Annals of Operations Research manuscript No.
(will be inserted by the editor)

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

Lukas Kroc · **Ashish Sabharwal** · **Bart Selman**

Received: date / Accepted: date

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 the Belief Propagation (BP) algorithm. 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.

Keywords Boolean satisfiability · SAT · number of solutions · model counting · BPCount · MiniCount · lower bounds · upper bounds

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 [28] and is widely believed to be significantly harder than the NP-complete SAT problem, which seeks an answer to whether or not the formula is satisfiable. With the amazing advances in the effectiveness of SAT solvers since the early 1990's, these solvers have come to be commonly used in combinatorial application areas such as hardware and software verification, planning, and design automation. Efficient

A preliminary version of this article appeared at the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR), Paris, France, 2008 [14].

L. Kroc · A. Sabharwal · B. Selman
Department of Computer Science, Cornell University, Ithaca NY 14853-7501, U.S.A.
E-mail: {kroc,sabhar,selman}@cs.cornell.edu

algorithms for #SAT will further open the doors to a whole new range of applications, most notably those involving probabilistic inference [1, 4, 15, 18, 22, 25].

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` [23, 24] adds caching schemes to this approach, `c2d` [3] converts formulas to the d-DNNF form which yields the model count as a by-product, `ApproxCount` [30] and `SampleCount` [10] exploit sampling techniques for estimating the count, `MBound` [11, 12] relies on the properties of random parity or XOR constraints to produce estimates with correctness guarantees, and the recently introduced `SampleMinisat` [9] 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 with confidence interval style correctness guarantees.

The first method, which we call `BPCount`, builds upon a successful approach for model counting using local search, called `ApproxCount` [30]. 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` [10], 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` [29] and by 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 [cf. 19], 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 a fixed point of these equations, 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 [21]). For the second issue, we add "safety checks" to prevent the algorithm from running into a contradiction by accidentally eliminating all assignments.[1] Somewhat surprisingly, once these rare but fatal mistakes are avoided, it turns out that we can obtain very close estimates and lower bounds for solution counts, suggesting that BP does provide useful information even

[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.

on highly structured and loopy formulas. To exploit this information even further, we extend the framework borrowed from `SampleCount` with the use of biased random coins during randomized value selection for variables.

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 [32]. 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 Davis-Putnam-Logemann-Loveland or DPLL [5, 6] based SAT solvers, which are extremely good at finding single solutions to Boolean formulas through backtrack search. (Gogate and Dechter [9] have independently proposed the use of DPLL solvers for model counting.) The problem of computing upper bounds on the model count has so far eluded an effective solution strategy in part because of an asymmetry that 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`'s correctness analysis 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 during the run), but these bounds are uninteresting because of the above asymmetry. For instance, if a search space of size $2^{1,000}$ has been explored for a $10,000$ variable formula with as many as $2^{5,000}$ solutions, the best possible upper bound one could hope to derive with this reasoning is $2^{10,000} - 2^{1,000}$, which is nearly as far away from the true count of $2^{5,000}$ as the trivial upper bound of $2^{10,000}$; the situation only gets worse when the formula has fewer solutions. 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 SAT 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,[2] the number d is provably not any lower, in expectation, than $\log_2(\text{model count})$. 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 "hard" solutions, yielding too low of an estimate for $\mathbb{E}[d]$ and thus an incorrect upper bound on the model count.

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, one need not necessarily encounter high values of d in order to correctly estimate $\mathbb{E}[d]$ for an upper bound. Instead, one can rely

[2] `MiniSat` by default always branches by setting variables first to FALSE.

on statistical tests and conservative computations [e.g. 27, 34] to obtain a statistical upper bound on $\mathbb{E}[d]$ within any specified confidence interval. This is the approach we take in this work for our upper bounds.

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 or both) and a 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 or more 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. In some cases, the lower bound provided by `MiniCount` is somewhat worse than that provided by `SampleCount`, but still quite competitive. With `MiniCount`, we are able to provide good upper bounds on the solution counts, often within seconds and within a reasonable distance from the true counts (if known) or lower bounds computed independently. These experimental results attest to the effectiveness of the two proposed approaches in significantly extending the reach of solution counters for hard combinatorial problems.

The article is organized as follows. We start in Section 2 with preliminaries and notation. Section 3 then describes our probabilistic lower bounding approach based on the proposed convergent form of belief propagation. It first discusses how marginal estimates produced by BP can be used to obtain lower bounds on the model count of a formula by modifying a previous sampling-based framework, and then suggests two new features to be added to the framework for robustness. Section 4 discusses how a backtrack search solver, with appropriate randomization and a careful restriction on restarts, can be used to obtain a process that provides an upper bound in expectation. It then proposes a statistical technique to estimate this expected value in a robust manner with statistical confidence guarantees. We present experimental results for both of these techniques in Section 5 and conclude in Section 6. The appendix gives technical details of the convergent form of BP that we propose, as well as experimental results on the performance of our upper bounding technique when “restarts” are disabled in the underlying backtrack search solver.

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 various truth assignments to the n variables. A special class of such formulas consists of those in the Conjunctive Normal Form or CNF: $F \equiv (l_{1,1} \vee \ldots \vee l_{1,k_1}) \wedge \ldots \wedge (l_{m,1} \vee \ldots \vee l_{m,k_m})$, where each literal $l_{l,k}$ 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 article.

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. Otherwise, F is said to have a *loopy* structure.

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. A *SAT solver* refers to an algorithm, and often an accompanying implementation, for the satisfiability problem. In this work we are concerned with the problem of counting the number of satisfying assignments for a given formula, known as the *propositional model counting* problem. We will also refer to it as the *solution counting* problem. In terms of worst case complexity, this problem is #P-complete [28] and is widely believed to be much harder than SAT itself. A *model counter* refers to an algorithm, and often an accompanying implementation, for the model counting problem. The model counter is said to be *exact* if it is guaranteed to output precisely the true model count of the input formula when it terminates. The model counters we propose in this work are randomized and provide either a lower bound or an upper bound on the true model count, with certain correctness guarantees.

3 Lower Bounds Using BP Marginal Estimates: BPCount

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` [10]. 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*. Finally, we will describe our variation of the belief propagation algorithm which is key to the performance of `BPCount`: a set of *parameterized* belief update equations which are guaranteed to converge for a small enough value of the parameter. Since the precise details of these parameterized iterative equations are somewhat tangential to the main focus of this work (namely, model counting techniques), we will defer many of the BP parameterization details to Appendix A.

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 $u = \text{TRUE}$. 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 $t, z \in \mathbb{Z}^+$ and $\alpha > 0$, `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 such as `Relsat` [2], `Cachet` [23], or `c2d` [3]; we will use `Cachet` in the rest of the discussion, as it is the exact model counter we used in our experiments. 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 correctness of 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 on the model count of F with probability at least $1-2^{-\alpha t}$.

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 value vs. with different values. 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, and not variable pairs. We note that our implementation of BPCount does support variable pairs.

The key observation in `SampleCount` is that when the formula is simplified by repeatedly assigning a positive or negative polarity (i.e., TRUE or FALSE values, respectively) 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. [10] 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 $pM \times 2$. Similarly, with probability a half, we will recursively get a net count of $(1-p)M \times 2$ solutions. On average, this gives $(1/2 \times pM \times 2) + (1/2 \times (1-p)M \times 2) = M$ solutions.

Observe that the correctness guarantee of this process holds irrespective of how good or bad the samples are, which determines how successful we are in identifying a balanced variable, i.e., how close is p to $1/2$. That said, if balanced variables *are* correctly identified, we have $p \approx 1/2$ in the informal analysis above, which means that for both coin flip outcomes we recursively search a space containing roughly $M/2$ solutions. This reduces the *variance* of this randomized procedure tremendously and 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 lower bounds of poor quality (although still with the same correctness guarantee).

Algorithm BPCount: *The idea behind `BPCount` is to "plug-in" belief propagation methods in place of solution sampling in the `SampleCount` framework discussed above,* 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 [19] 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.

The heart of the BP algorithm involves solving a set of iterative equations derived specifically for a given problem instance (the variables in the system are called "messages"). These equations are designed to provide accurate answers if applied to problems with no circular dependencies, such as constraint satisfaction problems with no loops in the corresponding constraint graph.

One bottleneck, however, is that the basic belief propagation process is iterative and does not even converge on most SAT instances of interest. In order to use BP for estimating marginal probabilities and identifying balanced variables, one must either cut off the iterative computation or use a modification that does converge. Unfortunately, some of the known improvements of the belief propagation technique that allow it to converge more often or be used on a wider set of problems, such as Generalized Belief Propagation [31], Loop Corrected Belief Propagation [17], or Expectation Maximization Belief Propagation [13], are not scalable enough for our purposes. The problem of very slow convergence on hard instances seems to plague also approaches based on other methods for solving BP equations than the simple iteration scheme, such as the convex-concave procedure introduced by Yuille [33]. Finally, in our context, the speed requirement is accentuated by the need to use marginal estimation repeatedly essentially every time a variable is chosen and assigned a value.

We consider a parameterized variant of BP that is guaranteed to converge when this parameter is small enough, and which imposes no additional computational cost per iteration over standard BP. (A similar but distinct parameterization was proposed by Pretti [21].) We found that this "damped" variant of BP provides much more useful information than BP iterations terminated without convergence. We refer to this particular way of damping the BP equations as $\mathbf{BP}^{\kappa}$, where $\kappa \geq 0$ is a real valued parameter that controls the extent of damping in the iterative equations. The exact details of the corresponding update equations are not essential for understanding the rest of this article; for completeness, we include the update equations for SAT in Figure 2 of Appendix A.

The damped equations are analogous to standard BP for SAT,[3] differing only in the added κ exponent in the iterative update equations. When $\kappa = 1$, BP^{κ} is identical to regular belief propagation. On the other hand, when $\kappa = 0$, the equations surely converge in one step to a unique fixed point and the marginal estimates obtained from this fixed point have a clear probabilistic interpretation in terms of a local property of the variables (we defer formal details of this property to Appendix A; see Proposition 1 and the related discussion). The κ parameter thus allows one to continuously interpolate between two regimes: one with $\kappa = 1$ where the equations are identical to standard BP equations and thus provide global information about the solution space if the iterations converge, and another with $\kappa = 0$ where the iterations surely converge but provide only local information about the solution space. In practice, κ is chosen to be roughly the highest value in the range $[0, 1]$ that allows convergence of the equations within a few seconds or less.

We use the output of BP^{κ} as an estimate of the marginals of the variables in `BPCount` (rather than solution samples as in `SampleCount`). Given this process of obtaining marginal estimates from BP, <u>`BPCount`</u> works almost exactly like `SampleCount` and *provides the same lower bound guarantees.* The only difference between the two algorithms is the manner in which marginal probabilities of variables is estimated. Formally,

Theorem 1 (Adapted from [10]) *Let s denote the number of variables randomly set by an iteration of `BPCount`, M' denote the number of solutions in the final residual formula given to an exact model counter, and $\alpha > 0$ be the slack parameter used. If `BPCount` is run with $t \geq 1$ iterations on a formula F, then its output—the minimum of $2^{s-\alpha} \times M'$ over the t iterations—is a correct lower bound on #F with probability at least $1 - 2^{-\alpha t}$.*

As the exponentially nature of the quantity $1 - 2^{-\alpha t}$ suggests, the correctness confidence for `BPCount` can be easily boosted by increasing the number of iterations, t, (thereby incurring a higher runtime), or by increasing the slack parameter, α, (thereby reporting a somewhat smaller lower bound and thus being conservative), or by a combination of both. In our experiments, we will aim for a correctness confidence of over 99%, by using values

[3] See, for example, Figure 4 of [16] with $\rho = 0$ for a full description of standard BP for SAT.

of t and α satisfying $\alpha t \geq 7$. Specifically, most runs will involve 7 iterations and $\alpha = 1$, while some will involve fewer iterations with a slightly higher value of α.

3.1 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 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 $(q \times pM/q) + ((1-q) \times (1-p)M/(1-q)) = M$ 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 counts 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.

3.2 Safety Checks

One issue that arises when using BP techniques to estimate marginals is that the estimates, in some cases, 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, which means 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` [7] or `Walksat` [26] in our implementation) before fixing the value of any variable. Note that using a SAT solver as a safety check is a powerful but somewhat expensive mechanism; fortunately, compared to the problem of counting solutions, the time to run a SAT solver as a safety check is relatively minor and did not result in any significant slow down in the instances we experimented with. The cost of running a SAT solver to find a solution is also significantly less than the cost other methods such as `ApproxCount` and `SampleCount` incur when collecting several near-uniform solution samples.

The idea behind the safety check is to simply ensure that there exists at least one solution both with $u =$ TRUE and with $u =$ FALSE, *before* flipping a random coin and fixing u to TRUE or to FALSE. If, say, `MiniSat` as the safety check solver finds 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 to continue with; no random coin is flipped in this case. If not, we run `MiniSat` with u forced to be FALSE. If `MiniSat` finds the formula to be unsatisfiable, we can immediately deduce $u =$ TRUE, simplify the formula, and look for a different balanced variable to continue with; again no random coin is flipped in this case. If not, i.e., MiniSat found solutions both with *u* set to TRUE and *u* set to FALSE, then *u* is said to pass the safety check—it is safe to flip a coin and fix the value of *u* randomly. This safety check prevents BPCount from reaching the undesirable state where there are no remaining solutions at all in the residual search space.

A slight variant of such a test can also be performed—albeit in a conservative fashion—with an incomplete solver such as Walksat. This works as follows. If Walksat is unable to find at least one solution both with *u* being TRUE and *u* being FALSE, we conservatively assume that it is not safe to flip a coin and fix the value of *u* randomly, and instead look for another variable for which Walksat *can* find solutions both with value TRUE and value FALSE. In the rare case that no such safe variable is found after a few tries, we call this a failed run of BPCount, and start from the beginning with possibly a higher cutoff for Walksat or a different safety check solver.

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 solution samples, i.e., if *u* takes value TRUE in all solution samples at a point, we conservatively assume that it is not safe to assign a random truth value to *u*.

Remark 1 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 fake 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 the values of several variables are fixed sequentially. With the use of BP rather than randomly generated fake marginals, however, the quality of the lower bounds was significantly improved, showing that BP does provide useful information about marginals even for highly loopy formulas.

4 Upper Bound Estimation Using Backtrack Search: MiniCount

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 backtrack 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—in contrast, recall that the correctness confidence expression $1-2^{-\alpha t}$ for the lower bound in Theorem 1 was independent of the nature of the underlying formula or the marginal estimation process. The applicability of the method will also be determined by a statistical test, which did succeed in most of our experiments.

For BPCount, we used a backtrack-less 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 DPLL-based backtrack search SAT solver MiniSat [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, there is no built-in specialized heuristic to decide which value the variable should 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. Note that there are other SAT solvers with this feature, e.g. `Rsat` [20], and similar results can be obtained for such solvers as well.

Algorithm MiniCount: Given a formula F, run `MiniSat`, choosing the truth value assignment for the variable selected at each choice point uniformly at random between TRUE and FALSE (command-line option `-polarity-mode=rnd`). When a solution is found, output 2^d, where d is the "perceived depth", i.e., 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 (a backtrack point). We rely on the fact that the default implementation of `MiniSat` never restarts unless it has backtracked at least once.[4]

We note that we are implicitly using the fact that `MiniSat`, and most SAT solvers available today, assign truth values to *all* variables of the formula when they declare that a solution has been found. In case the underlying SAT solver is designed to detect the fact that all clauses have been satisfied and to then declare that a solution has been found even with, say, u variables remaining unset, the definition of d should be modified to include these u variables; i.e., d should be u plus the number of choice points on the path minus the number of backtrack points on that path.

Note also that for an N variable formula, d can be alternatively defined as N minus the number of unit propagations on the path to the solution found minus the number of backtrack points on that path. This makes it clear that d is after all tightly related to N, in the sense that if we add a few "don't care" variables to the formula, the value of d will increase appropriately.

We now prove that we can use `MiniCount` to obtain an upper bound on the true model count of F. 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 theorem forms the basis of our upper bound estimation. We note that the theorem provides an essential building block but by itself does not fully justify the statistical estimation techniques we will introduce later; they rely on arguments discussed after the theorem.

Theorem 2 *For any CNF formula F, $\mathbb{E}[\#F_{\text{MiniCount}}] \geq \#F$.*

Proof The expected value is taken across all possible choices made by the `MiniCount` algorithm when run on F, i.e., all its possible computation histories on F. The proof uses the fact that the claimed inequality holds even if all computation histories that incurred at least one backtrack were modified to output 0 instead of 2^d once a solution was found. In other words, we will write the desired expected value, by definition, as a sum over all computation histories h and then simply discard a subset of the computation histories—those that involve at least one backtrack—from the sum to obtain a smaller quantity, which will eventually be shown to equal #F exactly.

Once we restrict ourselves to only those computation histories h that do not involve any backtracking, these histories correspond one-to-one to the paths p in the search tree underlying `MiniCount` that lead to a solution. Note that there are precisely as many such paths p as there are satisfying assignments for F. Further, since value choices of `MiniCount` at

[4] In a preliminary version of this work [14], we did not allow restarts at all. The reasoning given here extends the earlier argument and permits restarts as long as they happen after at least one backtrack.

various choice points are made independently at random, the probability that a computation history follows path p is precisely $1/2^{d_p}$, where d_p is the "perceived depth" of the solution at the leaf of p, i.e., the number of choice points till the solution is found (recall that there are no backtracks on this path; of course, there might—and often will—be unit propagations along p, due to which d_p may be smaller than the total number of variables in F). The value output by `MiniCount` on this path is 2^{d_p}.

Mathematically, the above reasoning can be formalized as follows:

$$
\begin{aligned}
\mathbb{E}\left[\#F_{\text{MiniCount}}\right] &= \sum_{\substack{\text{computation histories } h \\ \text{of MiniCount on } F}} \Pr[h] \cdot \text{output on } h \\
&\geq \sum_{\substack{\text{computation histories } h \\ \text{not involving any backtrack}}} \Pr[h] \cdot \text{output on } h \\
&= \sum_{\substack{\text{search paths } p \\ \text{that lead to a solution}}} \Pr[p] \cdot \text{output on } p \\
&= \sum_{\substack{\text{search paths } p \\ \text{that lead to a solution}}} \frac{1}{2^{d_p}} \cdot 2^{d_p} \\
&= \text{number of search paths } p \text{ that lead to a solution} \\
&= \#F
\end{aligned}
$$

This concludes the proof. □

Remark 2 The reason *restarts without at least one backtrack are not allowed* in `MiniCount` is hidden in the proof of Theorem 2. With such early restarts, only solutions reachable within the current setting of the restart threshold can be found. For restarts shorter than the number of variables, only "easier" solutions which require very few decisions are ever found. `MiniCount` with early restarts could therefore always undercount the number of solutions and not provide an upper bound—even in expectation. On the other hand, if restarts happen only after at least one backtrack point, then the proof of the above theorem shows that it is safe to even output 0 on such runs and still obtain a correct upper bound in expectation; restarting and reporting a non-zero number on such runs only helps the upper bound.

With enough random samples of the output, $\#F_{\text{MiniCount}}$, obtained from `MiniCount`, their average value will eventually converge to $\mathbb{E}[\#F_{\text{MiniCount}}]$ by the Law of Large Numbers [cf. 8], thereby providing an upper bound on $\#F$ because of Theorem 2. 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 $\#F_{\text{MiniCount}}$ often has significant mass in fairly high values and it might take very many samples for the sample mean to become as large as the true average of the distribution.

The way we proved Theorem 2, in fact, suggests that we could simply report 0 and start over every time we need to backtrack, which would actually result in a random variable that is in expectation *exact*, not only an upper bound. This approach is of course impractical, as we would almost always see zeros in the output and see a very high non-zero output with exponentially small probability. Although the expected value of these numbers is, in principle, the true model count of F, estimating the expected value of the underlying extreme zero-heavy 'bimodal' distribution through a few random samples is infeasible in practice. We therefore choose to trade off tightness of the reported bound for the ability to obtain values that can be argued about, as discussed next.

4.1 Justification for Using Statistical Techniques

As remarked earlier, the proof of Theorem 2 by itself does not provide a good justification for using statistical estimation techniques to compute $\mathbb{E}[\#F_{\text{MiniCount}}]$. This is because for the sake of the proving that what we obtain in expectation is an upper bound, we simplified the scenario and showed that it is sufficient to even report 0 solutions and start over whenever we need to backtrack. While these 0 outputs are enough to guarantee that we obtain an upper bound in expectation, they are by no means helpful in letting us estimate, in practice, the value of this expectation from a few samples of the output value. A bimodal distribution concentrated on 0 and with exponentially few very large numbers is difficult to estimate the expected value of. For the technique to be useful in practice, we need a smoother distribution for which we can use statistical estimation techniques, to be discussed shortly, in order to compute the expected value in a reasonable manner.

In order to achieve this, we will rely on an important observation: *when `MiniCount` does backtrack, we do* not *report* 0*; rather we continue to explore the other side of the "choice" point under consideration and eventually report a non-zero value.* Since our strategy will be to fit a statistical distribution on the output of several samples from `MiniCount`, and because except for rare occasions all of these samples come after at least one backtrack, it will be crucial that the non-zero value output by `MiniCount` when a solution is found *after* a backtrack does have information about the number of solutions of F. Fortunately, we argue that this is indeed the case—the value 2^d that `MiniCount` outputs even after at least one backtrack does contain valuable information about the number of solutions of F.

To see this, consider a stage in the algorithm that is perceived as a choice point but is in fact not a true choice point. Specifically, suppose at this stage, the formula has M solutions when $x = \text{TRUE}$ and no solutions when $x = \text{FALSE}$. With probability $1/2$, `MiniCount` will set x to TRUE and in fact estimate an upper bound on $2M$ from the resulting sub-formula, because it did not discover that it wasn't really at a "choice" point. This will, of course, still be a legal upper bound on M. More importantly, with probability $1/2$, `MiniCount` will set x to FALSE, discover that there are no solutions in this sub-tree, backtrack, set x to TRUE, realize that this is not actually a "choice" point, and recursively estimate an upper bound on M. Thus, even with backtracks, the output of `MiniCount` is very closely related to the actual number of solutions in the sub-tree at the current stage (unlike in the proof of Theorem 2, where it is thought of as being 0), and it is justifiable to deduce an upper bound on #F by fitting sample outputs of `MiniCount` to a statistical distribution. We also note that the number of solutions reported after a restart is just like taking another sample of the process with backtracks, and thus is also closely related to #F.

4.2 Estimating the Upper Bound Using Statistical Methods

In this section, we develop an approach based on statistical analysis of 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 m independent samples from $\#F_{\text{MiniCount}}$ by running `MiniCount` m 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 m 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 (equivalent to a 99% correctness confidence as before for the lower bounds), 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 the *statistical guarantee* that $c_{\max} \geq \mathbb{E}[\#F_{\text{MiniCount}}]$, and thus $c_{\max} \geq \#F$ by Theorem 2:

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

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 this statement is true only if our assumption of log-normality of the outputs of single runs of MiniCount on the given formula holds.

4.2.1 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 for whether Y is normally distributed can be used, and we use the Shapiro-Wilk test [cf. 27] 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 the diagonal line, the more likely it is that the data came from the desired distribution. See Figure 1 for some examples of QQ-plots.

We found that a surprising number of formulas had $\log_2(\#F_{\text{MiniCount}})$ very close to being normally distributed. Figure 1 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: $(\text{normalized } d_{\text{MiniCount}} = d) \sim \frac{1}{\sqrt{2\pi}} e^{-d^2/2}$. The 'supernormal' and 'subnormal' lines show that the fit is much worse when the exponent of d in the expression $e^{-d^2/2}$ above

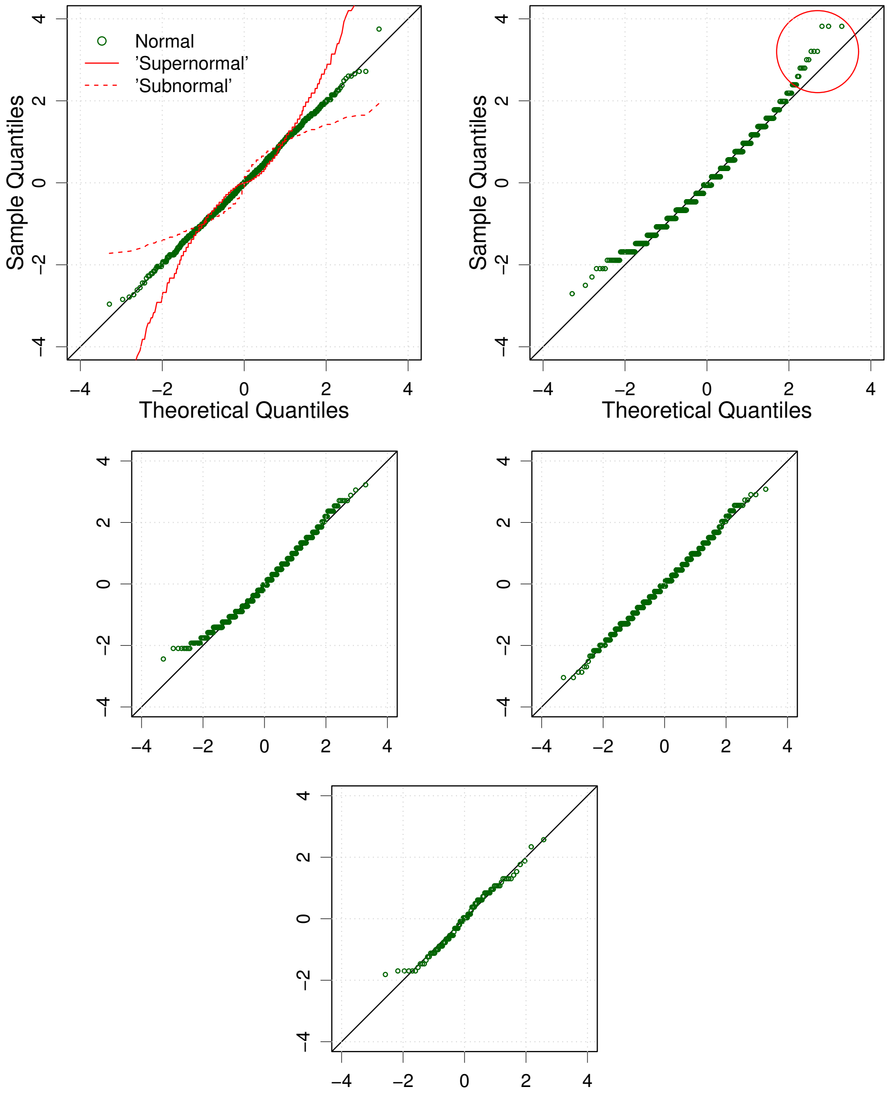

Fig. 1 Sampled and theoretical quantiles for formulas described in the experimental section (top: `alu2_gr_rcs_w8` and `lang19`; middle: `2bitmax_6` and `wff-3-150-525`; bottom: `ls11-norm`).

is, for example, taken to be 2.5 or 1.5. The top-right plot shows that $\#F_{\text{MiniCount}}$ on 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.

4.2.2 Confidence Interval Bound

Assuming the output samples from `MiniCount` $\{o_1, \ldots, o_m\}$ 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 [34] which yields $\tilde{c}_{\max}$, a quantity that is no smaller than $c_{\max}$. Let $y_i = \log(o_i)$, $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$ denote the sample mean, and $s^2 = \frac{1}{m-1}\sum_{i=1}^{m}(y_i - \bar{y})^2$ the sample variance. Then the conservative upper bound is constructed as

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

where $\chi^2_{\alpha}(m-1)$ is the α-percentile of the chi-square distribution with $m-1$ degrees of freedom. Since $\tilde{c}_{\max} \geq c_{\max}$, it follows from Equation (1) that

$$\Pr\left[\tilde{c}_{\max} \geq \#F\right] \;\geq\; 1 - \alpha \qquad (2)$$

This is the inequality that we will use when reporting our experimental results.

4.3 Limitations of `MiniCount` and Worst-Case Behavior

The main assumption of the upper bounding method described in this section is that the distribution of $\#F_{\text{MiniCount}}$ can be well approximated by a log-normal. This, of course, depends on the nature of the search process of `MiniCount` on the particular SAT instance under consideration. In particular, the resulting distribution could, in principle, vary significantly if the parameters of the underlying `MiniSat` solver are altered or if a different DPLL-based SAT solver is used as the basis of this model counting strategy. For some scenarios (i.e., for some solver-instance combinations), we might be able to have high confidence in log-normality, and for other scenarios, we might not and thus not claim an upper bound with this method. We found that using `MiniSat` with default parameters and with the random polarity mode as the basis for `MiniCount` worked well on several families of formulas.

As noted earlier, the assumption that the distribution is log-normal 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 for nearly all DPLL-based underlying SAT solvers. Consider a problem P that consists of two non-interacting (i.e., on disjoint sets of variables) 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 only a few choice points and they 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 only 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 $\#F_{\text{MiniCount}}$ is log-normally distributed, but this fact may be left unnoticed by the log-normality tests we perform, potentially resulting in a false upper bound. This possibility of a false upper bound is a consequence of the inability to statistically 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,[5] 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 Tables 1 and 2.

The columns show the performance of BPCount (version 1.2LES, based on SampleCount version 1.2L but adding external BP-based marginals and safety checks, using Cachet version 1.2 once the instance under consideration is sufficiently simplified) and MiniCount (based on MiniSat version 2.0), compared against the exact solution counters Relsat (version 2.00), Cachet (version 1.2), and c2d (version 2.20)[6], and the lower bounding solution counter SampleCount (version 1.2L, using Cachet version 1.2 once the instance is sufficiently simplified). The tables show 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 while still allowing BP^{κ} iterations to converge to a fixed point 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% correctness confidence.

Tables 1 and 2 show that a significant improvement in efficiency is achieved when the BP marginal estimation is used through BPCount, rather than 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 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 $m = 100$ 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 of 0.05, that is, if the Shapiro-Wilk test reports a 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 discussion in Section 4.2.2. The results are summarized in the last set of columns in Tables 1 and 2. We report whether the

[5] As stated earlier, we allow restarts in MiniCount after at least one backtrack has occurred, unlike the preliminary version of this work [14] where we reported results without restarts. Although the results in the two scenarios are sometimes fairly close, we believe allowing restarts will be effective and even indispensable on harder problem instances. We thus report here numbers only with restarts. For completeness, the numbers for MiniCount without restarts are reported in Table 3 of Appendix B.

[6] We report counts obtained from the best of the three exact model counters for each instance; for all but the first instance, c2d exceeded the memory limit.

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. Numbers in **bold** indicate the dominating techniques, if any, for that instance, i.e., that the best bound (lower and upper separately) obtained in the least amount of time.

			`Cachet` / `Relsat` / `c2d` (exact counters)		`SampleCount` (99% confidence)		`BPCount` (99% confidence)			`MiniCount` (99% confidence)		
Instance	num. of vars	True Count (if known)	Models	Time	LWR-bound	Time	LWR-bound	Time	S-W Test	Average	UPR-bound	Time
COMBINATORIAL PROBS.												
Ramsey-20-4-5	190	—	$\geq 9.0\times10^{11}$	12 hrs[C]	$\geq 3.3\times10^{35}$	3.5 min	$\geq 5.2\times10^{30}$	1.7 min	$\surd$	1.9×10^{37}	$\leq$ **9.1×10^{42}**	**2.7 sec**
Ramsey-23-4-5	253	—	$\geq 8.4\times10^{6}$	12 hrs[C]	$\geq 1.4\times10^{31}$	53 min	$\geq 2.1\times10^{24}$	12 min	$\surd$	2.3×10^{37}	$\leq$ **5.5×10^{43}**	**11 min**
Schur-5-100	500	—	$\geq 1.0\times10^{14}$	12 hrs[C]	$\geq$ **1.3×10^{17}**	**20 min**	—	12 hrs	$\surd$	1.7×10^{21}	$\leq$ **1.0×10^{27}**	**52 sec**
Schur-5-140	700	—	—	12 hrs[C]	—	12 hrs	—	12 hrs		—	—	12 hrs
fclqcolor-18-14-11	603	—	$\geq 2.4\times10^{33}$	12 hrs[C]	$\geq$ **3.9×10^{50}**	**3.5 min**	—	12 hrs	$\surd$	1.3×10^{47}	$\leq$ **1.2×10^{51}**	**1.5 sec**
fclqcolor-20-15-12	730	—	$\geq 8.6\times10^{38}$	12 hrs[C]	$\geq$ **3.1×10^{57}**	**6 min**	—	12 hrs	$\surd$	2.1×10^{60}	$\leq$ **5.1×10^{66}**	**2 sec**
CIRCUIT SYNTH.												
2bitmax_6	252	2.1×10^{29}	**2.1×10^{29}**	**2 sec**[C]	$\geq 2.4\times10^{28}$	29 sec	$\geq 2.8\times10^{28}$	5 sec	$\surd$	2.4×10^{29}	$\leq 8.9\times10^{30}$	2 sec
3bitadd_32	8704	—	—	12 hrs[C]	$\geq$ **5.9×10^{1339}**	**32 min**	—	12 hrs		—	—	12 hrs
RANDOM *k*-CNF												
wff-3-3.5	150	1.4×10^{14}	1.4×10^{14}	7 min[C]	$\geq 1.6\times10^{13}$	4 min	$\geq 1.6\times10^{11}$	3 sec	$\surd$	9.8×10^{14}	$\leq 1.7\times10^{17}$	0.6 sec
wff-3-1.5	100	1.8×10^{21}	1.8×10^{21}	3 hrs[C]	$\geq 1.6\times10^{20}$	4 min	$\geq 1.0\times10^{20}$	1 sec	$\surd$	5.9×10^{20}	$\leq 2.7\times10^{22}$	0.5 sec
wff-4-5.0	100	—	$\geq 1.0\times10^{14}$	12 hrs[C]	$\geq 8.0\times10^{15}$	2 min	$\geq$ **1.0×10^{16}**	**2 sec**	$\surd$	1.0×10^{17}	$\leq$ **1.1×10^{19}**	**0.6 sec**
FPGA routing (SAT2002)												
apex7_*_w5	1983	—	$\geq 4.5\times10^{47}$	12 hrs[R]	$\geq 8.8\times10^{85}$	20 min	$\geq$ **2.9×10^{94}**	**3 min**	$\surd$	2.9×10^{94}	$\leq$ **2.2×10^{109}**	**2 min**
9symml_*_w6	2604	—	$\geq 5.0\times10^{30}$	12 hrs[R]	$\geq 2.6\times10^{47}$	6 hrs	$\geq 1.8\times10^{46}$	6 min	$\surd$	6.5×10^{55}	$\leq$ **4.8×10^{61}**	**52 sec**
c880_*_w7	4592	—	$\geq 1.4\times10^{43}$	12 hrs[R]	$\geq 2.3\times10^{273}$	5 hrs	$\geq 7.9\times10^{253}$	18 min	$\surd$	2.6×10^{260}	$\leq$ **1.6×10^{315}**	**6 sec**
alu2_*_w8	4080	—	$\geq 1.8\times10^{56}$	12 hrs[R]	$\geq 2.4\times10^{220}$	143 min	$\geq 2.0\times10^{205}$	16 min	$\surd$	2.3×10^{209}	$\leq$ **7.7×10^{248}**	**7 sec**
vda_*_w9	6498	—	$\geq 1.4\times10^{88}$	12 hrs[R]	$\geq 1.4\times10^{326}$	11 hrs	$\geq 3.8\times10^{289}$	56 min	$\surd$	4.1×10^{302}	$\leq$ **4.4×10^{410}**	**13 sec**

Table 2 (Continued from Table 1.) Performance of `BPCount` and `MiniCount`.

Instance	num. 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	S-W Test	`MiniCount` (99% confidence) Average	UPR-bound	Time
LATIN SQUARE												
ls8-norm	301	5.4×10^{11}	$\geq 1.7\times10^{8}$	12 hrs[R]	$\geq 3.1\times10^{10}$	19 min	$\geq \mathbf{1.9\times10^{10}}$	**12 sec**	$\times$	7.3×10^{14}	$\leq 5.6\times10^{14}$	1 sec
ls9-norm	456	3.8×10^{17}	$\geq 7.0\times10^{7}$	12 hrs[R]	$\geq 1.4\times10^{15}$	32 min	$\geq \mathbf{1.0\times10^{16}}$	**11 sec**	$\surd$	4.9×10^{19}	$\leq \mathbf{2.2\times10^{21}}$	**2 sec**
ls10-norm	657	7.6×10^{24}	$\geq 6.1\times10^{7}$	12 hrs[R]	$\geq 2.7\times10^{21}$	49 min	$\geq \mathbf{1.0\times10^{23}}$	**22 sec**	$\surd$	7.2×10^{25}	$\leq \mathbf{1.4\times10^{30}}$	**10 sec**
ls11-norm	910	5.4×10^{33}	$\geq 4.7\times10^{7}$	12 hrs[R]	$\geq 1.2\times10^{30}$	69 min	$\geq \mathbf{6.4\times10^{30}}$	**1 min**	$\surd$	3.5×10^{33}	$\leq \mathbf{1.3\times10^{41}}$	**100 sec**
ls12-norm	1221	—	$\geq 4.6\times10^{7}$	12 hrs[R]	$\geq 6.9\times10^{37}$	50 min	$\geq \mathbf{2.0\times10^{41}}$	**70 sec**	$\times$	3.3×10^{43}	$\leq 3.8\times10^{54}$	10 min
ls13-norm	1596	—	$\geq 2.1\times10^{7}$	12 hrs[R]	$\geq 3.0\times10^{49}$	67 min	$\geq \mathbf{4.0\times10^{54}}$	**6 min**	$\times$	1.2×10^{51}	$\leq 3.5\times10^{62}$	1.5 hrs
ls14-norm	2041	—	$\geq 2.6\times10^{7}$	12 hrs[R]	$\geq 9.0\times10^{60}$	44 min	$\geq \mathbf{1.0\times10^{67}}$	**4 min**	$\surd$	3.0×10^{65}	$\leq \mathbf{3.7\times10^{88}}$	**12 hrs**
ls15-norm	2041	—	$\geq 9.1\times10^{6}$	12 hrs[R]	$\geq 1.1\times10^{73}$	56 min	$\geq 5.8\times10^{84}$	12 hrs		—	—	12 hrs
ls16-norm	2041	—	$\geq 1.0\times10^{7}$	12 hrs[R]	$\geq \mathbf{6.0\times10^{85}}$	**68 min**	—	12 hrs		—	—	12 hrs
LANGFORD PROBS.												
lang-2-12	576	1.0×10^{5}	1.0×10^{5}	15 min[R]	$\geq 4.3\times10^{3}$	32 min	$\geq 2.3\times10^{3}$	50 sec	$\times$	2.5×10^{6}	$\leq 8.8\times10^{6}$	1 sec
lang-2-15	1024	3.0×10^{7}	$\geq 1.8\times10^{5}$	12 hrs[R]	$\geq 1.0\times10^{6}$	60 min	$\geq 5.5\times10^{5}$	1 min	$\times$	8.1×10^{8}	$\leq 4.1\times10^{9}$	2.5 sec
lang-2-16	1024	3.2×10^{8}	$\geq 1.8\times10^{5}$	12 hrs[R]	$\geq 1.0\times10^{6}$	65 min	$\geq 3.2\times10^{5}$	1 min	$\surd$	3.4×10^{7}	$\leq \mathbf{5.9\times10^{8}}$	**2 sec**
lang-2-19	1444	2.1×10^{11}	$\geq 2.4\times10^{5}$	12 hrs[R]	$\geq 3.3\times10^{9}$	62 min	$\geq 4.7\times10^{7}$	26 min	$\times$	9.2×10^{9}	$\leq 2.5\times10^{11}$	3.7 sec
lang-2-20	1600	2.6×10^{12}	$\geq 1.5\times10^{5}$	12 hrs[R]	$\geq 5.8\times10^{9}$	54 min	$\geq 7.1\times10^{4}$	22 min	$\times$	1.1×10^{13}	$\leq 6.7\times10^{13}$	4 sec
lang-2-23	2116	3.7×10^{15}	$\geq 1.2\times10^{5}$	12 hrs[R]	$\geq 1.6\times10^{11}$	85 min	$\geq 1.5\times10^{5}$	15 min	$\surd$	1.5×10^{13}	$\leq \mathbf{7.9\times10^{15}}$	**6 sec**
lang-2-24	2304	—	$\geq 4.1\times10^{5}$	12 hrs[R]	$\geq 4.1\times10^{13}$	80 min	$\geq 8.9\times10^{7}$	18 min	$\times$	3.0×10^{13}	$\leq 1.5\times10^{16}$	7 sec
lang-2-27	2916	—	$\geq 1.1\times10^{4}$	12 hrs[R]	$\geq 5.2\times10^{14}$	111 min	$\geq 1.3\times10^{6}$	80 min	$\surd$	3.3×10^{15}	$\leq \mathbf{3.4\times10^{16}}$	**7 sec**
lang-2-28	3136	—	$\geq 1.1\times10^{4}$	12 hrs[R]	$\geq 4.0\times10^{14}$	117 min	$\geq 9.0\times10^{2}$	70 min	$\surd$	1.2×10^{14}	$\leq \mathbf{3.3\times10^{16}}$	**11 sec**

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.

Tables 1 and 2 show 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 can provide 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 as an effective strategy for obtaining more robust upper bounds.

We end this section with the observation that while the lower and upper bounds provided by BPCount and MiniCount, respectively, are in general of very good quality, there is still a gap in the exponent. From the results for the cases where the true solution count for the instance is known, we can see that either of these bounds can be closer to the true count than the other. For example, the lower bound reported by BPCount is tigher than the upper bound reported by MiniCount in the case of the Latin Square construction problem, the opposite holds for the Langford problem, and the true count lies roughly in the middle (in log-scale) for the randomly generated problem. This attests to the hardness of the model counting problem and leaves open room for further improvement in techniques for obtaining both lower bounds and upper bounds on the true count.

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, competitive lower bounds in a fraction of the time compared to previous, sampling-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 solver into an upper bound counter, with very minimal modifications to the code. Our MiniCount algorithm provably converges to an upper bound on the solution count as more and more samples are drawn, and a statistical estimate of this upper bound can be efficiently derived from just a few samples assuming an independently verified log-normality condition. MiniCount is shown to be remarkably fast at providing good upper bounds in practice.

Acknowledgements This work was supported by the Intelligent Information Systems Institute, Cornell University (Air Force Office of Scientific Research AFOSR, grant FA9550-04-1-0151), the Defence Advanced Research Projects Agency (DARPA, REAL Program, grant FA8750-04-2-0216), and the National Science Foundation (NSF IIS award, grant 0514429; NSF Expeditions in Computing award for Computational Sustainability, grant 0832782).

References

1. F. Bacchus, S. Dalmao, and T. Pitassi. Solving #SAT and Bayesian inference with backtracking search. *Journal of Artificial Intelligence Research*, 34:391–442, 2009.
2. R. J. Bayardo Jr. and J. D. Pehoushek. Counting models using connected components. In *Proceedings of AAAI-00: 17th National Conference on Artificial Intelligence*, pages 157–162, Austin, TX, July 2000.
3. A. Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of ECAI-04: 16th European Conference on Artificial Intelligence*, pages 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. *Communications of the ACM*, 5:394–397, 1962.
6. M. Davis and H. Putnam. A computing procedure for quantification theory. *Communications of the ACM*, 7:201–215, 1960.
7. N. Eén and N. Sörensson. MiniSat: A SAT solver with conflict-clause minimization. In *Proceedings of SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing*, St. Andrews, U.K., June 2005.
8. W. Feller. *An Introduction to Probability Theory and Its Applications, Volume 1*. Wiley, 3rd edition, 1968.
9. V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *Proceedings of AAAI-07: 22nd Conference on Artificial Intelligence*, pages 198–203, Vancouver, BC, July 2007.
10. C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *Proceedings of IJCAI-07: 20th International Joint Conference on Artificial Intelligence*, pages 2293–2299, Hyderabad, India, Jan. 2007.
11. C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proceedings of AAAI-06: 21st Conference on Artificial Intelligence*, pages 54–61, Boston, MA, July 2006.
12. C. P. Gomes, W.-J. van Hoeve, A. Sabharwal, and B. Selman. Counting CSP solutions using generalized XOR constraints. In *Proceedings of AAAI-07: 22nd Conference on Artificial Intelligence*, pages 204–209, Vancouver, BC, July 2007.
13. E. I. Hsu and S. A. McIlraith. Characterizing propagation methods for boolean satisfiabilty. In *Proceedings of SAT-06: 9th International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 325–338, Seattle, WA, Aug. 2006.
14. L. Kroc, A. Sabharwal, and B. Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. In *CPAIOR-08: 5th International Conference on Integration of AI and OR Techniques in Constraint Programming*, volume 5015 of *Lecture Notes in Computer Science*, pages 127–141, Paris, France, May 2008.
15. M. L. Littman, S. M. Majercik, and T. Pitassi. Stochastic Boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.
16. E. Maneva, E. Mossel, and M. J. Wainwright. A new look at survey propagation and its generalizations. *Journal of the ACM*, 54(4):17, July 2007.
17. J. M. Mooij, B. Wemmenhove, H. J. Kappen, and T. Rizzo. Loop corrected belief propagation. In *Proceedings of AISTATS-07: 11th International Conference on Artificial Intelligence and Statistics*, San Juan, Puerto Rico, Mar. 2007.

18. J. D. Park. MAP complexity results and approximation methods. In *Proceedings of UAI-02: 18th Conference on Uncertainty in Artificial Intelligence*, pages 388–396, Edmonton, Canada, Aug. 2002.
19. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
20. K. Pipatsrisawat and A. Darwiche. RSat 1.03: SAT solver description. Technical Report D–152, Automated Reasoning Group, Computer Science Department, UCLA, 2006.
21. M. Pretti. A message-passing algorithm with damping. *Journal of Statistical Mechanics*, P11008, 2005.
22. D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2): 273–302, 1996.
23. T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proceedings of SAT-04: 7th International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, BC, May 2004.
24. T. Sang, P. Beame, and H. A. Kautz. Heuristics for fast exact model counting. In *Proceedings of SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 226–240, St. Andrews, U.K., June 2005.
25. T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *Proceedings of AAAI-05: 20th National Conference on Artificial Intelligence*, pages 475–482, Pittsburgh, PA, July 2005.
26. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring and Satisfiability: the Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532. American Mathematical Society, 1996.
27. H. C. Thode. *Testing for Normality*. CRC, 2002.
28. L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
29. W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In *Proceedings of AAAI-04: 19th National Conference on Artificial Intelligence*, pages 670–676, San Jose, CA, July 2004.
30. W. Wei and B. Selman. A new approach to model counting. In *Proceedings of SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 324–339, St. Andrews, U.K., June 2005.
31. J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *Proceedings of NIPS-00: 14th Conference on Advances in Neural Information Processing Systems*, pages 689–695, Denver, CO, Nov. 2000.
32. J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.
33. A. L. Yuille. CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722, 2002.
34. X.-H. Zhou and G. Sujuan. Confidence intervals for the log-normal mean. *Statistics In Medicine*, 16:783–790, 1997.

Appendix

A Update Equations for BP$^\kappa$, a Convergent Variant of BP

The iterative update equations for the convergent form of belief propagation, BP^κ, are given in Figure 2. The only difference from the normal BP equations is the exponent κ in the updates.

Notation Used. $V(a)$: all variables in clause a. $C_a^u(i), i \in V(a)$: clauses where variable i appears with the *opposite* sign than it does in a. $C_a^s(i), i \in V(a)$: clauses where i appears with the *same* sign as it does in a (not including a itself).

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

Computing marginals from a fixed point η^* of the message equations:

$$\mu_i(1) \propto \prod_{b\in C^-(i)} (1-\eta^*_{b\to i})$$
$$\mu_i(0) \propto \prod_{b\in C^+(i)} (1-\eta^*_{b\to i})$$

$\mu_i(1)$ is the probability that variable i is positive in a satisfying assignment chosen uniformly at random, and $\mu_i(0)$ is the probability that it is negative. $\mu_i(0)+\mu_i(1)=1$.

Fig. 2 Modified belief propagation update equations.

The role of the parameter κ is to damp oscillations of the message values by pushing the variable-to-clause messages, Π, closer to 1. Intuitively speaking, the damping is realized by the function $y = x^\kappa$ for $\kappa < 1$. For inputs x that are positive and less than one, the function increases their value, or sets them to 1 in the case of $\kappa = 0$. As a result, after normalization, the Π values are less extreme. For $\kappa = 0$, we can obtain a probabilistic interpretation of the algorithm reminiscent of a local heuristic for SAT solving:

Proposition 1 *The system of BP^κ equations for $\kappa = 0$ converges in one iteration for any starting point, and the following holds for the resulting values μ_i (see Figure 2 for notation):*

$$\mu_i(1) \propto \prod_{b\in C^-(i)} \left(1-2^{-(|V(b)|-1)}\right)$$
$$\mu_i(0) \propto \prod_{b\in C^+(i)} \left(1-2^{-(|V(b)|-1)}\right)$$

Proof For any initial starting point η_0 (with values in $[0,1]$), the first iteration sets all $\Pi^u = 1$ and $\Pi^s = 1$. This means $\eta_{a\to i} = (\frac{1}{2})^{|V(a)|-1}$ for all clauses a containing variable i. This is the fixed point η^*, because applying the updates again yields the same values. The rest follows directly from the form of the μ_i equations in Figure 2. □

The intuitive interpretation of the values of μ_i in Proposition 1 is the following: assuming independence of variable occurrences in clauses, the value $2^{-(|V(b)|-1)}$ can be interpreted as the probability that clause $b \in C(i)$ is unsatisfied by a random truth assignment if variable i is not considered. $\mu_i(1)$ is thus the probability that all clauses b in which variable i appears negated are satisfied, and analogously for $\mu_i(0)$. Since clauses not considered in the expressions for μ_i are satisfied by i itself, the resulting values of μ_i are proportional to *the probability that all clauses where i appears are satisfied with the particular setting of variable i, when all other variables are set randomly.* This is very local information, and depends only on what clauses the variable appears negatively or positively in. The parameter κ can thus be used to tune the tradeoff between the ability of the iterative system to converge and the locality of the information obtained from the resulting fixed point.

B Performance of MiniCount Without Restarts

For completeness, we report here the upper bound results obtained using `MiniCount` with restarts turned off—the setup that was used in a preliminary version of this article [14]. In most cases, this is somewhat slower than running `MiniCount` with restarts. However, there are cases where this is somewhat faster or where the resulting numbers appear to fit a log-normal distribution better than with restarts.

Table 3 Performance of `MiniCount` without restarts.

Instance	num. of vars	True Count (if known)	S-W Test	`MiniCount` **without restarts** Average	(99% confidence) UPR-bound	Time
CIRCUIT SYNTH.						
2bitmax_6	252	2.1×10^{29}	$\surd$	3.5×10^{30}	$\leq 4.3 \times 10^{32}$	2 sec
RANDOM *k*-CNF						
wff-3-3.5	150	1.4×10^{14}	$\surd$	4.3×10^{14}	$\leq 6.7 \times 10^{15}$	2 sec
wff-3-1.5	100	1.8×10^{21}	$\surd$	1.2×10^{21}	$\leq 4.8 \times 10^{22}$	2 sec
wff-4-5.0	100	—	$\surd$	2.8×10^{16}	$\leq 5.7 \times 10^{28}$	2 sec
LATIN SQUARE						
ls8-norm	301	5.4×10^{11}	$\surd$	6.4×10^{12}	$\leq 1.8 \times 10^{14}$	2 sec
ls9-norm	456	3.8×10^{17}	$\surd$	6.9×10^{18}	$\leq 2.1 \times 10^{21}$	3 sec
ls10-norm	657	7.6×10^{24}	$\surd$	4.3×10^{26}	$\leq 7.0 \times 10^{30}$	7 sec
ls11-norm	910	5.4×10^{33}	$\surd$	1.7×10^{34}	$\leq 5.6 \times 10^{40}$	35 sec
ls12-norm	1221	—	$\surd$	9.1×10^{44}	$\leq 3.6 \times 10^{52}$	4 min
ls13-norm	1596	—	$\surd$	1.0×10^{54}	$\leq 8.6 \times 10^{69}$	42 min
ls14-norm	2041	—	$\surd$	3.2×10^{63}	$\leq 1.3 \times 10^{86}$	7.5 hrs
LANGFORD PROBS.						
lang-2-12	576	1.0×10^{5}	$\times$	5.2×10^{6}	$\leq 1.0 \times 10^{7}$	2.5 sec
lang-2-15	1024	3.0×10^{7}	$\surd$	1.0×10^{8}	$\leq 9.0 \times 10^{8}$	8 sec
lang-2-16	1024	3.2×10^{8}	$\times$	1.1×10^{10}	$\leq 1.1 \times 10^{10}$	7.3 sec
lang-2-19	1444	2.1×10^{11}	$\times$	1.4×10^{10}	$\leq 6.7 \times 10^{12}$	37 sec
lang-2-20	1600	2.6×10^{12}	$\surd$	1.4×10^{12}	$\leq 9.4 \times 10^{12}$	3 min
lang-2-23	2116	3.7×10^{15}	$\times$	3.5×10^{12}	$\leq 1.4 \times 10^{13}$	23 min
lang-2-24	2304	—	$\times$	2.7×10^{13}	$\leq 1.9 \times 10^{16}$	25 min
FPGA routing (SAT2002)						
apex7_*_w5	1983	—	$\surd$	7.3×10^{95}	$\leq 5.9 \times 10^{105}$	2 min
9symml_*_w6	2604	—	$\surd$	3.3×10^{58}	$\leq 5.8 \times 10^{64}$	24 sec
c880_*_w7	4592	—	$\surd$	1.0×10^{264}	$\leq 6.3 \times 10^{326}$	26 sec
alu2_*_w8	4080	—	$\surd$	1.4×10^{220}	$\leq 7.2 \times 10^{258}$	16 sec
vda_*_w9	6498	—	$\surd$	1.6×10^{305}	$\leq 2.5 \times 10^{399}$	42 sec