag{1}$$

$|S(h^m)|$ will be a random variable (as A^m is randomly sampled from $\mathcal{A}^m$, and b is randomly sampled from $\mathrm{Uniform}(\mathbb{F}_2^m)$). If we can decide with high probability that $|S(h^m)| \geq k$ (i.e. $\Pr[|S(h^m)| \geq k] \geq 1/2$), then we obtain a probabilistic lower bound on the size of S. This claim is formalized in the following proposition.

Proposition 1. *Let any distribution $\mathcal{A}_m$ over $\mathbb{F}_2^{m \times n}$, $\Delta \in (0,1)$, and $s \geq 1$ be inputs to Algorithm 1. The output of Algorithm 1 is correct with probability at least $1-\Delta$, where the probability is with respect to the random choices made by the algorithm. (Proof in Appendix.)*

When the linear search in Alg. 1 is replaced with the doubling binary search from Achlioptas and Theodoropoulos [2017, p. 4], m is iterated over in descending order. This may lead to up to $\log_2 n$ tests of invalid lower bounds, and one must guarantee that none of these tests incorrectly verifies the bound. In this case, setting $T = \left\lceil 8 \ln \frac{\log_2 n}{\Delta} \right\rceil$ and using a union bound guarantees the same overall correctness probability of at least $1 - \Delta$.

Statistical vs. Computational Efficiency. The correctness of Algorithm 1 does not depend on $\mathcal{A}$ according to Proposition 1. However, the tightness of the lower bound depends critically on the choice of $\mathcal{A}$. Suppose the true size of S is 2^m, then we know that $\mathbf{E}[|S(h^m)|] = 1$. However, $\Pr[|S(h^m)| = 1]$ could still be close to 0. This will happen if there are a few very large bins (i.e., $|S \cap (h^m)^{-1}(b)|$ is very large for a few values of b), while most of the bins are empty. Suppose we run Algorithm 1 with such a family of hash functions and it outputs "$|S| > s\lfloor 2^{j-3} \rfloor$". This bound will likely be very loose, that is $|S| \gg s\lfloor 2^{j-3} \rfloor$. In contrast, for an ideal hash function, each bin would have exactly one element, that is $\Pr[|S(h^m)| = 1] = 1$. In this case our algorithm would conclude that 2^{m-2} is a lower bound (with $s = 1$). We could get a perfect 2^m lower bound with small modifications to the algorithm, but this is not essential in our paper, as we focus on hard problems with lower bounds that are usually loose by hundreds of orders of magnitude. In general, we would like $\mathbf{Var}[|S(h^m)|]$ to be as close to zero as possible. $\mathbf{Var}[|S(h^m)|]$ depends on the choice of $\mathcal{A}$. Therefore, selecting a good $\mathcal{A}$ is crucial.

Early work used a parity matrix A sampled from an i.i.d. Bernoulli distribution with probability $1/2$ [Valiant and Vazirani, 1986, Gomes et al., 2006]. In expectation each row of A contains $n/2$ non-zero elements. In other words, half of the variables in x are involved in each parity constraint $\sum_{j=1}^{n} A_{ij}x_j = b_j$. A will act as a pairwise independent hash function with this construction. The hashed value of one element, Ax^1, will give no information about the hashed value of any other element, Ax^2, for any two elements $x^1, x^2 \in \mathbb{F}_2^n$. In turn, this yields a very small $\mathbf{Var}[|S(h)|]$ because the hash function acts almost independently. However, implementing this construction results in SAT problems that are computationally challenging to solve. Gomes et al. [2007], Ermon et al. [2014], Zhao et al. [2016], Achlioptas and Theodoropoulos [2017] reduced the length of these constraints, achieving dramatic computational speedups at the cost of degradation in statistical efficiency (usually measured by increased variance $\mathbf{Var}[|S(h)|]$).

As the length of parity constraints decreases, the selection of which specific variables are included in each constraint becomes increasingly critical. The statistical efficiency of short parity constraints can be improved by creating dependencies between constraints [Achlioptas and Theodoropoulos, 2017, Achlioptas et al., 2018]. Specifically, the parity constraint matrix A is required to be biregular, with the same number of 1's in every row and column. However, this improvement is still insufficient for a large fraction of benchmark problems. We propose a new class of methods that impose additional structure on which variables may group together in constraints based on knowledge of the solution set S shape, thus further increasing statistical efficiency. For example, "frozen" variables, which always take the same value across all satisfying assignments, are useless when included in parity constraints. Similarly, variables that almost always take the same value have little utility. Our approach is to balance the quality of parity constraints, ensuring that all constraints contain some variables that we believe to be useful, while also distributing variables believed to be less useful uniformly among constraints.

The two key contributions of this paper are modifications to Algorithm 1 that are shown in our Algorithm 2:

1. We construct hash functions *adaptively*, based on information about the specific set shape (line 7).

2. We introduce a general variance reduction tech-

nique. Instead of adding a single set of parity constraints to the SAT problem of interest and solving the resulting problem, we aggregate solutions to K problems resulting from modifying the SAT problem of interest with K independently sampled hash functions (for loop on line 6).

3 ADAPTIVE HASHING

Optimal Constraints for Statistical Efficiency. Intuitively, the reason short parity constraints usually lead to poor statistical efficiency is that the constraint fails to split the solution set S into equally sized parts. More precisely, $\mathbf{Var}[|S(h)|]$ is smallest (in fact, it is zero) if $|S \cap (h^m)^{-1}(b)|$ is the same for all $b \in \mathbb{F}_2^m$ and for all h^m in the family of hash functions. To hypothetically create such a family of hash functions, we consider constructing each hash function in the family by sequentially adding m constraints through the following idealized (albeit impossible) process.

1) Given the set S we first pick a single parity constraint, $h_1(x) = \sum_j A_{1j}^m x_j$, that splits S into 2 equally sized subsets. That is, exactly half of the elements in S satisfy $h_1(x) = 0$ and the other half satisfy $h_1(x) = 1$.

2) Given S and h_1 we add a second parity constraint, $h_2(x) = \sum_j A_{2j}^m x_j$, that splits the 2 current subsets into 4 equally sized subsets. That is, for each subset $S \cap h_1^{-1}(0)$ and $S \cap h_1^{-1}(1)$, exactly half of the elements satisfy $h_2(x) = 0$ and the other half satisfy $h_2(x) = 1$.

3) Continue adding constraints that equally split all current subsets until we have 2^m equally sized subsets.

The family of hash functions, defined as all hash functions that this process can construct, will have $\mathbf{Var}[|S(h)|] = 0$. Even though this idealized family of hash functions is impossible to construct in practice (especially using short length constraints), it suggests another interesting trade-off: improved statistical efficiency from hash functions that split a particular set evenly vs. the additional computation required to identify these hash functions. Even when we fix the length of each constraint to a set number of variables, we can still find good hash functions by choosing which variables we include in each constraint. Finding better constraints requires computational cost, but leads to improved statistical efficiency. There is potentially a large spectrum of algorithms exploring the optimal trade-offs.

Adaptive Biregular Constraints. In this section we show how to adaptively construct biregular constraint matrices utilizing knowledge about the shape of a particular set S. The biregular constraint matrices from Achlioptas and Theodoropoulos [2017] obtain state of the art performance for short constraints. Combined with our adaptive approach we see further improvements.

In this paper, we focus on a very cheap approximation to the idealized constraints of Section 3, with the desirable property that it can be used to adaptively restrict the family of biregular matrices. A constraint h_i that splits $S \cap h_1^{-1}(b_1) \cap \cdots \cap h_{i-1}^{-1}(b_{i-1})$ in half for every $(b_1, \cdots, b_{i-1})$ also split S in half. Therefore, constraints that do not split S in half cannot be optimal. While it is difficult to generate optimal, short constraints, it is relatively easy to avoid generating certain types of constraints that split the entire set into two highly imbalanced subsets.

Our approach is to restrict the family of hash functions by avoiding these parity constraints that are likely to be very poor for the particular set S. A constraint $h_i(x) = \sum_j A_{ij}^m x_j$ is useless when it fails to split the set S. That is, $P_S(h_i(x) = 1) \approx 1$ or $P_S(h_i(x) = 1) \approx 0$, where P_S is a uniform distribution on the elements in S. One scenario where this will occur is when each variable in h_i has marginal probability close to 0 or 1 under P_S, i.e. $P_S(x_u = 1) \approx 1$ or $P_S(x_u = 1) \approx 0$ for all u such that $A_{iu} = 1$. Our adaptive construction avoids such constraints. In general, variables with marginals close to 0 or 1 are useless in the sense that they do not improve the ability of a constraint to evenly split the set. On the other hand, variables with marginals closer to $1/2$ are generally more useful in constraints (note that this is a rough statement, as it ignores the potentially complicated dependencies between variables in the set).

Following this intuition, our adaptive approach ensures that all constraints contain at least some 'high quality' variables. We assign variables to constraints in a round-robin fashion. Beginning with the m variables whose marginals are closest to $1/2$, we randomly assign one variable to each of the m constraints. We repeat this process for groups of m variables with marginals successively farther from $1/2$. If the density of the constraint matrix is set such that variables appear in it more than once, this procedure will loop through the variables repeatedly, always traversing 'high quality' to 'low quality' groups. In this manner, every constraint is guaranteed to contain some 'high quality' variables while no constraint will contain only 'low quality' variables. This procedure is formalized in Algorithm 3.[2]

When the constraint density is set to $1/m$, so that each variable appears in exactly one constraint, we can visual-

[2]For ease of exposition we assume $(n \mod m) = 0$ and that d is an integer. When $(n \mod m) \neq 0$, a block of variables will wrap from the n-th to the first variable. When d is not an integer, the loop over b will break partway through and the last block may contain fewer than m variables.

Algorithm 3 SampleAdaptiveMatrix

Inputs: $\mathcal{D}$: Set of previously found SAT solutions
m: Number of constraints to sample
d: The constraint matrix will contain dn one-entries
Output: A constraint matrix $A \in \mathbb{F}_2^{m \times n}$.

// Empirical variable marginals
$p_\ell = |\{x \in \mathcal{D} \mid x_\ell = 1\}|/|\mathcal{D}|$, for $\ell = 1, \cdots, n$
// Uniformity scores
$s_\ell = |p_\ell - 1/2|$, for $\ell = 1, \cdots, n$
$(v_1, \cdots, v_n) =$ arg-sort smallest first $(s_1, \cdots, s_n)$
$A = 0^{m \times n}$
for d times **do**
  for $b = 0, \cdots, n/m - 1$ **do**
    repeat
      $i = bm + 1$
      $j = i + m - 1$
      $(v'_1, \cdots v'_b) = \text{permute}(v_i, \cdots, v_j)$
    until $A[l, v'_l] == 0$ for all $l \in \{1, \cdots, b\}$
    $A[l, v'_l] = 1$ for all $l \in \{1, \cdots, b\}$
Return A

ize the constraint sampling procedure of Algorithm 3 as a three step process:

1. Sort the variables $x_1, \cdots, x_n$ by $|p_i - 1/2|$ and re-index the variables in sorted order. For the rest of the section we assume that $x_1, \cdots, x_n$ are ordered so that $|p_1 - 1/2| < |p_2 - 1/2| < \cdots < |p_n - 1/2|$.

2. Construct an $m \times n$ block diagonal matrix as below:

$$\left[\begin{array}{ccccc|ccccc|c|cc} 1 & 0 & . & 0 & 0 & 1 & 0 & . & 0 & 0 & . & 1 & 0 \\ 0 & 1 & . & 0 & 0 & 0 & 1 & . & 0 & 0 & . & 0 & 1 \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & . & 1 & 0 & 0 & 0 & . & 1 & 0 & . & 0 & 0 \\ 0 & 0 & . & 0 & 1 & 0 & 0 & . & 0 & 1 & . & 0 & 0 \end{array}\right]$$

   Note that variables with estimated marginals closest to $1/2$ appear in the left most block while variables with estimated marginals furthest from $1/2$ appear in the right most block.

3. Randomly permute the columns of each submatrix of A_0.

Approximately Uniform Samples for Free. One snag with our adaptive approach is the problem of computing probabilities, P_S, of the uniform distribution over elements in S. Our approach is to estimate these probabilities using approximate samples from the distribution. Prior work has shown that SAT solvers draw just such *approximately uniform samples* [Gomes et al., 2007] when solving problems modified by long parity constraints. The main advantage of this approach is that there is no overhead involved, as the SAT solver is already invoked during the bounding procedure (line 7 in Algorithm 1) — we simply have to record the solutions found so far (lines 10 and 11) instead of discarding them as current methods do. We lose guarantees on the quality of these samples because our parity constraints are short, but still gain valuable information about the shape of the set [Gomes et al., 2007]. In our experiments we observe that our construction has very small overhead, but potentially large gains in performance.

4 VARIANCE REDUCTION VIA MULTIPLE REPETITIONS

The first and second moments of the random variables $|S(h)|$ play a key role in the analysis of hashing-based algorithms. In practice, the statistical performance depends crucially on how concentrated $|S(h)|$ is around its expected value. Specifically, performance is poor (bounds are loose) if most realizations of $S(h)$ are empty (no solutions survive after adding the random constraints), but there are a few unlikely, very large realizations (all solutions survive). Our procedure for adaptively sampling constraint matrices aims to improve hashing performance by reducing the variance of $|S(h)|$. In this section we present an orthogonal approach for reducing variance, which can be combined with our adaptive procedure.

Specifically, variance can always be reduced by averaging over multiple, independent realizations of $|S(h)|$. Averaging always preserves the expected value, and is *guaranteed* to reduce variance. Define $\overline{S}^m$ to be the mean of K independent realizations of $|S(h^m)|$. Then variance is reduced to $\mathbf{Var}[\overline{S}^m] = \mathbf{Var}[|S(h^m)|]/K$. Algorithm 2 achieves this variance reduction by calculating a lower bound on $\overline{S}^m$ in line 12. Note that the min operation on line 9 serves as an optimization, because calling the SAT solver is unnecessary once it has found sK solutions, and removing this min results in a mathematically equivalent algorithm. This reduction in variance leads to tighter lower bounds output by Algorithm 2, as we will see in the experiments. In contrast, note that increasing the number of trials, T, in Algorithm 1 will increase the probability that the output is correct, but even in the limit of $T \to \infty$ might not give a tight bound. Further, the probability that the tighter bound of Algorithm 2 is valid (see Proposition 2) remains the same as for Algorithm 1, because the two modifications—using adaptive matrices and multiple repetitions—do not significantly affect the proof of Proposition 1.

Remark 1. It is worth noting the subtle differences between Algorithm 2 and apparently similar modifica-

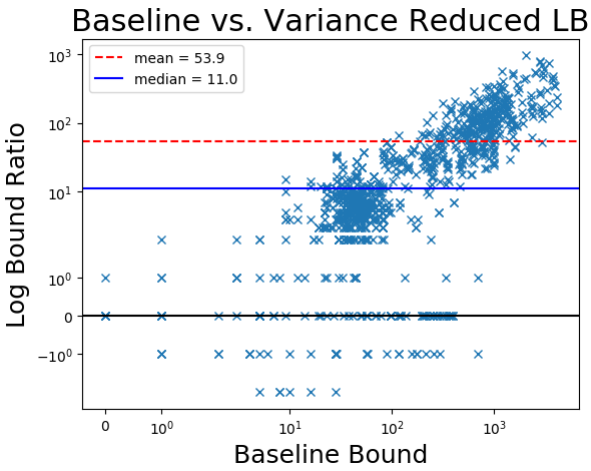

Figure 1: The x-axis shows the baseline bound. The y-axis shows $\log_2$ of the ratio of our variance reduced bound computed with $K = 10$ solutions to the baseline bound. Each point represents one problem and above the black line denotes improvement. In addition to the log ratio, data is displayed with a symmetric log scale.

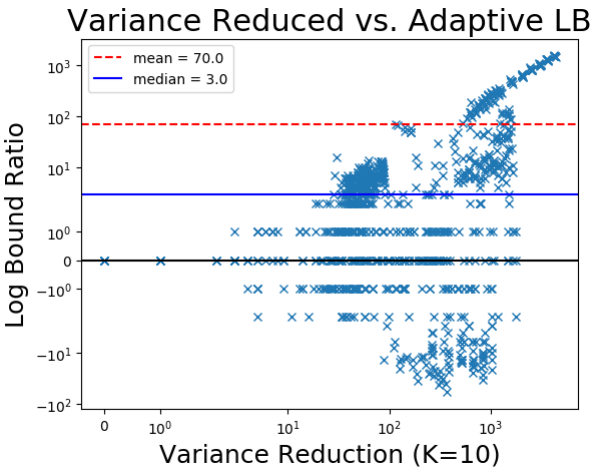

Figure 2: The x-axis shows our variance reduced bound computed with $K = 10$ solutions. The y-axis shows $\log_2$ of the ratio of our adaptive, variance reduced bound computed with $K = 10$ solutions to our variance reduced bound computed with $K = 10$ solutions. Data is displayed with a symmetric log scale.

tions to Algorithm 1. Replacing s in Algorithm 1 with $s' = sK$ results in a seemingly similar algorithm. However, the break condition on the while loop over m would compute the sum of only T random variables w^t, as opposed to TK random variables w_k in Algorithm 2. This difference would result in much earlier stopping and a looser bound relative to Algorithm 2. Replacing T in Algorithm 1 with $T' = TK$ would give a bound that holds with higher probability. However, this modification would only utilize s solutions per problem instance, losing the ability of Algorithm 2 to smooth between instances containing many and few solutions. This smoothing is critical for variance reduction.

Proposition 2. *Let $s \geq 1$, $K \geq 1$, and $\Delta \in (0, 1)$ be inputs to Algorithm 2. The output of Algorithm 2 is correct with probability at least $1-\Delta$, where the probability is with respect to the random choices made by the algorithm. (See Appendix for a proof.)*

Upper Bounds. While we have confined our discussion to lower bounds until this point, randomized hashing methods can also give *upper bounds* on the exact model count. While lower bounds can be derived for very general families of hash functions, upper bounds generally place more restrictions on the types of hash functions that can be used [Ermon et al., 2014]. For example, consider a slight variant of Algorithm 1 that outputs the upper bound "$|S| \leq s2^{m+1}$" (Algorithm 4 in the appendix). Here, the family of hash functions must have provably low variance with $16\sigma_m^2 < \mu_m^2$, where $\sigma_m^2 \geq \mathbf{Var}[|S(h^m)|]$ is a computable upper bound on the variance and $\mu_m = E[|S(h^m)|]$. While families of dense hash functions satisfy this condition, it places a restriction on the use of sparse hash functions with higher variance. However, employing our variance reduction strategy with $K > 1$ repetitions allows us to relax this condition to $16\sigma_m^2 < K\mu_m^2$ while maintaining correctness of the algorithm. Increasing the number of repetitions K has two primary effects on the upper bound. First, it allows for the use of families of hash functions with larger variance, stated in Theorem 1. Second, it reduces the number of trials, T, required for the upper bound to hold with a specified probability (line 4 in Algorithm 4). This formulation is similar to the upper bounding strategy in Achlioptas et al. [2018], cf. Lemma 1 and Theorem 2.

Theorem 1. *Let $s \geq 1$, $K \geq 1$, and $\Delta \in (0, 1)$ be inputs to Algorithm 4. In addition, let $\{\mathcal{A}^m\}_{m=1}^n$ be a set of distributions over parity matrices which defines a family of uniform hash functions with bounded variance. For each distribution define the upper bound, σ_m^2, on the variance of its associated hash function, satisfying $\mathbf{Var}[|S(h^m)|] \leq \sigma_m^2 < K\mu_m^2/16$. Then, with probability at least $1 - \Delta$, the output of Algorithm 4 is correct, where the probability is w.r.t the random choices made by the algorithm. (See Appendix for a proof.)*

5 EXPERIMENTS

We evaluated the performance of our proposed method on the suite of benchmarks used in Soos and Meel

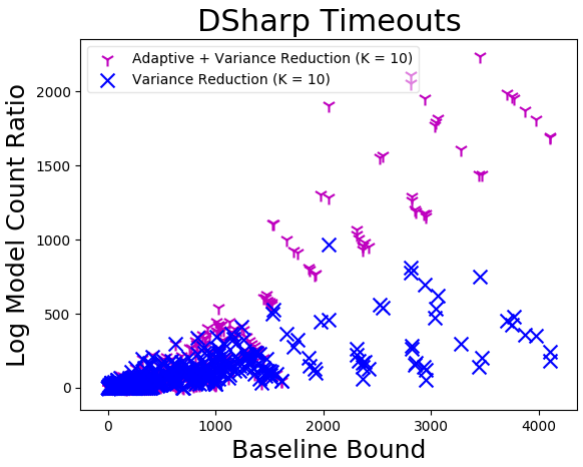

Figure 3: Instances where DSharp did not finish within 5,000 seconds. The x-axis shows the lower bound given by the baseline regular construction. The y-axis shows $\log_2$ of the ratio of the lower bound given by one of our proposed improvements to the baseline lower bound.

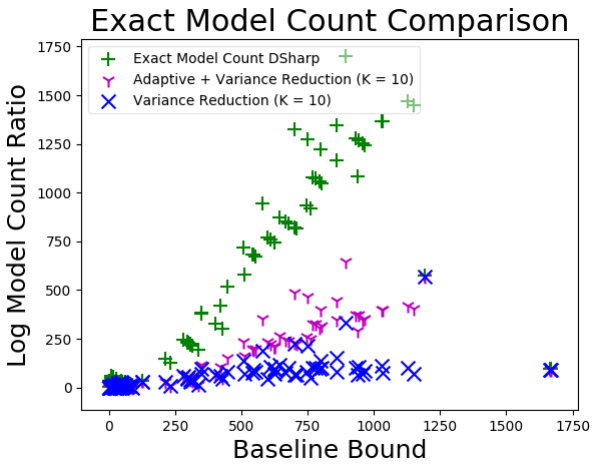

Figure 4: The x-axis shows the lower bound computed by the baseline regular construction. The y-axis shows $\log_2$ of the ratio of either the exact model count (DSharp) or the lower bound by one of our improvements to the baseline lower bound.

[2019]. These problems were taken from a range of application areas including probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, program synthesis, functional synthesis, and logistics. Of these 1,896 benchmarks, the exact model counter DSharp [Muise et al., 2012] solved 698 within 2 seconds. We evaluated our methods on the remaining 1,198 non-trivial instances.

Thanks to recent improvements in the underlying SAT solver, CryptoMiniSat [Soos and Meel, 2019], used by all randomized hashing frameworks, the range of problem instances that can be solved within a guaranteed multiplicative factor has dramatically increased. Soos and Meel [2019] report that ApproxMC3 is able to solve 1,140 of the problem instances in the data set we use within a timeout of 5,000 seconds, notably overtaking DSharp which was only able to solve 1,001 (we note that in experiments on our cluster, DSharp only solved 955 problems within this time limit). However, there still exist many problems that are beyond the reach of these approaches, with 756 in the data set. In our experiments we aggressively target increasing the range of SAT problems for which we can provide some information. As a baseline we compute a probabilistic lower bound using doubling binary search with state of the art, biregular constraint matrices, as in Algorithm 2 from Achlioptas and Theodoropoulos [2017, p. 4]. These constraint matrices have been shown to excel at giving tight bounds with short constraints. In our baseline we further reduce constraint length beyond what was considered in Achlioptas and Theodoropoulos [2017], such that every variable appears in exactly one constraint and constraints have balanced lengths (forming a regular constraint matrix). Such ultra-short constraints are very efficient to solve but result in loose bounds.

Our proposed methods can be used to significantly tighten these bounds without significantly increasing runtime. We computed lower bounds for our methods and the baseline using doubling binary search, a failure probability of $\Delta = .05$, and imposed a 5,000 second timeout on the cumulative time of all calls to the SAT solver.[3] We imposed the same 5,000 second timeout across all calls to the SAT solver when evaluating our variance reduction technique, rather than giving this technique access to parallel computation.

Figure 1 compares our variance reduction technique with the baseline, regular construction. Using $K = 10$ solutions, the median improvement in the lower bound was a factor of 2^{11}. The baseline construction computed bounds for all but 100 problems within the time limit, while our variance reduction technique computed bounds for all but 200 problems. With parallel computation this number would likely have been much closer to 100.

Figure 2 shows the performance of our variance reduction technique when combined with our adaptive construction compared to our variance reduction technique

[3]We did not include time spent in our python script during the bounding procedure, as this code is not optimized and the calls to the SAT solver limit computational efficiency for hard problems. For example, when computing the adaptive, regular bound (with the smallest density) in Figure 5, the time spent by our python script generating constraints is roughly 5% of the time spent calling the SAT solver.

alone. We estimated marginals and performed adaptation using only the samples naturally acquired during the search process (with $K = 10$). The median improvement in the lower bound was a factor of 8. In addition, many problem instances saw dramatic improvements, up to a factor of 2^{1000}. When our variance reduction technique was combined with our adaptive construction, 209 problem instances timed out.

We ran DSharp (using infinite precision numbers) on all problem instances with a timeout of 5,000 seconds. Figure 4 shows how each method compares with the exact model counts provided by DSharp on 242 problems DSharp was able to solve within this time budget (excluding the 698 problems it could solve within 2 seconds and 15 problem instances where the model count returned by DSharp was large than 2^{5000}, as these Figure 3 shows how our methods compared with the baseline on the 732 problem instances that DSharp timed out on, but our methods completed. The general trends in improvement between the baseline and our variance reduction method and between our variance reduction method and our adaptive construction combined with variance reduction appear similar between problem instances that DSharp did and did not time out on.

There is a trade-off between the statistical and computational efficiency of hash functions. Using a higher density of ones improves statistical efficiency but hurts computational efficiency. By reducing the density of the biregular matrix construction from Achlioptas and Theodoropoulos [2017], we were able to dramatically improve computational efficiency at the cost of statistical efficiency, resulting in weaker bounds. Since our variance reduction technique requires additional (although highly parallel) computation, it is natural to ask how the resulting bound compares with denser, biregular constructions. We chose a computationally challenging problem instance (90-34-3-q) and show bound improvement and computational requirements when the density of biregular constraints is varied in Figure 5. Constraint density is varied from variables appearing in 1.0 to 1.7 constraints on average. We plot $\log_2$ of the bound obtained by using variance reduction alone, variance reduction combined with adaptation methods, and the baseline. We set $K = 10$ for both our methods. We show both sequential and naive parallel runtimes (time used by the SAT solver). The naive parallel runtime was simulated by sequentially solving each of the $K = 10$ SAT instances during a search step but recording the solver time at that step as the maximum of the 10 runtimes. Note that with true parallelization and proper integration with the SAT solver this time could be significantly reduced, by terminating all computation as soon as the required number of solutions are found. We see that combining

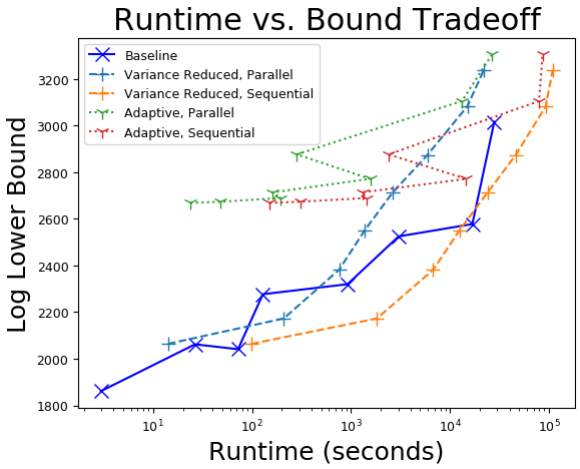

Figure 5: Tradeoff between runtime and lower bound for problem instance 90-34-3-q, selected for its difficulty.

our two strategies can reduce runtime by orders of magnitude while simultaneously computing a bound that is tighter by many orders of magnitude. The zig-zag pattern is likely caused by doubling binary search [Achlioptas et al., 2018, p. 6], where random failures during the final verification stage may result in both a significant increase in computation time and weaker lower bound.

6 CONCLUSIONS

Randomized hashing algorithms have emerged as a leading approach to approximate model counting. The performance of these algorithms depends crucially on the statistical and computational efficiency of the hash functions used. We introduced a novel adaptive construction for hash functions which utilizes the solutions to satisfiable instances that are found during the search for a bound. We also introduced a general variance reduction technique for tightening lower bounds at the expense of additional, but highly parallel, computation. When combined, these two approaches lead to improved lower bounds across a broad range of benchmarks. While we focused on a version of the regular constraint matrices from Achlioptas and Theodoropoulos [2017] with even shorter constraints, the general idea is likely to be applicable to other randomized constructions. In future work, it would be interesting to explore alternative strategies that leverage the information contained in the approximate samples obtained from the SAT solver.

Acknowledgements. Research supported by NSF (#1651565, #1522054, #1733686), ONR (N00014-19-1-2145), AFOSR (FA9550- 19-1-0024), and FLI.

References

D. Achlioptas and P. Jiang. Stochastic integration via error-correcting codes. In *Proc. Uncertainty in Artificial Intelligence*, 2015.

D. Achlioptas and P. Theodoropoulos. Probabilistic model counting with short XORs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 3–19. Springer, 2017.

D. Achlioptas, Z. Hammoudeh, and P. Theodoropoulos. Fast and flexible probabilistic model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 148–164. Springer, 2018.

V. Belle, G. Van den Broeck, and A. Passerini. Hashing-based approximate probabilistic inference in hybrid domains. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

A. Biere, M. Heule, H. van Maaren, and T. Walsh. Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol. 185, 2009.

S. Chakraborty, K. Meel, and M. Vardi. A scalable and nearly uniform generator of SAT witnesses. In *Proc. of the 25th International Conference on Computer Aided Verification (CAV)*, 2013a.

S. Chakraborty, K. Meel, and M. Vardi. A scalable approximate model counter. In *Proc. of the 19th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 200–216, 2013b.

S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic sat calls. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 3569–3576. AAAI Press, 2016.

S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of the 30th International Conference on Machine Learning (ICML)*, 2013.

S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Low-density parity constraints for hashing-based discrete integration. In *Proc. of the 31st International Conference on Machine Learning (ICML)*, pages 271–279, 2014.

C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proc. of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 54–61, 2006.

C. P. Gomes, A. Sabharwal, and B. Selman. Near-uniform sampling of combinatorial spaces using xor constraints. In *Advances In Neural Information Processing Systems*, pages 481–488, 2007.

S. Hadjis and S. Ermon. Importance sampling over sets: A new probabilistic inference scheme. In *UAI*, pages 355–364, 2015.

A. Ivrii, S. Malik, K. S. Meel, and M. Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, pages 1–18, 2015.

C. Muise, S. A. McIlraith, J. C. Beck, and E. Hsu. DSHARP: Fast d-DNNF Compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*, 2012.

M. Soos and K. S. Meel. Bird: Engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 1 2019.

L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3): 410–421, 1979.

L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47: 85–93, 1986.

S. Zhao, S. Chaturapruek, A. Sabharwal, and S. Ermon. Closing the gap between short and long XORs for model counting. In *AAAI*, pages 3322–3329, 2016.