From Sampling to Model Counting

Carla P. Gomes[†] **Joerg Hoffmann**[‡] **Ashish Sabharwal**[†] **Bart Selman**[†]

[†]Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, U.S.A.*
{gomes,sabhar,selman}@cs.cornell.edu
[‡]University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria
joerg.hoffmann@deri.org

Abstract

We introduce a new technique for counting models of Boolean satisfiability problems. Our approach incorporates information obtained from sampling the solution space. Unlike previous approaches, our method does not require uniform or near-uniform samples. It instead converts local search sampling without any guarantees into very good bounds on the model count with guarantees. We give a formal analysis and provide experimental results showing the effectiveness of our approach.

1 Introduction

Boolean satisfiability (SAT) solvers have been successfully applied in a range of domains, most prominently, in AI planning and hardware and software verification. In these applications, the basic task is to decide whether a SAT encoding of the underlying problem domain is satisfiable or not. Given the tremendous progress in state of the art SAT solvers and their applications, researchers have become interested in pursuing other questions concerning SAT encodings to further extend the reach of SAT technology. For example, can one randomly sample from the set of satisfying assignments? Or can one count the total number of satisfying assignments? In both a formal complexity theoretic sense as well as in practice, these computational questions are much harder than "just" determining satisfiability versus unsatisfiability of a formula. Formally, counting the number of solutions is a #P-complete problem, making it complete for a complexity class at least as hard as the polynomial-time hierarchy (PH) [Toda, 1989].[1] On the positive side, efficient methods for sampling and counting would open up a wide range of new applications, e.g., involving various forms of probabilistic reasoning [Darwiche, 2005; Roth, 1996; Littman *et al.*, 2001; Park, 2002; Sang *et al.*, 2005].

Uniform sampling and model counting are closely related tasks. In particular, Jerrum, Valiant, and Vazirani [1986] showed that for many combinatorial problems, if one can sample uniformly from the set of solutions, then one can use samples to obtain a highly accurate count of the total number of solutions.[2] We will explain this strategy below. This approach was exploited in the work on `ApproxCount` [Wei and Selman, 2005], where a model sampling procedure called `SampleSat` [Wei *et al.*, 2004] was used to provide samples of satisfying assignments of the underlying SAT instance. In this setting, the counting is only approximate — with no guarantees on the accuracy — because `SampleSat` does in general not sample purely uniformly; it uses Markov Chain Monte Carlo (MCMC) methods [Madras, 2002; Metropolis *et al.*, 1953; Kirkpatrick *et al.*, 1983], which often have exponential (and thus impractical) mixing times. In fact, the main drawback of Jerrum *et al.*'s counting strategy is that for it to work well one needs (near-)uniform sampling, which is a very hard problem in itself. Moreover, biased sampling can lead to arbitrarily bad under- or over-estimates of the true count.

The key new insight of this paper is that, somewhat surprisingly, using sampling with a modified strategy, one can get very good lower-bounds on the total model count, with high confidence guarantees, *without any requirement on the quality of the sampling*. We will provide both a formal analysis of our approach and experimental results demonstrating its practical effectiveness.

Our strategy, `SampleCount`, provides provably (probabilistic) guaranteed lower-bounds on the model counts of Boolean formulas using solution sampling. Interestingly, the correctness of the obtained bounds holds even when the sampling method used is arbitrarily bad; only the quality of the bounds may go down (i.e., the bound may get farther away from the true count on the lower side). Thus, our strategy remains sound even when a heuristic-based practical solution sampling method is used instead of a true sampler.

`SampleCount` can also be viewed as a way of extending the reach of exact model counting procedures, such as `Relsat` [Bayardo Jr. and Pehoushek, 2000] and `Cachet` [Sang *et al.*, 2004].[3] More specifically, `SampleCount` first uses sampling to select a set of variables of the formula to fix. Once a sufficient number of variables have been set,

*Research supported by Intelligent Information Systems Institute (IISI), Cornell University (AFOSR grant F49620-01-1-0076) and DARPA (REAL grant FA8750-04-2-0216).

[1]The class NP is the first of an infinite sequence of levels in PH.

[2]Conversely, one can also design a uniform solution sampler using a solution counter.

[3]These exact counters also provide a lower-bound on the true model count if they are aborted before terminating.

the remaining formula can be counted using an exact model counter. From the exact residual model count and the number of fixed variables, we can then obtain a lower-bound on the total number of models. As our experiments will show, these bounds can be surprisingly close to optimal. Moreover, on many problem classes our method scales very well. To mention just one example, we consider a circuit synthesis formula from the literature, called `3bitadd_32.cnf`. This formula can be solved quite easily with a local search style SAT solver, such as `Walksat` [McAllester *et al.*, 1997]. However, it is out of reach for all but the most recent DPLL style solvers (`MiniSat` [Eén and Sörensson, 2005] takes almost two hours to find a single solution). Consequently, it is completely out of reach of exact model counters based on DPLL. The original formula has nearly 9,000 variables. `SampleCount` sets around 3,000 variables in around 30 minutes, with the remaining formula solved by `Cachet` in under two minutes. The overall lower-bound on the model count — with 99% confidence — thus obtained is an astonishing $10^{\mathbf{1339}}$ models. We see that the formula has a remarkable number of truth assignments, yet exact counters cannot find any models after over 12 hours of run time.

1.1 Main Idea

To provide the reader with an intuitive understanding of our approach, we first give a high-level description of how sampling can be used to count. We will then discuss how this can be made to work well in practice.

(a) From Sampling to Counting [Jerrum *et al.*, 1986]. Consider a Boolean formula F with M satisfying assignments. Assuming we could sample these satisfying assignments uniformly at random, we can measure the fraction of all models that have x_1 set to True, M^+, by taking the ratio of the number of assignments in the sample that have x_1 set to True over the sample size. This fraction will converge with increasing sample size to the true fraction of models with x_1 set positively, $\gamma = M^+/M$. (For now, assume that $\gamma > 0$.) It follows immediately that $M = (1/\gamma)M^+$. We will call $1/\gamma$ the "multiplier" (> 0). We have thus reduced the problem of counting the models of F to counting the models of a simpler formula, F^+. We can recursively repeat the process, leading to a series of multipliers, until all variables are assigned or until we can count the number of models of the remaining formulas with an exact counter. For robustness, one usually sets selected variable to the truth value that occurs more often in the sample. This also avoids the problem of having $\gamma = 0$ and therefore an infinite multiplier. (Note that the more frequently occurring truth value gives a multiplier of at most 2.)

(b) `ApproxCount` [Wei and Selman, 2005]. In `ApproxCount`, the above strategy is made practical by using a SAT solution sampling method called `SampleSat` [Wei *et al.*, 2004]. Unfortunately, there are no guarantees on the uniformity of the samples from `SampleSat`. Although the counts obtained can be surprisingly close to the true model counts, one can also observe cases where the method significantly over-estimates or under-estimates. Our experimental results will confirm this behavior.

(c) `SampleCount`. Our approach presented here uses a probabilistic modification of strategy (a) to obtain true accurate counts in expectation. The key strength of `SampleCount` is that it works no matter how biased the model sampling is, thereby circumventing the main drawback of approach (a). Instead of using the sampler to select the variable setting and compute a multiplier, we use the sampler only as a heuristic to determine *in what order* to set the variables. In particular, we use the sampler to select a variable whose positive and negative setting occurs most balanced in our set of samples (ties are broken randomly). Note that such a variable will have the highest possible multiplier (closest to 2) in the `SampleCount` setting discussed above. Informally, setting the most balanced variable will divide the solution space most evenly (compared to setting one of the other variables). Of course, as we noted, our sampler may be heavily biased and we therefore cannot really rely on the observed ratio between positive and negative settings of a variable. Interestingly, we can simply set the variable to a randomly selected truth value and use the multiplier 2. This strategy will still give — in expectation — the true model count. A simple example shows why this is so. Consider our formula above and assume x_1 occurs most balanced in our sample. Let the model count of F^+ be $2M/3$ and of F^- be $M/3$. If we select with probability 1/2 to set x_1 to True, we obtain a total model count of $2 \times 2M/3$, i.e., too high; but, with probability 1/2, we will set the variable to False, obtaining a total count of $2 \times M/3$, i.e., too low. Overall, our expected (average) count will be exactly M.

Technically, the expected total model count is correct because of the linearity of expectation. However, we also see that we may have significant variance between specific counts, especially since we are setting a series of variables in a row (obtaining a sequence of multipliers of 2), until we have a simplified formula that can be counted exactly. In fact, in practice, the total counts distributions (over different runs) are often heavy-tailed [Kilby *et al.*, 2006]. To mitigate the fluctuations between runs, we use our samples to select the best variables to set next. Clearly, a good heuristic would be to set such "balanced" variables first. We use `SampleSat` to get guidance on finding such balanced variables. The random value setting of the selected variable leads to an expected model count that is equal to the actual model count of the formula. We show how this property can be exploited using Markov's inequality to obtain lower-bounds on the total model count with predefined confidence guarantees. These guarantees can be made arbitrarily strong by repeated iterations of the process.

We further boost the effectiveness of our approach by using variable "equivalence" when no single variable appears sufficiently balanced in the sampled solutions. For instance, if variables x_1 and x_2 occur with the same polarity (either both positive or both negative) in nearly half the sampled solutions and with a different polarity in the remaining, we randomly replace x_2 with either x_1 or $\overline{x_1}$, and simplify. This turns out to have the same positive effect as setting a single variable, but is more advantageous when no single variable is well balanced.

Our experimental results demonstrate that in practice, `SampleCount` provides surprisingly good lower-bounds — with high confidence and within minutes — on the model counts of many problems which are completely out of reach of current exact counting methods.

2 Preliminaries

Let V be a set of propositional (Boolean) variables that take value in the set $\{0,1\}$. We think of 1 as True and 0 as False. Let F be a propositional formula over V. A solution or model of F (also referred to as a satisfying assignment for F) is a 0-1 assignment to all variables in V such that F evaluates to 1. *Propositional Satisfiability* or SAT is the decision problem of determining whether F has any models. This is the canonical NP-complete problem. In practice, one is also interested in finding a model, if there exists one. *Propositional Model Counting* is the problem of computing the number of models for F. This is the canonical #P-complete problem. It is theoretically believed to be significantly harder than SAT, and also turns out to be so in practice.

The correctness guarantee for our algorithm relies on basic concepts from probability theory. Let X and Y be two discrete random variables. The *expected value* of X, denoted $\mathbb{E}[X]$, equals $\sum_x x \Pr[X = x]$. The *conditional expectation* of X given Y, denoted $\mathbb{E}[X \mid Y]$, is a function of Y defined as follows: $\mathbb{E}[X \mid Y = y] = \sum_x x \Pr[X = x \mid Y = y]$. We will need the following two results whose proof may be found in standard texts on probability theory.

Proposition 1 (The Law of Total Expectation). *For any discrete random variables X and Y, $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$.*

Proposition 2 (Markov's inequality). *For any random variable X, $\Pr[X > k\mathbb{E}[X]] < 1/k$.*

3 Model Counting Using Solution Sampling

This section describes our sample-based model counting algorithm, `SampleCount` (see Algorithm 1). The algorithm has three parameters: t, the number of iterations; z, the number of samples for each variable setting; and α, the "slack" factor, which is a positive real number. The input is a formula F over n variables. The output is a lower-bound on the model count of F. `SampleCount` is a randomized algorithm with a high probability of success. We will shortly formalize its success probability quantitatively as Theorem 1.

`SampleCount` is formally stated as Algorithm 1 and described below. `SampleCount` performs t iterations on F and finally reports the minimum of the counts obtained in these iterations. Within an iteration, it makes a copy G of F and, as long as G is too hard to be solved by an exact model counting subroutine,[4] does the following. It calls `SampleSolutions(`G,z`)` to sample up to z solutions of G; if the sampling subroutine times out, it may return fewer than z samples.[5] Call this set of solutions S. `GetMostBalancedVar(`S`)` selects a variable u of G whose positive and negative occurrences in S are as balanced as possible, i.e., as close to $|S|/2$ each as possible. Ties are broken randomly. Similarly, `GetMostBalancedVarPair(`S`)` selects a variable pair (v,w) whose same and different occurrences are as balanced as possible. Here "same occurrence" in a solution means that either both v and w appear positively or they both appear negatively. If u is at least as balanced as (v,w), `SampleCount` uniformly randomly sets u to 0 or 1 in G. Otherwise, it uniformly randomly replaces w with either v or $\neg v$, forcing a literal equivalence. Now G is simplified by unit propagating this variable restriction, it is tested again for the possibility of exact counting, and, if necessary, the sampling and simplification process is repeated. Once G comes within the range of the exact counting subroutine after s simplification steps, the number of remaining models in G is computed. This, multiplied with $2^{s-\alpha}$, is the final count for this iteration. After t iterations, the minimum of these counts is reported as the lower-bound.

By setting variables and equivalences, the original formula is reduced in size. By doing this repeatedly, we eventually reach a formula that can be counted exactly. `SampleCount` provides a practically effective way of selecting the variables to set or the equivalence to assert. Each time it looks for the most evenly way to divide the set of remaining solutions. By picking one of the two subsets randomly, we obtain real guarantees (see Section 3.1) and the formula is eventually sufficiently simplified to enable an exact count. The sampling is used to attempt an as evenly as possible division of the solution space. The better this works, the better our lower-bound. But we don't need any guarantees on the sampling quality; we will still obtain a valid, non-trivial lower-bound.

```
Params: integers t, z; real α > 0
Input : A CNF formula F over n variables
Output: A lower-bound on the model count of F
begin
    minCount ← 2^n
    for iteration ← 1 to t do
        G ← F
        s ← 0
        until G becomes feasible for ExactModelCount do
            s ← s + 1
            S ← SampleSolutions(G, z)
            u ← GetMostBalancedVar(S)
            (v, w) ← GetMostBalancedVarPair(S)
            r ← a random value chosen uniformly from {0, 1}
            if balance(u) ≥ balance(v, w) then
                set u to r in G
            else
                if r = 0 then replace w with v in G
                else replace w with ¬v in G
            Simplify(G)
        count ← 2^(s−α) · ExactModelCount(G)
        if count < minCount then
            minCount ← count
    return Lower-bound: minCount
end
```

Algorithm 1: `SampleCount`

3.1 Correctness Analysis of `SampleCount`

We now analyze `SampleCount` as a randomized algorithm and show that the probability that it provides an incorrect lower-bound on an input formula decreases exponentially to zero with the number of iterations, t, and the slack factor, α.

[4] This is decided based on the number of remaining variables or other natural heuristics. A common strategy is to run an exact counter for a pre-defined amount of time [Wei and Selman, 2005].

[5] Even when no samples are found, the procedure can continue by selecting a random variable. Of course, the quality of the bound obtained will suffer.

Somewhat surprisingly, as we discuss below, the bound on the error probability we obtain is independent of the number z of samples used and the *quality* of the samples (i.e., how uniformly they are distributed in the solution space).

Theorem 1. *For parameters (t, z, α), the lower-bound returned by* `SampleCount` *is correct with probability at least $1 - 2^{-\alpha t}$ independent of the number and quality of solution samples.*

Proof. Let $2^{s^*}, s^* > 0$, be the true model count of F. Suppose `SampleCount` returns an incorrect lower-bound, i.e., $minCount > 2^{s^*}$. This implies that $count > 2^{s^*}$ in *all* of the t iterations. We will show that this happens in any single iteration with probability at most $2^{-\alpha}$. By probabilistic independence of the t iterations, the overall error probability would then be at most $2^{-\alpha t}$, proving the theorem.

Fix any iteration of `SampleCount`. When executing the algorithm, as variables of G are fixed or replaced with another variable within the inner (**until-do**) loop, G is repeatedly simplified and the number of variables in it is reduced. For the analysis, it is simpler to consider a closely related formula $\rho(G)$ over all n variables. At the start of the iteration, $\rho(G) = G = F$. However, within the inner loop, instead of fixing a variable u to 0 or 1 or replacing w with v or $\neg v$ and simplifying (as we did for G), we add additional constraints $u = 0$, $u = 1$, $w = v$, or $w = \neg v$, respectively, to $\rho(G)$. Clearly, at any point during the execution of `SampleCount`, the solutions of $\rho(G)$ are isomorphic to the solutions of G; every solution of G uniquely extends to a solution of $\rho(G)$ and every solution of $\rho(G)$ restricted to the variables of G is a solution of G. In particular, the number of solutions of G is always the same as that of $\rho(G)$. Further, every solution of $\rho(G)$ is also a solution of F.

Let $\widehat{G}$ denote the final formula G on which the subroutine `ExactModelCount(`G`)` is applied after s variable restrictions. Note that s itself is a random variable whose value is determined by which variables are restricted to which random value and how that propagates to simplify G so that its residual models may be counted exactly. Consider the variant of $\widehat{G}$ defined on all n variables, $\rho(\widehat{G})$. $\rho(\widehat{G})$ is a random formula determined by F and the random bits used in the iteration. Finally, the variable *count* for this iteration is a random variable whose value is $2^{s-\alpha}$ times the model count of $\rho(\widehat{G})$. We are interested in the behavior of *count* as a random variable.

Recall that every solution of $\rho(\widehat{G})$ is also a solution of F. For every solution σ of F, let Y_σ be an indicator random variable which is 1 iff σ is a solution of $\widehat{G}$. Then, $count = 2^{s-\alpha}\sum_\sigma Y_\sigma$. We will compute the expected value of *count* using the law of total expectation: $\mathbb{E}[count] = \mathbb{E}[\mathbb{E}[count \mid s]]$.

Fix s. For each σ, $\Pr[Y_\sigma = 1 \mid s]$ equals the probability that each of the s variable restrictions within the inner loop is consistent with σ, i.e., if σ has $u = 0$ then u is not restricted to 1, if σ has $v = w$ then w is not replaced with $\neg v$, etc. Because of the uniformly random value r used in the restrictions, this happens with probability exactly $1/2$ in each restriction. Note that the s restrictions set or replace s different variables, and are therefore probabilistically independent with respect to being consistent with σ. Consequently, $\Pr[Y_\sigma = 1 \mid s] = 2^{-s}$. This implies that the conditional expectation of *count* given s is $\mathbb{E}[count \mid s] = \mathbb{E}[2^{s-\alpha}\sum_\sigma Y_\sigma \mid s] = 2^{s-\alpha}\sum_\sigma \mathbb{E}[Y_\sigma \mid s] = 2^{s-\alpha}\sum_\sigma \Pr[Y_\sigma = 1 \mid s] = 2^{s-\alpha}\sum_\sigma 2^{-s} = 2^{-\alpha}\sum_\sigma 1$. Since F has 2^{s^*} solutions, we have $\mathbb{E}[count \mid s] = 2^{s^*-\alpha}$. Applying the law of total expectation, $\mathbb{E}[count] = \mathbb{E}[\mathbb{E}[count \mid s]] = \mathbb{E}\left[2^{s^*-\alpha}\right] = 2^{s^*-\alpha}$.

Finally, using Markov's inequality, $\Pr\left[count > 2^{s^*}\right] < \mathbb{E}[count]/2^{s^*} = 2^{-\alpha}$. This proves that the error probability in any single iteration is at most $2^{-\alpha}$ (in fact, strictly less than $2^{-\alpha}$). From our argument at the beginning of this proof, the overall probability of error after t iterations is less than $2^{-\alpha t}$. Since we did not use any property of the number or quality of samples, the error bound holds independent of these. □

We end with a discussion of the effect of the number of samples, z, on `SampleCount`. It is natural to expect more samples to lead to a "better" bound at the cost of a higher runtime. Note, however, that z does not factor into our formal result above. This is, in fact, one of the key points of this paper, that we provide guaranteed bounds without making any assumptions whatsoever on the quality of the sampling process or the structure of the formula. Without any such assumptions, there is no reason for more samples to guide `SampleCount` towards a better lower-bound. In the worst case, a highly biased sampler could output the same small set of solutions over and over again, making more samples futile.

However, in practice, we do gain from any sampling process that is not totally biased. It guides us towards balanced variables whose true multipliers are close to 2, which reduces probabilistic fluctuations arising from randomly fixing the selected variables. Indeed, under weak uniformity-related assumptions on the sampler and assuming the formula has a mix of balanced and imbalanced variables, a higher number of samples will reduce the variation in the lower-bound reported by `SampleCount` over several runs.

4 Experimental Results

We conducted experiments on a cluster of 3.8 GHz Intel Xeon machines with 2GB memory per node running Linux. The model counters used were `SampleCount`, `Relsat` version 2.00 with counting, `Cachet` version 1.2 extended to report partial counts, and `ApproxCount` version 1.2. Both `SampleCount` and `ApproxCount` internally use `SampleSat` for obtaining solution samples. `SampleCount` was created by modifying `ApproxCount` to ignore its sample-based multipliers, fix the most balanced variables at random, and analyze equivalences, and by creating a wrapper to perform multiple iterations (t) with the specified slack factor α.

In all our experiments with `SampleCount`, α and t were set so that $\alpha t = 7$, giving a correctness confidence of $1 - 2^{-7} = 99\%$ (see Theorem 1). t ranged from 1 to 7 so as to keep the runtimes of `SampleCount` well below two hours, while the other model counters were allowed a full 12 hours. The number of samples per variable setting, z, was typically chosen to be 20. Our results demonstrate that `SampleCount` is quite robust even with so few samples. Of course, it can be made to produce even better results with more samples of better quality, or by using a "buckets" strategy that we will

Table 1: Performance of SampleCount compared with exact counters and with an approximate counter without guarantees.

Instance	True Count	**SampleCount** (99% confidence) Models	Time	Exact Counters: Relsat Models	Time	Cachet Models	Time	ApproxCount (without guarantees) Models	Time
CIRCUIT SYNTH.									
2bitmax_6	2.1×10^{29}	$\geq \mathbf{2.4 \times 10^{28}}$	**29 sec**	2.1×10^{29}	66 sec	2.1×10^{29}	2 sec	$\approx 5.6 \times 10^{28}$	8 sec
3bitadd_32	—	$\geq \mathbf{5.9 \times 10^{1339}}$	**32 min**	—	12 hrs	—	12 hrs	$\approx 7.3 \times 10^{941}$	43 min
RANDOM k-CNF									
wff-3-3.5	1.4×10^{14}	$\geq \mathbf{1.6 \times 10^{13}}$	**4 min**	1.4×10^{14}	2 hrs	1.4×10^{14}	7 min	$\approx 8.4 \times 10^{13}$	11 sec
wff-3-1.5	1.8×10^{21}	$\geq \mathbf{1.6 \times 10^{20}}$	**4 min**	$\geq 4.0 \times 10^{17}$	12 hrs	1.8×10^{21}	3 hrs	$\approx 9.3 \times 10^{18}$	8 sec
wff-4-5.0	—	$\geq \mathbf{8.0 \times 10^{15}}$	**2 min**	$\geq 1.8 \times 10^{12}$	12 hrs	$\geq 1.0 \times 10^{14}$	12 hrs	$\approx 4.2 \times 10^{15}$	11 sec
LATIN SQUARE									
ls8-norm	5.4×10^{11}	$\geq \mathbf{3.1 \times 10^{10}}$	**19 min**	$\geq 1.7 \times 10^{8}$	12 hrs	$\geq 1.9 \times 10^{7}$	12 hrs	$\approx 2.7 \times 10^{12}$	5 sec
ls9-norm	3.8×10^{17}	$\geq \mathbf{1.4 \times 10^{15}}$	**32 min**	$\geq 7.0 \times 10^{7}$	12 hrs	$\geq 1.7 \times 10^{7}$	12 hrs	$\approx 9.5 \times 10^{17}$	11 sec
ls10-norm	7.6×10^{24}	$\geq \mathbf{2.7 \times 10^{21}}$	**49 min**	$\geq 6.1 \times 10^{7}$	12 hrs	$\geq 2.4 \times 10^{7}$	12 hrs	$\approx 2.1 \times 10^{27}$	22 sec
ls11-norm	5.4×10^{33}	$\geq \mathbf{1.2 \times 10^{30}}$	**69 min**	$\geq 4.7 \times 10^{7}$	12 hrs	$\geq 1.2 \times 10^{7}$	12 hrs	$\approx 5.1 \times 10^{40}$	1 min
ls12-norm	—	$\geq \mathbf{6.9 \times 10^{37}}$	**50 min**	$\geq 4.6 \times 10^{7}$	12 hrs	$\geq 1.5 \times 10^{7}$	12 hrs	$\approx 1.8 \times 10^{51}$	8 min
ls13-norm	—	$\geq \mathbf{3.0 \times 10^{49}}$	**67 min**	$\geq 2.1 \times 10^{7}$	12 hrs	$\geq 2.0 \times 10^{7}$	12 hrs	$\approx 4.1 \times 10^{64}$	12 min
ls14-norm	—	$\geq \mathbf{9.0 \times 10^{60}}$	**44 min**	$\geq 2.6 \times 10^{7}$	12 hrs	$\geq 1.5 \times 10^{7}$	12 hrs	$\approx 2.3 \times 10^{89}$	18 min
ls15-norm	—	$\geq \mathbf{1.1 \times 10^{73}}$	**56 min**	—	12 hrs	$\geq 9.1 \times 10^{6}$	12 hrs	$\approx 5.6 \times 10^{115}$	2 hrs
ls16-norm	—	$\geq \mathbf{6.0 \times 10^{85}}$	**68 min**	—	12 hrs	$\geq 1.0 \times 10^{7}$	12 hrs	$\approx 5.4 \times 10^{123}$	2.5 hrs
LANGFORD PROBS.									
lang-2-12	1.0×10^{5}	$\geq \mathbf{4.3 \times 10^{3}}$	**32 min**	1.0×10^{5}	15 min	1.0×10^{5}	4 hrs	$\approx 3.7 \times 10^{5}$	1.5 min
lang-2-15	3.0×10^{7}	$\geq \mathbf{1.0 \times 10^{6}}$	**60 min**	$\geq 1.8 \times 10^{5}$	12 hrs	$\geq 1.1 \times 10^{5}$	12 hrs	$\approx 7.4 \times 10^{10}$	23 min
lang-2-16	3.2×10^{8}	$\geq \mathbf{1.0 \times 10^{6}}$	**65 min**	$\geq 1.8 \times 10^{5}$	12 hrs	$\geq 1.0 \times 10^{5}$	12 hrs	$\approx 6.3 \times 10^{10}$	6 min
lang-2-19	2.1×10^{11}	$\geq \mathbf{3.3 \times 10^{9}}$	**62 min**	$\geq 2.4 \times 10^{5}$	12 hrs	$\geq 1.1 \times 10^{5}$	12 hrs	$\approx 1.2 \times 10^{14}$	12 min
lang-2-20	2.6×10^{12}	$\geq \mathbf{5.8 \times 10^{9}}$	**54 min**	$\geq 1.5 \times 10^{5}$	12 hrs	$\geq 1.0 \times 10^{5}$	12 hrs	$\approx 9.9 \times 10^{15}$	24 min
lang-2-23	3.7×10^{15}	$\geq \mathbf{1.6 \times 10^{11}}$	**85 min**	$\geq 1.2 \times 10^{5}$	12 hrs	$\geq 8.4 \times 10^{4}$	12 hrs	$\approx 1.3 \times 10^{20}$	75 min
lang-2-24	—	$\geq \mathbf{4.1 \times 10^{13}}$	**80 min**	$\geq 4.1 \times 10^{5}$	12 hrs	—	12 hrs	$\approx 1.3 \times 10^{22}$	1.5 hrs
lang-2-27	—	$\geq \mathbf{5.2 \times 10^{14}}$	**111 min**	$\geq 1.1 \times 10^{4}$	12 hrs	—	12 hrs	$\approx 1.7 \times 10^{33}$	3 hrs
lang-2-28	—	$\geq \mathbf{4.0 \times 10^{14}}$	**117 min**	$\geq 1.1 \times 10^{4}$	12 hrs	—	12 hrs	$\approx 2.3 \times 10^{26}$	2 hrs

briefly outline in Section 5. On the other hand, ApproxCount often significantly under- or over-estimated the number of solutions with 20 samples. We therefore allowed it around 100 samples per variable setting in all our runs, except for the very easy instances (circuit synthesis and random formulas) where it used 1000 samples. Other parameters of ApproxCount were set so as to obtain the desired number of samples in a reasonable amount of time from SampleSat. A local search "cutoff" between 2,000 and 20,000 was sufficient for most problems, while the Langford instances required a cutoff of 100,000 to obtain enough samples. Finally, both SampleCount and ApproxCount were set to call Cachet when typically between 50 and 350 variables remained unset.

Table 1 summarizes our main results, where we evaluate our approach on formulas from four domains: circuit synthesis, random k-CNF, Latin square, and Langford problems. We see that SampleCount scales well with problem size and provides good high-confidence lower-bounds close to the true counts, in most cases within an hour. It clearly outperforms exact counters, which almost always time out after 12 hours, providing counts that are several orders of magnitude below those of SampleCount. We also report results on ApproxCount even though the comparison is not really meaningful as ApproxCount does not provide any correctness guarantees. Indeed, it, for example, under-estimates the count by at least $10^{\mathbf{398}}$ on 3bitadd_32, and over-estimates by 10^{7} (with an increasing trend) on the Latin square formulas. (Of course, there are also classes of formulas where ApproxCount appears to count quite accurately when given good quality samples.) We discuss the results in detail below.

The circuit synthesis formulas are for finding minimal size circuits for a given Boolean function. These are known to quickly become very difficult for DPLL style procedures. The instance 2bitmax_6 is still easy for exact model counting procedures, and SampleCount also gets a very good lower-bound quickly. 3bitadd_32, on the other hand, was only recently solved for a single solution using MiniSat in about 6,000 seconds. 3bitadd_32 is certainly far beyond the reach of exact counting. The total solution count for this formula is astonishing; SampleCount reports a lower-bound of $5.9 \times 10^{\mathbf{1339}}$. Note that the formula has close to 9,000 variables and therefore the solution set is still an exponentially small fraction of the total number of assignments. SampleCount sets around 3,000 variables in around 30 min-

Table 2: Comparison of SampleCount with MBound, both with 99% confidence.

Instance	**SampleCount**		MBound		Relsat	Cachet	
	Models	Time	Models	Time	Models	Models	Time
Ramsey-20-4-5	$\geq \mathbf{3.3 \times 10^{35}}$	**3.5 min**	$\geq 1.2 \times 10^{30}$	23 min	$\geq 9.1 \times 10^{9}$	$\geq 9.0 \times 10^{11}$	12 hrs
Ramsey-23-4-5	$\geq \mathbf{1.4 \times 10^{31}}$	**53 min**	$\geq 1.8 \times 10^{19}$	25 min	$\geq 6.8 \times 10^{5}$	$\geq 8.4 \times 10^{6}$	12 hrs
Schur-5-100	$\geq \mathbf{1.3 \times 10^{17}}$	**20 min**	$\geq 2.8 \times 10^{14}$	25 min	$\geq 8.1 \times 10^{4}$	$\geq 1.0 \times 10^{14}$	12 hrs
Schur-5-140	**—**	**12 hrs**	$\geq 6.7 \times 10^{7}$	1 hr	—	—	12 hrs
fclqcolor-18-14-11	$\geq \mathbf{3.9 \times 10^{50}}$	**3.5 min**	$\geq 2.1 \times 10^{40}$	28 sec	$\geq 1.2 \times 10^{31}$	$\geq 2.4 \times 10^{33}$	12 hrs
fclqcolor-20-15-12	$\geq \mathbf{3.1 \times 10^{57}}$	**6 min**	$\geq 2.2 \times 10^{46}$	2 min	$\geq 9.0 \times 10^{27}$	$\geq 8.6 \times 10^{38}$	12 hrs

utes, with the remaining formula solved by Cachet in under two minutes. Finally, ApproxCount seriously underestimates the true count, reporting only $9.3 \times 10^{\mathbf{941}}$.

Our random formulas are selected from the under-constrained area, i.e., with clause-to-variable ratios below the SAT-UNSAT threshold. As noted by Bayardo Jr. and Pehoushek [2000], such formulas have a large number of assignments and are much harder to count than formulas of the same size near the phase transition. The table gives results on three such formulas: wff-3-150-525, wff-3-100-150, and wff-4-100-500. SampleCount comes close to the true counts within minutes, while Cachet takes up to 3 hours. ApproxCount again under-estimates the counts.

Our third domain involves the problem of counting the number of normalized Latin squares of a given order. A normalized Latin square is a Latin square with the first row and column fixed. The exact counts for these formulas are known up to order 11. We see that SampleCount scales nicely as n increases, giving good bounds in a relatively small amount of time. We used averaging over buckets of size two for better bounds (cf. Section 5). Both Relsat and Cachet consistently time out with partial or no counts. Interestingly, ApproxCount over-estimates the counts by several orders of magnitude for the harder formulas whose true count is known.

Our final domain, Langford's problem, is parameterized by two values, k and n. In our instances, $k = 2$ and the following problem is encoded: produce a sequence S of length $2n$ such that for each $i \in \{1, 2, \ldots, n\}$, i appears twice in S and the two occurrences of i are exactly i apart from each other. This problem is satisfiable only if n is 0 or 3 modulo 4. We see that SampleCount, with buckets of size 3, scales well as n increases, quickly giving good lower-bounds on the true count (which is known for some of our instances, cf. http://www.lclark.edu/~miller/langford.html). Again, ApproxCount over-estimates the counts by many orders of magnitude, while Relsat and Cachet produce significant under-counts in 12 hours of CPU time.

In Table 2, we compare the performance of SampleCount with an XOR-streamlining based model counting method that we recently proposed, called MBound [Gomes *et al.*, 2006]. These two approaches are very different in spirit. MBound was designed for challenging combinatorial problems for which even finding a single solution is often computationally difficult. SampleCount, on the other hand, is targeted towards problems for which multiple solutions can be repeatedly sampled efficiently. While MBound adds randomly chosen "XOR" constraints to the formula, potentially making it harder for SAT solvers, SampleCount adaptively eliminates variables, simplifying the formula. SampleCount has fewer parameters than MBound and is easier to use in practice. As we will see, on formulas where enough samples can be obtained easily, SampleCount outperforms MBound. However, when it is hard to find samples, MBound wins. This shows that the two techniques are complementary.

All formulas considered for the comparison in Table 2 are beyond the reach of current exact model counting methods, and are also challenging for ApproxCount (see [Gomes *et al.*, 2006] for details). We see that on both the Ramsey and the clique coloring problems, SampleCount provides much stronger lower-bounds than MBound (at 99% confidence for both approaches). For the Schur problems, SampleCount dominates on the easier instance, but is unable to sample solutions at all for the harder instance.

Finally, we demonstrate that although in expectation accurate model counts are still obtained even when samples are not used in the form of a guiding heuristic, techniques based on randomly selecting and fixing variables can suffer from a highly undesirable heavy-tail effect. Consider the following settings for SampleCount: $\alpha = 0, t = 1$, and an exact counter is called only when all variables are fixed (so that it returns either 0, inconsistency, or 1 as the residual count). We use two variations of this, the first where variables are selected at random and the second where sampling is used to select variables likely to be more balanced.

Figure 1 shows the result on the Latin square formula ls7-normalized. It plots the *cumulative average* of the number of solutions obtained using the two variants over 1,800 independent runs. The values plotted for run i are the average of the first i model counts for the two variants, respectively. Theoretically, both of these cumulative averages must converge to the true count for this formula, 1.69×10^{7} (shown as a horizontal line), after sufficiently many runs. It is clear from the figure that when balanced variables are used even by considering only 20 solution samples, the obtained model count approaches the true count significantly faster than when variables are selected at random. This fast convergence is key to SampleCount's good performance in practice.

Note that in the random variable selection case, the model count happened to start quite low in this experiment, keeping the cumulative average down. The sudden upward jumps

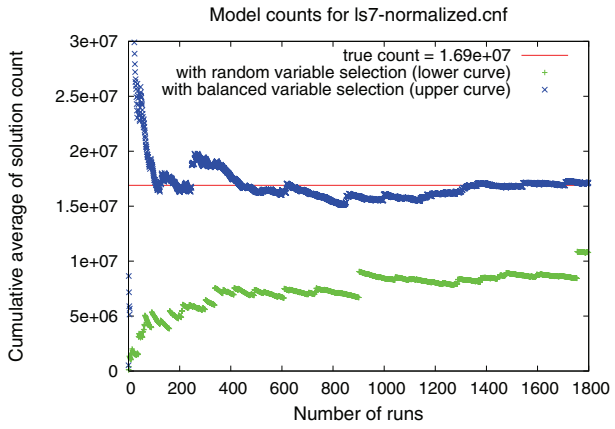

Figure 1: Convergence to the true model count: random variable selection vs. balanced variables using solution sampling

in the plot correspond to excessively high model counts that compensate for all runs up till that point and bump up the cumulative average. Here, these over-estimated counts are 2.1×10^9 in run 903 and 4.3×10^9 in run 1756, both over 100 times the true count. Extremes of this nature, and even more drastic, occur fairly often when selecting variables at random.

5 Extending SampleCount with Buckets

For randomized algorithms, more computational resources and more iterations typically result in improved behavior. While this is true for SampleCount as well, it is not immediately obvious. In fact, the more iterations we perform (i.e., the higher the t), the *worse* lower-bound on the model count we are likely to get because SampleCount takes the minimum over these t iterations. This apparent paradox is resolved by noting that as t grows, the minimum count may reduce a bit but the probability of obtaining an incorrect bound, as given by Theorem 1, converges exponentially to zero. Thus, more iterations directly translate into a much higher confidence level in the correctness of SampleCount.

SampleCount can in fact be extended so that more iterations translate into a trade-off between higher bounds and higher confidence. The idea is to perform $T = bt$ iterations and group these into t *buckets* or groups of size b each. The lower-bound in this extended version is obtained by computing the average of the b counts within each bucket and taking the minimum over these t averages.

For $b = 1$, this bucket strategy is identical to SampleCount. As b increases while t remains unchanged, we are likely to get lower-bounds even closer to the true count. To see this, fix $\alpha = 0$ so that the count obtained in each iteration is a random variable whose expected value is precisely the true count. Over different iterations, this count varies from its expected value, lowering the value of the minimum over all counts. By taking averages over several iterations within a bucket, we stabilize the individual count for each bucket, and the minimum over these stabilized counts increases. One can show that the correctness guarantee for this bucket strategy is exactly the same as Theorem 1. We leave experimental evaluation for a full version of the paper.

6 Conclusion

We presented SampleCount, a new method for model counting which capitalizes on the ability to efficiently draw (possibly biased) samples from the solution space of problems using SAT solvers. A key feature of this approach is that it gives probabilistic correctness guarantees on the obtained bounds on the solution counts without assuming anything at all about the quality (i.e., uniformity) of the sampling method used. In practice, SampleCount provides very good lower-bounds on the model counts of computationally challenging problems, and scales well as problem complexity increases.

References

[Bayardo Jr. and Pehoushek, 2000] R. J. Bayardo Jr. and J. D. Pehoushek. Counting models using connected components. In *17th AAAI*, pg. 157–162, Austin, TX, Jul 2000.

[Darwiche, 2005] A. Darwiche. The quest for efficient probabilistic inference, Jul 2005. Invited Talk, IJCAI-05.

[Eén and Sörensson, 2005] N. Eén and N. Sörensson. MiniSat: A SAT solver with conflict-clause minimization. In *8th SAT*, St. Andrews, U.K., Jun 2005. Poster.

[Gomes *et al.*, 2006] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *21th AAAI*, pg. 54–61, Boston, MA, Jul 2006.

[Jerrum *et al.*, 1986] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Comput. Sci.*, 43:169–188, 1986.

[Kilby *et al.*, 2006] P. Kilby, J. Slaney, S. Thiébaux, and T. Walsh. Estimating search tree size. In *21th AAAI*, pg. 1014–1019, Boston, MA, Jul 2006.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, D. Gelatt Jr., and M. Vecchi. Optimization by simuleated annealing. *Science*, 220(4598):671–680, 1983.

[Littman *et al.*, 2001] M. L. Littman, S. M. Majercik, and T. Pitassi. Stochastic Boolean satisfiability. *J. Auto. Reas.*, 27(3):251–296, 2001.

[Madras, 2002] N. Madras. Lectures on Monte Carlo methods. In *Field Institute Monographs*, volume 16. Amer. Math. Soc., 2002.

[McAllester *et al.*, 1997] D. A. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *AAAI/IAAI*, pg. 321–326, Providence, RI, Jul 1997.

[Metropolis *et al.*, 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phy.*, 21:1087–1092, 1953.

[Park, 2002] J. D. Park. MAP complexity results and approximation methods. In *18th UAI*, pg. 388–396, Edmonton, Canada, Aug 2002.

[Roth, 1996] D. Roth. On the hardness of approximate reasoning. *J. AI*, 82(1-2):273–302, 1996.

[Sang *et al.*, 2004] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *7th SAT*, Vancouver, B.C., Canada, May 2004. Online Proceedings.

[Sang *et al.*, 2005] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *20th AAAI*, pg. 475–482, Pittsburgh, PA, Jul 2005.

[Toda, 1989] S. Toda. On the computational power of PP and $\oplus$P. In *30th FOCS*, pg. 514–519, 1989.

[Wei and Selman, 2005] W. Wei and B. Selman. A new approach to model counting. In *8th SAT*, volume 3569 of *LNCS*, pg. 324–339, St. Andrews, U.K., Jun 2005.

[Wei *et al.*, 2004] W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In *19th AAAI*, pg. 670–676, San Jose, CA, Jul 2004.