Leveraging Belief Propagation, Backtrack Search, and Statistics for Model Counting

Lukas Kroc   Ashish Sabharwal   Bart Selman

Department of Computer Science
Cornell University, Ithaca NY 14853-7501, U.S.A.
{kroc,sabhar,selman}@cs.cornell.edu *

Abstract. We consider the problem of estimating the model count (number of solutions) of Boolean formulas, and present two techniques that compute estimates of these counts, as well as either lower or upper bounds with different trade-offs between efficiency, bound quality, and correctness guarantee. For lower bounds, we use a recent framework for probabilistic correctness guarantees, and exploit message passing techniques for marginal probability estimation, namely, variations of Belief Propagation (BP). Our results suggest that BP provides useful information even on structured loopy formulas. For upper bounds, we perform multiple runs of the `MiniSat` SAT solver with a minor modification, and obtain statistical bounds on the model count based on the observation that the distribution of a certain quantity of interest is often very close to the normal distribution. Our experiments demonstrate that our model counters based on these two ideas, `BPCount` and `MiniCount`, can provide very good bounds in time significantly less than alternative approaches.

1 Introduction

The model counting problem for Boolean satisfiability or SAT is the problem of computing the number of solutions or satisfying assignments for a given Boolean formula. Often written as #SAT, this problem is #P-complete [21] and is widely believed to be significantly harder than the NP-complete SAT problem, which seeks an answer to whether or not the formula in satisfiable. With the amazing advances in the effectiveness of SAT solvers since the early 90's, these solvers have come to be commonly used in combinatorial application areas like hardware and software verification, planning, and design automation. Efficient algorithms for #SAT will further open the doors to a whole new range of applications, most notably those involving probabilistic inference [1, 4, 12, 14, 17, 19].

A number of different techniques for model counting have been proposed over the last few years. For example, `Relsat` [2] extends systematic SAT solvers for model counting and uses component analysis for efficiency, `Cachet` [18] adds caching schemes to this approach, `c2d` [3] converts formulas to the d-DNNF form

* Research supported by IISI, Cornell University (AFOSR grant FA9550-04-1-0151), DARPA (REAL Grant FA8750-04-2-0216), and NSF (Grant 0514429).

which yields the model count as a by-product, `ApproxCount` [23] and `SampleCount` [9] exploit sampling techniques for estimating the count, `MBound` [10] relies on the properties of random parity or XOR constraints to produce estimates with correctness guarantees, and the recently introduced `SampleMinisat` [8] uses sampling of the backtrack-free search space of systematic SAT solvers. While all of these approaches have their own advantages and strengths, there is still much room for improvement in the overall scalability and effectiveness of model counters.

We propose two new techniques for model counting that leverage the strength of message passing and systematic algorithms for SAT. The first of these yields probabilistic lower bounds on the model count, and for the second we introduce a statistical framework for obtaining upper bounds.

The first method, which we call `BPCount`, builds upon a successful approach for model counting using local search, called `ApproxCount`. The idea is to efficiently obtain a rough estimate of the "marginals" of each variable: what fraction of solutions have variable x set to TRUE and what fraction have x set to FALSE? If this information is computed accurately enough, it is sufficient to recursively count the number of solutions of only *one* of $F|_x$ and $F|_{\neg x}$, and scale the count up appropriately. This technique is extended in `SampleCount`, which adds randomization to this process and provides lower bounds on the model count with high probability correctness guarantees. For both `ApproxCount` and `SampleCount`, true variable marginals are estimated by obtaining several solution samples using local search techniques such as `SampleSat` [22] and computing marginals from the samples. In many cases, however, obtaining many near-uniform solution samples can be costly, and one naturally asks whether there are more efficient ways of estimating variable marginals.

Interestingly, the problem of computing variable marginals can be formulated as a key question in Bayesian inference, and the Belief Propagation or BP algorithm [15], at least in principle, provides us with exactly the tool we need. The BP method for SAT involves representing the problem as a factor graph and passing "messages" back-and-forth between variable and factor nodes until a fixed point is reached. This process is cast as a set of mutually recursive equations which are solved iteratively. From the fixed point, one can easily compute, in particular, variable marginals.

While this sounds encouraging, there are two immediate challenges in applying the BP framework to model counting: (1) quite often the iterative process for solving the BP equations does not converge to a fixed point, and (2) while BP provably computes exact variable marginals on formulas whose constraint graph has a tree-like structure (formally defined later), its marginals can sometimes be substantially off on formulas with a richer interaction structure. To address the first issue, we use a "message damping" form of BP which has better convergence properties (inspired by a damped version of BP due to Pretti [16]). For the second issue, we add "safety checks" to prevent the algorithm from running into a contradiction by accidentally eliminating all assignments.[1] Somewhat

[1] A tangential approach for handling such fatal mistakes is incorporating BP as a heuristic within backtrack search, which our results suggest has clear potential.

surprisingly, avoiding these rare but fatal mistakes turns out to be sufficient for obtaining very close estimates and lower bounds for solution counts, suggesting that BP does provide useful information even on highly structured loopy formulas. To exploit this information even further, we extend the framework borrowed from `SampleCount` with the use of biased coins during randomized value selection.

The model count can, in fact, also be estimated directly from just one fixed point run of the BP equations, by computing the value of so-called partition function [24]. In particular, this approach computes the exact model count on tree-like formulas, and appeared to work fairly well on random formulas. However, the count estimated this way is often highly inaccurate on structured loopy formulas. `BPCount`, as we will see, makes a much more robust use of the information provided by BP.

The second method, which we call `MiniCount`, exploits the power of modern DPLL [5, 6] based SAT solvers, which are extremely good at finding single solutions to Boolean formulas through backtrack search.[2] The problem of computing upper bounds on the model count has so far eluded solution because of an asymmetry which manifests itself in at least two inter-related forms: the set of solutions of interesting N variable formulas typically forms a minuscule fraction of the full space of 2^N variable assignments, and the application of Markov's inequality as in `SampleCount` does not yield interesting upper bounds. Note that systematic model counters like `Relsat` and `Cachet` can also be easily extended to provide an upper bound when they time out (2^N minus the number of non-solutions encountered), but these bounds are uninteresting because of the above asymmetry. To address this issue, we develop a statistical framework which lets us compute upper bounds under certain statistical assumptions, which are independently validated. To the best of our knowledge, this is the first effective and scalable method for obtaining good upper bounds on the model counts of formulas that are beyond the reach of exact model counters.

More specifically, we describe how the DPLL-based solver `MiniSat` [7], with two minor modifications, can be used to estimate the total number of solutions. The number d of branching decisions (not counting unit propagations and failed branches) made by `MiniSat` before reaching a solution, is the main quantity of interest: when the choice between setting a variable to TRUE or to FALSE is randomized,[3] the number d is provably not any lower, in expectation, than $\log_2$(model count). This provides a strategy for obtaining upper bounds on the model count, only if one could efficiently estimate the expected value, $\mathbb{E}[d]$, of the number of such branching decisions. A natural way to estimate $\mathbb{E}[d]$ is to perform multiple runs of the randomized solver, and compute the average of d over these runs. However, if the formula has many "easy" solutions (found with a low value of d) and many "hard" solutions, the limited number of runs one can perform in a reasonable amount of time may be insufficient to hit many of the

[2] Gogate and Dechter [8] have recently independently proposed the use of DPLL solvers for model counting.

[3] `MiniSat` by default always sets variables to FALSE.

"hard" solutions, yielding too low of an estimate for $\mathbb{E}[d]$ and thus an incorrect upper bound on the model count.

Interestingly, we show that for many families of formulas, d has a distribution that is very close to the normal distribution. Under the assumption that d is normally distributed, when sampling various values of d through multiple runs of the solver, we need not necessarily encounter high values of d in order to correctly estimate $\mathbb{E}[d]$ for an upper bound. Instead, we can rely on statistical tests and conservative computations [20, 26] to obtain a statistical upper bound on $\mathbb{E}[d]$ within any specified confidence interval.

We evaluated our two approaches on challenging formulas from several domains. Our experiments with `BPCount` demonstrate a clear gain in efficiency, while providing much higher lower bound counts than exact counters (which often run out of time or memory) and competitive lower bound quality compared to `SampleCount`. For example, the runtime on several difficult instances from the FPGA routing family with over 10^{100} solutions is reduced from hours for both exact counters and `SampleCount` to just a few minutes with `BPCount`. Similarly, for random 3CNF instances with around 10^{20} solutions, we see a reduction in computation time from hours and minutes to seconds. With `MiniCount`, we are able to provide good upper bounds on the solution counts, often within seconds and fairly close to the true counts (if known) or lower bounds. These experimental results attest to the effectiveness of the two proposed approaches in significantly extending the reach of solution counters for hard combinatorial problems.

2 Notation

A Boolean variable x_i is one that assumes a value of either 1 or 0 (TRUE or FALSE, respectively). A truth assignment for a set of Boolean variables is a map that assigns each variable a value. A Boolean formula F over a set of n such variables is a logical expression over these variables, which represents a function $f : \{0,1\}^n \to \{0,1\}$ determined by whether or not F evaluates to TRUE under a truth assignment for the n variables. A special class of such formulas consists of those in the Conjunctive Normal Form or CNF: $F \equiv (l_{11} \vee \ldots \vee l_{1k_1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mk_m})$, where each literal l_{lk} is one of the variables x_i or its negation $\neg x_i$. Each conjunct of such a formula is called a clause. We will be working with CNF formulas throughout this paper.

The constraint graph of a CNF formula F has variables of F as vertices and an edge between two vertices if both of the corresponding variables appear together in some clause of F. When this constraint graph has no cycles (i.e., it is a collection of disjoint trees), F is called a *tree-like* or *poly-tree* formula.

The problem of finding a truth assignment for which F evaluates to TRUE is known as the *propositional satisfiability* problem, or SAT, and is the canonical NP-complete problem. Such an assignment is called a *satisfying assignment* or a *solution* for F. In this paper we are concerned with the problem of counting the number of satisfying assignments for a given formula, known as the *propositional model counting* problem. This problem is #P-complete [21].

3 Lower Bounds Using BP Marginal Estimates

In this section, we develop a method for obtaining a lower bound on the solution count of a given formula, using the framework recently used in the SAT model counter `SampleCount` [9]. The key difference between our approach and `SampleCount` is that instead of relying on solution samples, we use a variant of belief propagation to obtain estimates of the fraction of solutions in which a variable appears positively. We call this algorithm `BPCount`. After describing the basic method, we will discuss two techniques that improve the tightness of `BPCount` bounds in practice, namely, *biased variable assignments* and *safety checks*.

3.1 Counting using BP: `BPCount`

We begin by recapitulating the framework of `SampleCount` for obtaining lower bound model counts with probabilistic correctness guarantees. A variable u will be called *balanced* if it occurs equally often positively and negatively in all solutions of the given formula. In general, the *marginal probability* of u being TRUE in the set of satisfying assignments of a formula is the fraction of such assignments where $u =$ TRUE. Note that computing the marginals of each variable, and in particular identifying balanced or near-balanced variables, is quite non-trivial. The model counting approaches we describe attempt to estimate such marginals using indirect techniques such as solution sampling or iterative message passing.

Given a formula F and parameters $t, z \in \mathbb{Z}^+, \alpha > 0$, `SampleCount` performs t iterations, keeping track of the minimum count obtained over these iterations. In each iteration, it samples z solutions of (potentially simplified) F, identifies the most balanced variable u, uniformly randomly sets u to TRUE or FALSE, simplifies F by performing any possible unit propagations, and repeats the process. The repetition ends when F is reduced to a size small enough to be feasible for exact model counters like `Cachet`. At this point, let s denote the number of variables randomly set in this iteration before handing the formula to `Cachet`, and let M' be the model count of the residual formula returned by `Cachet`. The count for this iteration is computed to be $2^{s-\alpha} \times M'$ (where α is a "slack" factor pertaining to our probabilistic confidence in the bound). Here 2^s can be seen as scaling up the residual count by a factor of 2 for every uniform random decision we made when fixing variables. After the t iterations are over, the minimum of the counts over all iterations is reported as the lower bound for the model count of F, and the correctness confidence attached to this lower bound is $1 - 2^{-\alpha t}$. This means that the reported count is a correct lower bound with probability $1 - 2^{-\alpha t}$.

The performance of `SampleCount` is enhanced by also considering balanced variable pairs (v, w), where the balance is measured as the difference in the fractions of all solutions in which v and w appear with the same sign vs. with different signs. When a pair is more balanced than any single literal, the pair is used instead for simplifying the formula. In this case, we replace w with v or $\neg v$ uniformly at random. For ease of illustration, we will focus here only on identifying and randomly setting balanced or near-balanced variables.

The key observation in `SampleCount` is that when the formula is simplified by repeatedly assigning a positive or negative polarity to variables, the expected value of the count in each iteration, $2^s \times M'$ (ignoring the slack factor α), is exactly the true model count of F, from which lower bound guarantees follow. We refer the reader to Gomes et al. [9] for details. Informally, we can think of what happens when the first such balanced variable, say u, is set uniformly at random. Let $p \in [0, 1]$. Suppose F has M solutions, $F|_u$ has pM solutions, and $F|_{\neg u}$ has $(1-p)M$ solutions. Of course, when setting u uniformly at random, we don't know the actual value of p. Nonetheless, with probability a half, we will recursively count the search space with pM solutions and scale it up by a factor of 2, giving a net count of $pM.2$. Similarly, with probability a half, we will recursively get a net count of $(1-p)M.2$ solutions. On average, this gives $^1\!/_2.pM.2 + ^1\!/_2.(1-p)M.2 = M$ solutions.

Interestingly, the correctness guarantee of this process holds irrespective of how good or bad the samples are. However, when balanced variables are correctly identified, we have $p \approx ^1\!/_2$ in the informal analysis above, so that for both coin flip outcomes we recursively search a space with roughly $M/2$ solutions. This reduces the variance tremendously, which is crucial to making the process effective in practice. Note that with high variance, the minimum count over t iterations is likely to be much smaller than the true count; thus high variance leads to poor quality lower bounds.

The idea of <u>*`BPCount`*</u> *is to "plug-in" belief propagation methods in place of solution sampling in the `SampleCount` framework,* in order to estimate "p" in the intuitive analysis above and, in particular, to help identify balanced variables. As it turns out, a solution to the BP equations [15] provides exactly what we need: an estimate of the marginals of each variable. This is an alternative to using sampling for this purpose, and is often orders of magnitude faster. One bottleneck, however, is that the basic belief propagation process is iterative and does not even converge on most formulas of interest. We therefore use a "message damping" variant of standard BP, very similar to the one introduced by Pretti [16]. This variant is parameterized by $\kappa \in [0, 1]$, and has the property that as κ decreases, the dynamics of the equations go from standard BP (for $\kappa = 1$) to a damped variant with assured convergence (for $\kappa = 0$). The equations are analogous to standard BP for SAT (see e.g. [13] Figure 4 with $\rho = 0$ for a full description), differing only in the added κ exponent in the iterative update equation as shown in Figure 1. We use its output as an estimate of the marginals of the variables in `BPCount`. Note that there are several variants of BP that assure convergence, such as by Yuille [25] and Hsu and McIlraith [11]; we chose the "κ" variant because of its good scaling behavior.

Given this process of obtaining marginal estimates from BP, `BPCount` works almost exactly like `SampleCount` and *provides the same lower bound guarantees.*

Using Biased Coins. We can improve the performance of `BPCount` (and also of `SampleCount`) by using biased variable assignments. The idea here is that when fixing variables repeatedly in each iteration, the values need not be chosen uniformly. The correctness guarantees still hold even if we use a biased coin

$$\eta_{a\to i} = \prod_{j\in V(a)\setminus i}\left[\frac{\left(\prod_{b\in C_a^s(i)}(1-\eta_{b\to i})\right)^{\kappa}}{\left(\prod_{b\in C_a^s(i)}(1-\eta_{b\to i})\right)^{\kappa}+\left(\prod_{b\in C_a^u(i)}(1-\eta_{b\to i})\right)^{\kappa}}\right]$$

Notation. $V(a)$: all variables in clause a. $C_a^u(i), i \in V(a)$: clauses where i appears with the *opposite* sign than it has in a. $C_a^s(i), i \in V(a)$: clauses where i appears with the *same* sign as it has in a (except for a).

Fig. 1. $BP(\kappa)$ update equation

and set the chosen variable u to TRUE with probability q and to FALSE with probability $1 - q$, for any $q \in (0, 1)$. Using earlier notation, this leads us to a solution space of size pM with probability q and to a solution space of size $(1 - p)M$ with probability $1 - q$. Now, instead of scaling up with a factor of 2 in both cases, we scale up based on the bias of the coin used. Specifically, with probability q, we go to one part of the solution space and scale it up by $1/q$, and similarly for $1 - q$. The net result is that in expectation, we still get $q.pM/q + (1 - q).(1 - p)M/(1 - q) = M$ solutions. Further, the variance is minimized when q is set to equal p; in `BPCount`, q is set to equal the estimate of p obtained using the BP equations. To see that the resulting variance is minimized this way, note that with probability q, we get a net count of pM/q, and with probability $(1 - q)$, we get a net count of $(1 - p)M/(1 - q)$; these balance out to exactly M in either case when $q = p$. Hence, when we have confidence in the correctness of the estimates of variable marginals (i.e., p here), it provably reduces variance to use a biased coin that matches the marginal estimates of the variable to be fixed.

Safety Checks. One issue that arises when using BP techniques to estimate marginals is that the estimates, in some case, may be far off from the true marginals. In the worst case, a variable u identified by BP as the most balanced may in fact be a backbone variable for F, i.e., may only occur, say, positively in all solutions to F. Setting u to FALSE based on the outcome of the corresponding coin flip thus leads one to a part of the search space with no solutions at all, so that the count for this iteration is zero, making the minimum over t iterations zero as well. To remedy this situation, we use safety checks using an off-the-shelf SAT solver (`Minisat` or `Walksat` in our implementation) before fixing the value of any variable. The idea is to simply check that u can be set both ways *before* flipping the random coin and fixing u to TRUE or FALSE. If `Minisat` finds, e.g., that forcing u to be TRUE makes the formula unsatisfiable, we can immediately deduce $u =$ FALSE, simplify the formula, and look for a different balanced variable. This safety check prevents `BPCount` from reaching the undesirable state where there are no remaining solutions at all.

In fact, with the addition of safety checks, we found that the lower bounds on model counts obtained for some formulas were surprisingly good even when the marginal estimates were generated purely at random, i.e., without actually

running BP. This can perhaps be explained by the errors introduced at each step somehow canceling out when several variables are fixed. With the use of BP, the quality of the lower bounds was significantly improved, showing that BP does provide useful information about marginals even for loopy formulas. Lastly, we note that with `SampleCount`, the external safety check can be conservatively replaced by simply avoiding those variables that appear to be backbone variables from the obtained samples.

4 Upper Bound Estimation

We now describe an approach for estimating an upper bound on the solution count. We use the reasoning discussed for `BPCount`, and apply it to a DPLL style search procedure. There is an important distinction between the nature of the bound guarantees presented here and earlier: here we will derive *statistical* (as opposed to probabilistic) guarantees, and their quality may depend on the particular family of formulas in question. The applicability of the method will also be determined by a statistical test, which succeeded in most of our experiments.

4.1 Counting using Backtrack Search: `MiniCount`

For `BPCount`, we used a backtrack-less branching search process with a random outcome that, in expectation, gives the exact number of solutions. The ability to randomly assign values to selected variables was crucial in this process. Here we extend the same line of reasoning to a search process *with* backtracking, and argue that the expected value of the outcome is an upper bound on the true count. We extend the `MiniSat` SAT solver [7] to compute the information needed for upper bound estimation. `MiniSat` is a very efficient SAT solver employing conflict clause learning and other state-of-the-art techniques, and has one important feature helpful for our purposes: whenever it chooses a variable to branch on, it is left unspecified which value should the variable assume first. One possibility is to assign values TRUE or FALSE randomly with equal probability. Since `MiniSat` does not use any information about the variable to determine the most promising polarity, this random assignment in principle does not lower `MiniSat`'s power.

Algorithm `MiniCount`: Given a formula F, run `MiniSat` with *no restarts*, choosing a value for a variable uniformly at random at each choice point (option `-polarity-mode=rnd`). When a solution is found, output 2^d where d is the number of choice points on the path to the solution (the final decision level), not counting those choice points where the other branch failed to find a solution.

The restriction that `MiniCount` cannot use restarts is the only change to the solver. This limits somewhat the range of problems `MiniCount` can be applied to compared to the original `MiniSat`, but is a crucial restriction for the guarantee of an upper bound (as explained below). We found that `MiniCount` is still efficient on a wide range of formulas. Since `MiniCount` is a probabilistic algorithm, its

output, 2^d, on a given formula F is a random variable. We denote this random variable by $\#F_{\text{MiniCount}}$, and use $\#F$ to denote the true number of solutions of F. The following proposition forms the basis of our upper bound estimation.

Proposition 1. $\mathbb{E}\left[\#F_{\text{MiniCount}}\right] \geq \#F$.

Proof. The proof follows a similar line of reasoning as for `BPCount`, and we give a sketch of it. Note that if no backtracking is allowed (i.e., the solver reports 0 solutions if it finds a contradiction), the result follows, with strict equality, from the proof that `BPCount` (or `SampleCount`) provides accurate counts in expectation. We will show that the addition of backtracking can only increase the value of $\mathbb{E}\left[\#F_{\text{MiniCount}}\right]$, by looking at its effect on any choice point. Let u be any choice point variable with at least one satisfiable branch in its subtree, and let M be the number of solutions in the subtree, with pM in the left branch (when $u =$FALSE) and $(1-p)M$ in the right branch (when $u =$TRUE). If both branches under u are satisfiable, then the expected number of solutions computed at u is $1/2.pM.2 + 1/2.(1-p)M.2 = M$, which is the correct value. However, if either branch is unsatisfiable, then two things might happen: with probability half the search process will discover this fact by exploring the contradictory branch first and u will not be counted as a choice point in the final solution (i.e., its multiplier will be 1), and with probability half this fact will go unnoticed and u will retain its multiplier of 2. Thus the expected number of reported solutions at u is $1/2.M.2 + 1/2.M = \frac{3}{2}M$, which is no smaller than M. This finishes the proof.

The reason restarts are not allowed in `MiniCount` is exactly Proposition 1. With restarts, only solutions reachable within the current setting of the restart threshold can be found. This biases the search towards "easier" solutions, since they are given more opportunities to be found. For formulas where easier solutions lie on paths with fewer choice points, `MiniCount` with restarts could undercount and thus not provide an upper bound in expectation.

With enough random sample outputs, $\#F_{\text{MiniCount}}$, obtained from `MiniCount`, their average value will eventually converge to $\mathbb{E}\left[\#F_{\text{MiniCount}}\right]$ by the Law of Large Numbers, thereby providing an upper bound on $\#F$ because of Proposition 1. Unfortunately, providing a useful correctness guarantee on such an upper bound in a manner similar to the lower bounds seen earlier turns out to be impractical, because the resulting guarantees, obtained using a reverse variant of the standard Markov's inequality, are too weak. Further, relying on the simple average of the obtained output samples might also be misleading, since the distribution of $\#F_{\text{MiniCount}}$ is often heavy tailed, and it might take very many samples for the sample mean to become as large as the true solution count.

4.2 Estimating the Upper Bound

In this section, we develop an approach based on statistical analysis of the sample outputs that allows one to estimate the expected value of $\#F_{\text{MiniCount}}$, and thus an upper bound with statistical guarantees, using relatively few samples.

Assuming the distribution of $\#F_{\text{MiniCount}}$ is known, the samples can be used to provide an unbiased estimate of the mean, along with confidence intervals on this estimate. This distribution is of course not known and will vary from formula to formula, but it can again be inferred from the samples. We observed that for many formulas, the distribution of $\#F_{\text{MiniCount}}$ is well approximated by a log-normal distribution. Thus we develop the method under the assumption of log-normality, and include techniques to independently test this assumption. The method has three steps:

1. Generate n independent samples from $\#F_{\text{MiniCount}}$ by running `MiniCount` n times on the same formula.
2. Test whether the samples come from a log-normal distribution (or a distribution sufficiently similar).
3. Estimate the true expected value of $\#F_{\text{MiniCount}}$ from the samples, and calculate the $(1-\alpha)\%$ confidence interval for it, using the assumption that the underlying distribution is log-normal. We set the confidence level α to 0.01, and denote the upper bound of the resulting confidence interval by c_{max}.

This process, some of whose details will be discussed shortly, yields an upper bound c_{max} along with a statistical guarantee that $c_{\text{max}} \geq \mathbb{E}[\#F_{\text{MiniCount}}]$ and thus $c_{\text{max}} \geq \#F$:

$$\Pr[c_{\text{max}} \geq \#F] \geq 1 - \alpha$$

The caveat in this statement (and, in fact, the main difference from the similar statement for the lower bounds for `BPCount` given earlier) is that it is true only if our assumption of log-normality holds.

Testing for Log-Normality. By definition, a random variable X has a log-normal distribution if the random variable $Y = \log X$ has a normal distribution. Thus a test whether Y is normally distributed can be used, and we use the Shapiro-Wilk test [cf. 20] for this purpose. In our case, $Y = \log(\#F_{\text{MiniCount}})$ and if the computed p-value of the test is below the confidence level $\alpha = 0.05$, we conclude that our samples do *not* come from a log-normal distribution; otherwise we assume that they do. If the test fails, then there is sufficient evidence that the underlying distribution is not log-normal, and the confidence interval analysis to be described shortly will not provide any statistical guarantees. Note that non-failure of the test does not mean that the samples *are* actually log-normally distributed, but inspecting the Quantile-Quantile plots (QQ-plots) often supports the hypothesis that they are. QQ-plots compare sampled quantiles with theoretical quantiles of the desired distribution: the more the sample points align on a line, the more likely it is that the data comes from the distribution.

We found that a surprising number of formulas had $\log_2(\#F_{\text{MiniCount}})$ very close to being normally distributed. Figure 2 shows normalized QQ-plots for $d_{\text{MiniCount}} = \log_2(\#F_{\text{MiniCount}})$ obtained from 100 to 1000 runs of `MiniCount` on various families of formulas (discussed in the experimental section). The top-left QQ-plot shows the best fit of normalized $d_{\text{MiniCount}}$ (obtained by subtracting the average and dividing by the standard deviation) to the normal distribution:

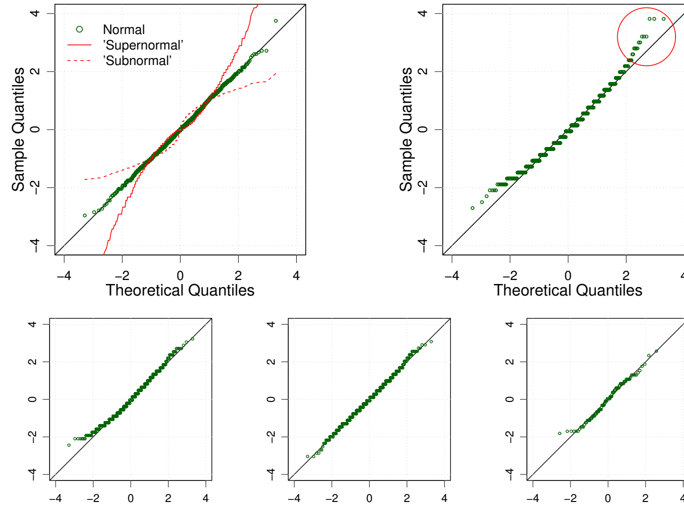

Fig. 2. Sampled and theoretical quantiles for formulas described in the experimental section (top: `alu2_gr_rcs_w8`, `lang19`; bottom: `2bitmax_6`, `wff-3-150-525`, `ls11-norm`).

(normalized $d_{\text{MiniCount}} = d) \sim \frac{1}{\sqrt{2\pi}} e^{-d^2/2}$. The 'supernormal' and 'subnormal' lines show that the fit is much worse when the exponent of d is, for example, 1.5 or 2.5. The top-right plot shows that the corresponding domain (Langford problems) is somewhat on the border of being log-normally distributed, which is reflected in our experimental results to be described later.

Note that the nature of statistical tests is such that if the distribution of $\mathbb{E}[\#F_{\text{MiniCount}}]$ is not *exactly* log-normal, obtaining more and more samples will eventually lead to rejecting the log-normality hypothesis. For most practical purposes, being "close" to log-normally distributed suffices.

Confidence Interval Bound. Assuming the output samples from `MiniCount` $\{o_1, \ldots, o_n\}$ come from a log-normal distribution, we use them to compute the upper bound $c_{\max}$ of the confidence interval for the mean of $\#F_{\text{MiniCount}}$. The exact method for computing $c_{\max}$ for a log-normal distribution is complicated, and seldom used in practice. We use a conservative bound computation [26]: let $y_i = \log(o_i)$, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ denote the sample mean, and $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$ the sample variance. Then the conservative upper bound is constructed as

$$\tilde{c}_{\max} = \bar{y} + \frac{s^2}{2} + \left(\frac{n-1}{\chi^2_\alpha(n-1)} - 1\right)\sqrt{\frac{s^2}{2}\left(1 + \frac{s^2}{2}\right)}$$

where $\chi^2_\alpha(n-1)$ is the α-percentile of the chi-square distribution with $n-1$ degrees of freedom. Since $\tilde{c}_{\max} \geq c_{\max}$ we still have $\Pr[\tilde{c}_{\max} \geq \mathbb{E}[\#F_{\text{MiniCount}}]] \geq 1 - \alpha$.

The main assumption of the method described in this section is that the distribution of $\#F_{\mathrm{MiniCount}}$ can be well approximated by a log-normal. This, of course, depends on the nature of the search space of `MiniCount` on a particular formula. As noted before, the assumption may sometimes be incorrect. In particular, one can construct a pathological search space where the reported upper bound will be lower than the actual number of solutions. Consider a problem P that consists of two non-interacting subproblems P_1 and P_2, where it is sufficient to solve either one of them to solve P. Suppose P_1 is very easy to solve (e.g., requires few choice points that are easy to find) compared to P_2, and P_1 has very few solutions compared to P_2. In such a case, `MiniCount` will almost always solve P_1 (and thus estimate the number of solutions of P_1), which would leave an arbitrarily large number of solutions of P_2 unaccounted for. This situation violates the assumption that $\#F_{\mathrm{MiniCount}}$ is log-normally distributed, but it may be left unnoticed. This possibility of a false upper bound is a consequence of the inability to prove from samples that a random variable is log-normally distributed (one may only disprove this assertion). Fortunately, as our experiments suggest, this situation is rare and does not arise in many real-world problems.

5 Experimental Results

We conducted experiments with `BPCount` as well as `MiniCount`, with the primary focus on comparing the results to exact counters and the recent `SampleCount` algorithm providing probabilistically guaranteed lower bounds. We used a cluster of 3.8 GHz Intel Xeon computers running Linux 2.6.9-22.ELsmp. The time limit was set to 12 hours and the memory limit to 2 GB.

We consider problems from five different domains, many of which have previously been used as benchmarks for evaluating model counting techniques: circuit synthesis, random k-CNF, Latin square construction, Langford problems, and FPGA routing instances from the SAT 2002 competition. The results are summarized in Table 1. The columns show the performance of `BPCount` and `MiniCount`, compared against the exact solution counters `Relsat`, `Cachet`, and `c2d` (we report the best of the three for each instance; for all but the first instance, `c2d` exceeded the memory limit) and `SampleCount`. The table shows the reported bounds on the model counts and the corresponding runtime in seconds.

For `BPCount`, the damping parameter setting (i.e., the κ value) we use for the damped BP marginal estimator is 0.8, 0.9, 0.9, 0.5, and either 0.1 or 0.2 for the five domains, respectively. This parameter is chosen (with a quick manual search) as high as possible so that BP converges in a few seconds or less. The exact counter `Cachet` is called when the formula is sufficiently simplified, which is when 50 to 500 variables remain, depending on the domain. The lower bounds on the model count are reported with 99% confidence. We see that a significant improvement in efficiency is achieved when the BP marginal estimation is used through `BPCount`, compared to solution sampling as in `SampleCount` (also run with 99% correctness confidence). For the smaller formulas considered, the lower bounds reported by `BPCount` border the true model counts. For the

Table 1. Performance of `BPCount` and `MiniCount`. [R] and [C] indicate partial counts obtained from `Cachet` and `Relsat`, respectively. `c2d` was slower for the first instance and exceeded the memory limit of 2 GB for the rest. Runtime is in seconds.

Instance	# of vars	True Count (if known)	`Cachet / Relsat / c2d` (exact counters) Models	Time	`SampleCount` (99% confidence) LWR-bound	Time	`BPCount` (99% confidence) LWR-bound	Time	`MiniCount` S-W Test	Average	(99% confidence) UPR-bound	Time
CIRCUIT SYNTH.												
2bitmax_6	252	2.1×10^{29}	2.1×10^{29}	2 sec[C]	$\geq 2.4 \times 10^{28}$	29 sec	$\geq \mathbf{2.8 \times 10^{28}}$	**5 sec**	✓	3.5×10^{30}	$\leq \mathbf{4.3 \times 10^{32}}$	**2 sec**
RANDOM k-CNF												
wff-3-3.5	150	1.4×10^{14}	1.4×10^{14}	7 min[C]	$\geq 1.6 \times 10^{13}$	4 min	$\geq \mathbf{1.6 \times 10^{11}}$	**3 sec**	✓	4.3×10^{14}	$\leq \mathbf{6.7 \times 10^{15}}$	**2 sec**
wff-3-1.5	100	1.8×10^{21}	1.8×10^{21}	3 hrs[C]	$\geq 1.6 \times 10^{20}$	4 min	$\geq \mathbf{1.0 \times 10^{20}}$	**1 sec**	✓	1.2×10^{21}	$\leq \mathbf{4.8 \times 10^{22}}$	**2 sec**
wff-4-5.0	100	—	$\geq 1.0 \times 10^{14}$	12 hrs[C]	$\geq 8.0 \times 10^{15}$	2 min	$\geq \mathbf{2.0 \times 10^{15}}$	**2 sec**	✓	2.8×10^{16}	$\leq \mathbf{5.7 \times 10^{28}}$	**2 sec**
LATIN SQUARE												
ls8-norm	301	5.4×10^{11}	$\geq 1.7 \times 10^{8}$	12 hrs[R]	$\geq 3.1 \times 10^{10}$	19 min	$\geq \mathbf{1.9 \times 10^{10}}$	**12 sec**	✓	6.4×10^{12}	$\leq \mathbf{1.8 \times 10^{14}}$	**2 sec**
ls9-norm	456	3.8×10^{17}	$\geq 7.0 \times 10^{7}$	12 hrs[R]	$\geq 1.4 \times 10^{15}$	32 min	$\geq \mathbf{1.0 \times 10^{16}}$	**11 sec**	✓	6.9×10^{18}	$\leq \mathbf{2.1 \times 10^{21}}$	**3 sec**
ls10-norm	657	7.6×10^{24}	$\geq 6.1 \times 10^{7}$	12 hrs[R]	$\geq 2.7 \times 10^{21}$	49 min	$\geq \mathbf{1.0 \times 10^{23}}$	**22 sec**	✓	4.3×10^{26}	$\leq \mathbf{7.0 \times 10^{30}}$	**7 sec**
ls11-norm	910	5.4×10^{33}	$\geq 4.7 \times 10^{7}$	12 hrs[R]	$\geq 1.2 \times 10^{30}$	69 min	$\geq \mathbf{6.4 \times 10^{30}}$	**1 min**	✓	1.7×10^{34}	$\leq \mathbf{5.6 \times 10^{40}}$	**35 sec**
ls12-norm	1221	—	$\geq 4.6 \times 10^{7}$	12 hrs[R]	$\geq 6.9 \times 10^{37}$	50 min	$\geq \mathbf{2.0 \times 10^{41}}$	**70 sec**	✓	9.1×10^{44}	$\leq \mathbf{3.6 \times 10^{52}}$	**4 min**
ls13-norm	1596	—	$\geq 2.1 \times 10^{7}$	12 hrs[R]	$\geq 3.0 \times 10^{49}$	67 min	$\geq \mathbf{4.0 \times 10^{54}}$	**6 min**	✓	1.0×10^{54}	$\leq \mathbf{8.6 \times 10^{69}}$	**42 min**
ls14-norm	2041	—	$\geq 2.6 \times 10^{7}$	12 hrs[R]	$\geq 9.0 \times 10^{60}$	44 min	$\geq \mathbf{1.0 \times 10^{67}}$	**4 min**	✓	3.2×10^{63}	$\leq \mathbf{1.3 \times 10^{86}}$	**7.5 hrs**
LANGFORD PROBS.												
lang-2-12	576	1.0×10^{5}	1.0×10^{5}	15 min[R]	$\geq 4.3 \times 10^{3}$	32 min	$\geq \mathbf{2.3 \times 10^{3}}$	**50 sec**	×	5.2×10^{6}	$\leq 1.0 \times 10^{7}$	2.5 sec
lang-2-15	1024	3.0×10^{7}	$\geq 1.8 \times 10^{5}$	12 hrs[R]	$\geq 1.0 \times 10^{6}$	60 min	$\geq \mathbf{5.5 \times 10^{5}}$	**1 min**	✓	1.0×10^{8}	$\leq \mathbf{9.0 \times 10^{8}}$	**8 sec**
lang-2-16	1024	3.2×10^{8}	$\geq 1.8 \times 10^{5}$	12 hrs[R]	$\geq 1.0 \times 10^{6}$	65 min	$\geq \mathbf{3.2 \times 10^{5}}$	**1 min**	×	1.1×10^{10}	$\leq 1.1 \times 10^{10}$	7.3 sec
lang-2-19	1444	2.1×10^{11}	$\geq 2.4 \times 10^{5}$	12 hrs[R]	$\geq 3.3 \times 10^{9}$	62 min	$\geq \mathbf{4.7 \times 10^{7}}$	**26 min**	×	1.4×10^{10}	$\leq 6.7 \times 10^{12}$	37 sec
lang-2-20	1600	2.6×10^{12}	$\geq 1.5 \times 10^{5}$	12 hrs[R]	$\geq 5.8 \times 10^{9}$	54 min	$\geq \mathbf{7.1 \times 10^{4}}$	**22 min**	✓	1.4×10^{12}	$\leq \mathbf{9.4 \times 10^{12}}$	**3 min**
lang-2-23	2116	3.7×10^{15}	$\geq 1.2 \times 10^{5}$	12 hrs[R]	$\geq 1.6 \times 10^{11}$	85 min	$\geq \mathbf{1.5 \times 10^{5}}$	**15 min**	×	3.5×10^{12}	$\leq 1.4 \times 10^{13}$	23 min
lang-2-24	2304	—	$\geq 4.1 \times 10^{5}$	12 hrs[R]	$\geq 4.1 \times 10^{13}$	80 min	$\geq \mathbf{8.9 \times 10^{7}}$	**18 min**	×	2.7×10^{13}	$\leq 1.9 \times 10^{16}$	25 min
FPGA routing (SAT2002)												
apex7_*_w5	1983	—	$\geq 4.5 \times 10^{47}$	12 hrs[R]	$\geq 8.8 \times 10^{85}$	20 min	$\geq \mathbf{3.0 \times 10^{82}}$	**3 min**	✓	7.3×10^{95}	$\leq \mathbf{5.9 \times 10^{105}}$	**2 min**
9symml_*_w6	2604	—	$\geq 5.0 \times 10^{30}$	12 hrs[R]	$\geq 2.6 \times 10^{47}$	6 hrs	$\geq \mathbf{1.8 \times 10^{46}}$	**6 min**	✓	3.3×10^{58}	$\leq \mathbf{5.8 \times 10^{64}}$	**24 sec**
c880_*_w7	4592	—	$\geq 1.4 \times 10^{43}$	12 hrs[R]	$\geq 2.3 \times 10^{273}$	5 hrs	$\geq \mathbf{7.9 \times 10^{253}}$	**18 min**	✓	1.0×10^{264}	$\leq \mathbf{6.3 \times 10^{326}}$	**26 sec**
alu2_*_w8	4080	—	$\geq 1.8 \times 10^{56}$	12 hrs[R]	$\geq 2.4 \times 10^{220}$	143 min	$\geq \mathbf{2.0 \times 10^{205}}$	**16 min**	✓	1.4×10^{220}	$\leq \mathbf{7.2 \times 10^{258}}$	**16 sec**
vda_*_w9	6498	—	$\geq 1.4 \times 10^{88}$	12 hrs[R]	$\geq 1.4 \times 10^{326}$	11 hrs	$\geq \mathbf{3.8 \times 10^{289}}$	**56 min**	✓	1.6×10^{305}	$\leq \mathbf{2.5 \times 10^{399}}$	**42 sec**

larger ones that could only be counted partially by exact counters in 12 hours, BPCount gave lower bound counts that are very competitive with those reported by SampleCount, while the running time of BPCount is, in general, an order of magnitude lower than that of SampleCount, often just a few seconds.

For MiniCount, we obtain $n = 100$ samples of the estimated count for each formula, and use these to estimate the upper bound statistically using the steps described earlier. The test for log-normality of the sample counts is done with a rejection level 0.05, that is, if the Shapiro-Wilk test reports p-value below 0.05, we conclude the samples do *not* come from a log-normal distribution, in which case no upper bound guarantees are provided (MiniCount is "unsuccessful"). When the test passed, the upper bound itself was computed with a confidence level of 99% using the computation of Zhou and Sujuan [26]. The results are summarized in the last set of columns in Table 1. We report whether the log-normality test passed, the average of the counts obtained over the 100 runs, the value of the statistical upper bound $c_{\max}$, and the total time for the 100 runs. We see that the upper bounds are often obtained within seconds or minutes, and are correct for all instances where the estimation method was successful (i.e., the log-normality test passed) and true counts or lower bounds are known. In fact, the upper bounds for these formulas (except lang-2-23) are correct w.r.t. the best known lower bounds and true counts even for those instances where the log-normality test failed and a statistical guarantee cannot be provided. The Langford problem family seems to be at the boundary of applicability of the MiniCount approach, as indicated by the alternating successes and failures of the test in this case. The approach is particularly successful on industrial problems (circuit synthesis, FPGA routing), where upper bounds are computed within seconds. Our results also demonstrate that a simple average of the 100 runs provides a very good approximation to the number of solutions. However, simple averaging can sometimes lead to an incorrect upper bound, as seen in wff-3-1.5, ls13-norm, alu2_gr_rcs_w8, and vda_gr_rcs_w9, where the simple average is below the true count or a lower bound obtained independently. This justifies our statistical framework, which as we see provides more robust upper bounds.

6 Conclusion

This work brings together techniques from message passing, DPLL-based SAT solvers, and statistical estimation in an attempt to solve the challenging model counting problem. We show how (a damped form of) BP can help significantly boost solution counters that produce lower bounds with probabilistic correctness guarantees. BPCount is able to provide good quality bounds in a fraction of the time compared to previous, sample-based methods. We also describe the first effective approach for obtaining good upper bounds on the solution count. Our framework is general and enables one to turn any state-of-the-art complete SAT/CSP solver into an upper bound counter, with very minimal modifications to the code. Our MiniCount algorithm provably converges to an upper bound, and is remarkably fast at providing good results in practice.

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44nd FOCS*, pp. 340–351, Oct. 2003.
[2] R. J. Bayardo Jr. and J. D. Pehoushek. Counting models using connected components. In *17th AAAI*, pp. 157–162, Austin, TX, July 2000.
[3] A. Darwiche. New advances in compiling CNF into decomposable negation normal form. In *16th ECAI*, pp. 328–332, Valencia, Spain, Aug. 2004.
[4] A. Darwiche. The quest for efficient probabilistic inference, July 2005. Invited Talk, IJCAI-05.
[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *CACM*, 5:394–397, 1962.
[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *CACM*, 7:201–215, 1960.
[7] N. Eén and N. Sörensson. MiniSat: A SAT solver with conflict-clause minimization. In *8th SAT*, St. Andrews, U.K., June 2005.
[8] V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *22th AAAI*, pp. 198–203, Vancouver, BC, July 2007.
[9] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *20th IJCAI*, pp. 2293–2299, Hyderabad, India, Jan. 2007.
[10] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *21th AAAI*, pp. 54–61, Boston, MA, July 2006.
[11] E. I. Hsu and S. A. McIlraith. Characterizing propagation methods for boolean satisfiabilty. In *9th SAT*, vol. 4121 of *LNCS*, pp. 325–338, Seattle, WA, Aug. 2006.
[12] M. L. Littman, S. M. Majercik, and T. Pitassi. Stochastic Boolean satisfiability. *J. Auto. Reas.*, 27(3):251–296, 2001.
[13] E. Maneva, E. Mossel, and M. J. Wainwright. A new look at survey propagation and its generalizations. *J. Assoc. Comput. Mach.*, 54(4):17, July 2007.
[14] J. D. Park. MAP complexity results and approximation methods. In *18th UAI*, pp. 388–396, Edmonton, Canada, Aug. 2002.
[15] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
[16] M. Pretti. A message-passing algorithm with damping. *J.Stat.Mech.*,P11008,2005.
[17] D. Roth. On the hardness of approximate reasoning. *AI J.*, 82(1-2):273–302, 1996.
[18] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. *7th SAT*, 2004.
[19] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *20th AAAI*, pp. 475–482, Pittsburgh, PA, July 2005.
[20] H. C. Thode. *Testing for Normality*. CRC, 2002.
[21] L. G. Valiant. The complexity of computing the permanent. *Theoretical Comput. Sci.*, 8:189–201, 1979.
[22] W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In *19th AAAI*, pp. 670–676, San Jose, CA, July 2004.
[23] W. Wei and B. Selman. A new approach to model counting. In *8th SAT*, vol. 3569 of *LNCS*, pp. 324–339, St. Andrews, U.K., June 2005.
[24] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Trans. Inf. Theory*, 51(7):2282–2312, 2005.
[25] A. L. Yuille. CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief prop. *Neural Comput.*, 14(7):1691–1722, 2002.
[26] X.-H. Zhou and G. Sujuan. Confidence intervals for the log-normal mean. *Statistics In Medicine*, 16:783–790, 1997.