A Simple Application of Sampling Importance Re-sampling (SIR) for Solution Sampling

Vibhav Gogate and Rina Dechter

Donald Bren School of Information and Computer Science
University of California, Irvine, CA 92697, USA

Abstract. We introduce a new technique of SampleSearch-SIR to generate random solutions of a Boolean satisfiability problem from a uniform distribution over the solutions. Our technique operates in two phases. In the first phase, it uses a recently proposed SampleSearch scheme to generate approximately random solutions from the satisfiability problem and then in the second phase it uses the Sampling Importance Resampling (SIR) principle to reduce the approximation error introduced by SampleSearch. The use of SIR guarantees convergence (in the limit) that none of the current state-of-the-art schemes have. Our empirical results demonstrate the superior performance and better convergence of SampleSearch-SIR as compared to state-of-the-art schemes.

1 Introduction

In this paper, we present a new algorithm to solve the solution sampling task which is the task of generating solutions from a uniform distribution over the solutions of a Boolean satisfiability problem. As pointed out in [15, 2, 6], the sampling task has wide range of applications in fields such as Verification, probabilistic reasoning and Distributed AI.

The problem of test program generation in the functional verification domain, for example, can be modeled as a random sampling task. The main vehicle for verification of large and complex hardware designs is simulation of a large number of random test programs [1]. These large hardware designs can be modeled as Boolean satisfiability problems in which case the test programs are its solutions. Typically, the number of solutions of the modeled program could be as large as 10^{1000} and the typical number of selected test programs are in the range of 10^3 or 10^4. Naturally, the best test generator is the one that would uniformly sample the space of test programs which translates to solving the random sampling task.

Theoretically, the random sampling task is closely related to the #P-Complete problem of counting the number of solutions of a satisfiability problem. In fact, it is known [7] that if one can sample uniformly from the set of solutions then one can design a highly accurate method for counting solutions. Conversely [2], one can also design a highly efficient method for sampling solutions from an exact counting algorithm.

The principle that algorithms for counting can be used for sampling was exploited in [2] in which the authors show how an exact counting algorithm like Bucket (Variable) Elimination can be used to solve the random sampling task. The Bucket elimination algorithm, however, is exponential in a graph parameter called treewidth and is impractical when the treewidth is large. Therefore, in order to make their random sampling

algorithms practical, [2, 5] propose to use approximate solution counters based on Mini-bucket Elimination and Generalized Belief propagation instead of Bucket Elimination. Empirical results in [2, 5] show that the proposed approximate sampling schemes work quite well on loosely constrained SAT instances.

However, [5] point out that earlier schemes may fail to generate even a single solution for some hard SAT instances. The problem is that both Mini-bucket Elimination and Generalized Belief Propagation achieve only bounded consistency and therefore many inconsistencies are not detected. Consequently, a large number of samples generated from the output of Mini-bucket elimination and Generalized Belief Propagation are not solutions. Obviously, these non-solutions are irrelevant to the random sampling task and are effectively thrown away or rejected (*the rejection problem*) .

In [5], the authors circumvent the rejection problem by systematically searching for a solution when a sample is rejected instead of returning with a non-solution yielding the SampleSearch scheme. SampleSearch is basically a randomized backtracking procedure whose value selection is guided by sampling from the output of Mini-bucket Elimination or Generalized Belief Propagation. The experimental study in [5] showed that SampleSearch is quite competitive with state-of-the-art solution samplers.

In spite of the good empirical performance of SampleSearch, it has a fundamental shortcoming. For any sampling procedure, one expects that increasing the number of samples reduces the sampling error. However, SampleSearch does not have any such guarantees of convergence (even in the limit). In fact, in a recent work [4] which explores the use of SampleSearch for solution counting, we prove that the distribution over the solutions generated using SampleSearch would converge to the *backtrack-free distribution*. Because the backtrack-free distribution is different from the uniform distribution over the solutions, the use of SampleSearch seems to be problematic.

In this paper, we propose to circumvent this problem by using the Sampling Importance Resampling (SIR) principle yielding the SampleSearch-SIR scheme. Unlike pure SampleSearch, SampleSearch-SIR guarantees [13, 10] that as the number of samples increases, the expected sampling error goes to zero. We will explore various improvements to the pure SIR scheme such as (a) sampling without replacement [3] and (b) Improved SIR [12].

We present empirical comparison of SampleSearch-SIR with state-of-the-art solution sampling schemes of WALKSAT [15] and pure SampleSearch [5]. We found that SampleSearch-SIR has better performance in terms of sampling error than both WALKSAT and pure SampleSearch on most benchmarks. In particular, we observe that as the time increases (or as more samples are drawn), the sampling error of SampleSearch-SIR decreases approaching zero while the sampling error of WALKSAT and pure SampleSearch remains almost constant.

The rest of the paper is organized as follows. In section 2, we present preliminaries and previous work on the SampleSearch scheme. In section 3, we show how SampleSearch can be integrated with the Sampling Importance Resampling (SIR) framework. Empirical results are presented in section 4 and we end with a short discussion and summary in section 5.

2 Preliminaries and Related Work

We represent sets by bold capital letters and members of a set by capital letters. An assignment of a value to a variable is denoted by a small letter while bold small letters indicate an assignment to a set of variables. For the rest of the paper, let $|\mathbf{X}| = n$ be the propositional variables. A variable assignment $\mathbf{X} = \mathbf{x}$, $\mathbf{x} = (x_1, \ldots, x_n)$ assigns a value in $\{0,1\}$ to each variable in $\mathbf{X}$. We use the notation $\overline{x_i}$ to mean the negation of a value assignment x_i. Let $F = F_1 \wedge \ldots \wedge F_m$ be a formula in conjunctive normal form (cnf) with clauses $F_1, \ldots, F_m$ defined over $\mathbf{X}$ and let $\mathbf{X} = \mathbf{x}$ be a variable assignment. If $\mathbf{X} = \mathbf{x}$ satisfies all clauses of F, then $\mathbf{X} = \mathbf{x}$ is a model or a solution of F.

Definition 1 (The Random Sampling task). *Let $\mathbf{S} = \{\mathbf{x} | \mathbf{x}$ is a solution of $F\}$ be the set of models of formula F. Given a uniform probability distribution $\mathscr{P}(\mathbf{x} \in \mathbf{S}) = 1/|\mathbf{S}|$ over all solutions of F and an integer M, the random sampling task is to generate M solutions such that each solution is generated with probability $1/|\mathbf{S}|$.*

2.1 Previous approaches to solve the sampling task

An obvious way to solve the random sampling task is to first generate (and store) all solutions and then generate M samples from the stored solutions such that each solution is sampled with equal probability (see Algorithm 1). The problem with this approach is that there may be exponential number of solutions and it defeats the purpose of sampling because in most applications we resort to sampling because we cannot enumerate the entire solution search space.

Another approach developed by Dechter et al. [2] to solve the random sampling task works as follows. The authors first show how to express the uniform distribution $\mathscr{P}(\mathbf{x})$ over the solutions in the product form [1] $\mathscr{P}(\mathbf{x} = (x_1, \ldots, x_n)) = \prod_{i=1}^{n} P_i(x_i | x_1, \ldots, x_{i-1})$ and then use the standard Monte Carlo (MC) sampler (also called logic sampling [8]) to sample from the product form of $\mathscr{P}$. In particular, given an ordering of variables $O = \langle X_1, \ldots, X_n \rangle$, the sampling task can be solved using Algorithm 2. Here, at each step, given a partial assignment $(x_1, \ldots, x_{i-1})$ to the previous $i-1$ variables, a value is assigned to variable X_i by sampling it from the distribution $P_i(X_i | x_1, \ldots, x_{i-1})$. Repeating this process n times generates one sample.

The probability $P_i(X_i = x_i | x_1, \ldots, x_{i-1})$ is equal to the ratio of the number of solutions that the partial assignment $(x_1, \ldots, x_i)$ participates in and the number of solutions that the partial assignment $(x_1, \ldots, x_{i-1})$ participates in. Therefore, counting algorithms can be utilized to compute $P_i(X_i = x_i | x_1, \ldots, x_{i-1})$ and to this end Dechter

[1] This follows from standard probability theory (see [8])

Algorithm 1 SimpleSampler (F)

1: Generate all solutions of F and store them in a set $\mathbf{S} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$.
2: **for** $i = 1$ to M **do**
3: p = a random number between 1 and N
4: Output $\mathbf{x}^p$
5: **end for**

Algorithm 2 Monte-Carlo Sampler (F)

1: **for** $j = 1$ to k **do**
2: $\mathbf{x}=\phi$
3: **for** $i = 1$ to n **do**
4: p = a random real number between 0 and 1
5: **IF** $p < P_i(X_i = 0|\mathbf{x})$ **THEN** $\mathbf{x} = \mathbf{x} \cup (X_i = 0)$ **ELSE** $\mathbf{x} = \mathbf{x} \cup (X_i = 1)$
6: **end for**
7: Output $\mathbf{x}$
8: **end for**

et al. [2] utilize a variable-elimination based solution counting scheme to compute $P_i(X_i|x_1,\ldots,x_{i-1})$. In this scheme, a bucket (variable) elimination algorithm is run (just once) as a pre-processing step so that $P_i(X_i|x_1,\ldots,x_{i-1})$ can be computed for any partial assignment $(x_1,\ldots,x_{i-1})$ to the previous $i-1$ variables by performing a (constant time) table look-up. The time complexity of this scheme is exponential in a graph parameter called the treewidth (induced-width). Obviously, when the treewidth is large, the variable elimination based method is impractical.

To address the exponential blow up, Dechter et al. [2] propose to compute an approximation $Q_i(X_i|x_1,\ldots,x_{i-1})$ of $P_i(X_i|x_1,\ldots,x_{i-1})$ by running the mini-bucket elimination scheme instead of the bucket elimination scheme. Mini-bucket elimination (MBE) is an approximate counting algorithm whose time (and space) complexity is exponential in a user-specified parameter i and is therefore polynomial when i is constant.

However, the MBE-based scheme suffers from the so-called *rejection problem*. It is known that MBE only achieves bounded level of consistency and therefore may not make the SAT problem backtrack-free. Consequently, a sample generated from the approximation Q of P may not be a solution of the formula F. In some cases (e.g. for instances in the phase transition), the probability of sampling a solution from the approximation Q of P may be so small that almost all samples generated will be non-solutions. These non-solutions are irrelevant to the solution sampling task and are effectively thrown away or rejected.

Algorithm 3 *SampleSearch* $SS(F,Q,O)$

Input: a cnf formula F, a distribution Q and Ordering O
Output: A solution $\mathbf{x} = (x_1,\ldots,x_n)$

1: UnitPropagate(F)
2: **IF** there is an empty clause in F **THEN** Return 0
3: **IF** all variables are assigned a value **THEN** Return 1
4: Select the earliest variable X_i in O not yet assigned a value
5: p = Generate a random number between 0 and 1
6: **Value Assignment:** Given the partial assignment $(x_1,\ldots,x_{i-1})$
IF $p < Q_i(X_i = 0|x_1,\ldots,x_{i-1})$ **THEN** set $x_i = 0$ **ELSE** set $x_i = 1$.
7: Return $SS((F \wedge x_i),Q,O) \vee SS((F \wedge \overline{x_i}),Q,O)$

To circumvent the rejection problem, Gogate and Dechter [5] presented the SampleSearch scheme. Instead of returning with a sample that is inconsistent, SampleSearch progressively revises the inconsistent sample via DPLL-style backtracking search until a solution is found. SampleSearch is presented as Algorithm 3. It takes as input a formula F, an ordering $O = \langle X_1, \ldots, X_n \rangle$ of variables and a distribution $Q = \prod_{i=1}^{n} Q_i(x_i|x_1, \ldots, x_{i-1})$. Given a partial assignment $(x_1, ..., x_{i-1})$ already generated, the next variable in the ordering X_i is selected and its value $X_i = x_i$ is sampled from the conditional distribution $Q_i(x_i|x_1, \ldots, x_{i-1})$. Then the algorithm applies unit-propagation with the new unit clause $X_i = x_i$ created over the formula F. Note that the unit-propagation step is optional. Any backtracking style search can be used, with any of the current advances. If no empty clause is generated, then the algorithm proceeds with the next variable. Otherwise, the algorithm tries $X_i = \overline{x_i}$, performs unit propagation (or any level of constraint propagation) and either proceeds forward (if no empty clause generated) or it backtracks. On termination, the output of SampleSearch is a solution of F (assuming a solution exists).

Gogate and Dechter [5] showed that on most benchmarks SampleSearch is competitive with another approach for solution sampling based on the WALKSAT solver [15]. In this paper, however we point out a rather negative result for SampleSearch in that the samples generated do not converge to the uniform distribution over the solutions. We fix this problem by exploiting the Sampling Importance Resampling (SIR) principle yielding the SampleSearch-SIR scheme that is guaranteed to converge to the uniform distribution.

3 The SampleSearch-SIR scheme

3.1 Convergence of SampleSearch to the wrong distribution

In a recently accepted paper [4] which explores the use of SampleSearch for counting solutions, we proved that SampleSearch generates independent and identically distributed (i.i.d.) samples from the backtrack-free distribution which we define below:

Definition 2 (Backtrack-free distribution). *Given a distribution $Q(\mathbf{x}) = \prod_{i=1}^{n} Q_i(x_i|x_1, \ldots, x_{i-1})$ [2], an ordering $O = \langle x_1, \ldots, x_n \rangle$ and a cnf formula F, the backtrack-free distribution Q^F is given by $Q^F(\mathbf{x}) = \prod_{i=1}^{n} Q_i^F(x_i|x_1, \ldots, x_{i-1})$ where $Q_i^F(x_i|x_1, \ldots, x_{i-1})$ is defined as follows:*

1. *$Q_i^F(x_i|x_1, \ldots, x_{i-1}) = 0$ if $(x_1, \ldots, x_{i-1}, x_i)$ cannot be extended to a solution of F.*
2. *$Q_i^F(x_i|x_1, \ldots, x_{i-1}) = 1$ if $(x_1, \ldots, x_{i-1}, x_i)$ can be extended to a solution but $(x_1, \ldots, x_{i-1}, \overline{x_i})$ cannot be extended to a solution of F.*
3. *$Q_i^F(x_i|x_1, \ldots, x_{i-1}) = Q_i(x_i|x_1, \ldots, x_{i-1})$ if both $(x_1, \ldots, x_{i-1}, x_i)$ and $(x_1, \ldots, x_{i-1}, \overline{x_i})$ can be extended to a solution of F.*

Theorem 1. *The samples generated by SampleSearch(F,Q,O) are distributed according to the backtrack-free distribution Q^F.*

Proof. See [4] □

[2] We assume that Q satisfies the condition that for any partial assignment $(x_1, \ldots, x_i)$ that can be extended to a solution, $Q_i(x_i|x_1, \ldots, x_{i-1}) > 0$

Algorithm 4 $SampleSearch-SIR(F,Q,O,N,M)$

1: Generate N i.i.d. samples $\mathbf{A}=\{\mathbf{x}^1,\ldots,\mathbf{x}^N\}$ by executing SampleSearch(F,O,Q) N times.
2: Compute importance weights $\{w^1=\frac{1}{Q^F(\mathbf{x}^1)},\ldots,w^N=\frac{1}{Q^F(\mathbf{x}^N)}\}$ for each sample where Q^F is the backtrack-free distribution (see Definition 2). We show how to compute Q^F in subsection 3.4.
3: Normalize the importance weights using $\widetilde{w}^i=w^i/\sum_{j=1}^N w^j$
4: **Re-sampling Step:** Generate M i.i.d. samples $\{\mathbf{y}^1,\ldots,\mathbf{y}^M\}$ from $\mathbf{A}$ by sampling each sample $\mathbf{x}^i$ with probability $\widetilde{w}^i$.

Because SampleSearch generates independent and identically distributed samples from the backtrack-free distribution $Q^F(\mathbf{x})$, standard sampling theory [11] tells us that the distribution over the samples would converge to $Q^F(\mathbf{x})$ in the limit of infinite samples M and therefore the use of SampleSearch for solution sampling is problematic. In fact, one can construct cases in which the distance between the backtrack-free distribution and the uniform distribution is arbitrarily large. Next we show how this problem can be resolved using the Sampling Importance Resampling (SIR) principle.

3.2 Sampling Importance Resampling

The sampling importance re-sampling (SIR) [10, 13] algorithm aims at drawing random samples from a target distribution $\mathscr{P}(\mathbf{x})$ by using a proposal distribution $Q(\mathbf{x})$ such that $\mathscr{P}(\mathbf{x})>0\Rightarrow Q(\mathbf{x})>0$. First, a set of independent and identically distributed random samples $\mathbf{A}=(\mathbf{x}^1,\ldots,\mathbf{x}^N)$ are drawn from a proposal distribution $Q(\mathbf{x})$. Then, second, a possibly smaller number of samples $\mathbf{B}=(\mathbf{y}^1,\ldots,\mathbf{y}^M)$ are drawn from $\mathbf{A}$ with sample probabilities, proportional to the weights $w(\mathbf{x}^i)=\mathscr{P}(\mathbf{x}^i)/Q(\mathbf{x}^i)$ (this step is referred to as the *re-sampling step*). The samples from SIR will, as $N\rightarrow\infty$, consist of independent draws from $\mathscr{P}$ [10, 13].

3.3 SampleSearch-SIR

Assume for the purpose of discussion that the backtrack-free distribution can be computed (we will address the issue of computing the backtrack-free distribution in the next subsection). We can easily incorporate SampleSearch within the SIR framework as shown in Algorithm 4. Here, a set $\mathbf{A}=(\mathbf{x}^1,\ldots,\mathbf{x}^N)$ of solution samples ($N>M$) is first generated using SampleSearch. Then, M samples are drawn from $\mathbf{A}$ with sample probabilities proportional to $\frac{1}{Q^F(\mathbf{x}^i)}$.

Theorem 2 (Main Theorem). *As $N\rightarrow\infty$, the samples generated by $SampleSearch-SIR$ consist of independent draws from the uniform distribution over the solutions of F.*

Proof. From, SIR theory [10], we know that SampleSearch-SIR would generate i.i.d. samples from the uniform distribution $\mathscr{P}$ over the solutions of F as $N\rightarrow\infty$ if the following conditions are satisfied:

– **C.1** The N samples are generated i.i.d from some distribution Q^F (satisfied trivially from Theorem 1).

– **C.2** The M samples are generated in the re-sampling step by sampling each sample $\mathbf{x}$ with probability proportional to $\mathscr{P}(\mathbf{x})/Q^F(\mathbf{x})$

Because, $\mathscr{P}$ is a uniform distribution over the solutions, we have

$$w(\mathbf{x}) \propto \frac{\mathscr{P}(\mathbf{x})}{Q^F(\mathbf{x})} \propto \frac{1}{Q^F(\mathbf{x})} \tag{1}$$

Note that the M samples are generated in the re-sampling step of SampleSearch-SIR by sampling each generated solution with probability $\propto \frac{1}{Q^F(\mathbf{x})}$ and therefore from Equation 1 we can see that condition **C.2** is also satisfied by SampleSearch-SIR. □

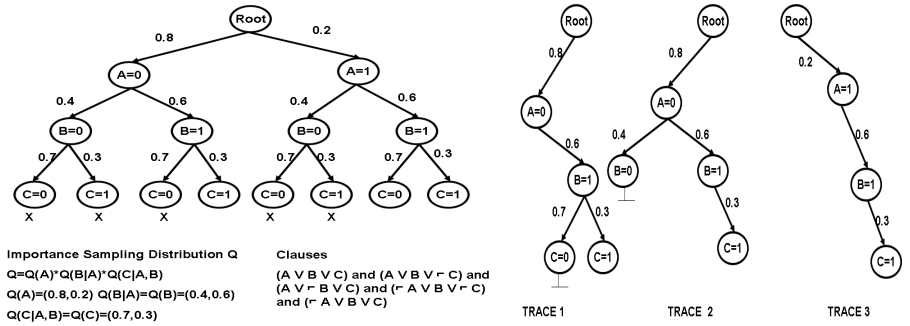
Theorem 2 is useful because it basically says that with increase in the number of initial samples N, the samples drawn in the re-sampling step would converge to the required uniform distribution over the solutions of F. To our knowledge, the three state-of-the-art schemes in literature for solution sampling (a) Pure SampleSearch, (b) Walksat [15] and (c) the recently proposed XorSample scheme [6] do not have such guarantees. In particular, SampleSearch converges to the wrong (backtrack-free) distribution while it is not known whether both XorSample and Walksat converge and if they do converge what distribution do they converge to. Finally, although the samples drawn from the scheme based on bucket elimination [2] do converge to the uniform distribution over the solutions, it is clearly not practical when the tree-width is large.

3.4 Computing $Q^F(\mathbf{x})$

As mentioned earlier, in this subsection we describe how to compute the backtrack-free distribution Q^F (see Definition 2) that is used to weigh each sample. From Definition 2, we can see that to compute the components $Q_i^F(x_i|x_1,\ldots,x_{i-1})$ of Q^F given a (solution) sample $\mathbf{x} = (x_1,\ldots,x_n)$, we have to determine whether $(x_1,\ldots,x_{i-1},\overline{x_i})$ can be extended to a solution. To that end, we can run any complete SAT solver (such as minisat [14]) on the formula $(F \wedge x_1 \wedge \ldots \wedge x_{i-1} \wedge \overline{x_i})$. If the solver proves that $(F \wedge x_1 \wedge \ldots \wedge x_{i-1} \wedge \overline{x_i})$ has a solution, we set $Q_i^F(x_i|x_1,\ldots,x_{i-1}) = Q_i(x_i|x_1,\ldots,x_{i-1})$. Otherwise, we set $Q_i^F(x_i|x_1,\ldots,x_{i-1}) = 1$. Finally, once $Q_i^F(x_i|x_1,\ldots,x_{i-1})$ is computed for all i, we can compute $Q^F(\mathbf{x})$ using $Q^F(\mathbf{x}) = \prod_{i=1}^n Q_i^F(x_i|x_1,\ldots,x_{i-1})$.

Approximating $Q^F(\mathbf{x})$ Since computing $Q^F(\mathbf{x})$ requires $O(n)$ invocations of a complete SAT solver per sample, as n gets larger, the SampleSearch-SIR scheme is likely to be too slow. Instead, we propose to approximate $Q^F(\mathbf{x})$ by utilizing the information gathered by SampleSearch itself while generating the samples.

Note that to compute $Q^F(\mathbf{x})$, we have to determine whether $(x_1,\ldots,x_{i-1},\overline{x_i})$ can be extended to a solution. While generating a sample $\mathbf{x}$, SampleSearch may have determined that $(x_1,\ldots,x_{i-1},\overline{x_i})$ cannot be extended to a solution for one or more i and thus, we can build the following approximation for $Q^F(\mathbf{x})$. If SampleSearch has itself determined that $(x_1,\ldots,x_{i-1},\overline{x_i})$ cannot be extended to a solution while generating $\mathbf{x}$, we set $A_i^F(x_i|x_1,\ldots,x_{i-1}) = 1$, otherwise we set $A_i^F(x_i|x_1,\ldots,x_{i-1}) = Q_i(x_i|x_1,\ldots,x_{i-1})$.



(a) (b)

Fig. 1. (a) An example Probability (Search) Tree for the shown Formula and importance sampling distribution Q. The leaf nodes marked with X are not solutions while the remaining leaf nodes are solutions (b) DPLL-Traces of SampleSearch. The grounded nodes were proved inconsistent

Finally, we compute $A^F(\mathbf{x}) = \prod_{i=1}^{n} A_i^F(x_i|x_1,\ldots,x_{i-1})$. However, if we use the approximation $A^F(\mathbf{x})$ to compute sample weights within SampleSearch-SIR, the resulting algorithm would not converge to the right distribution.

We can go one step further which will guarantee convergence. We can store (cache) each solution $(x_1,\ldots,x_n)$ and all partial assignments $(x_1,\ldots,\bar{x}_i)$ that were proved to be inconsistent during each independent execution of SampleSearch, which we refer to as search-traces. After executing SampleSearch N times, (i.e. once we have our required samples) we use the stored traces to compute $A_N^F(\mathbf{x})$ for each sample as follows (the approximation is indexed by N to denote dependence on N). For each partial sample $(x_1,\ldots,x_i)$ if $(x_1,\ldots,\bar{x}_i)$ was found to be inconsistent during any of the N executions of SampleSearch, we set $A_{N_i}^F(x_i|x_1,\ldots,x_{i-1}) = 1$, otherwise we set it to $Q_i(x_i|x_1,\ldots,x_{i-1})$. Finally, we compute $A_N^F(\mathbf{x}) = \prod_{i=1}^{n} A_{N_i}^F(x_i|x_1,\ldots,x_{i-1})$. Clearly, as N grows, more inconsistencies will be discovered by SampleSearch and eventually all inconsistencies will be discovered making $A_N^F(\mathbf{x})$ equal to $Q^F(\mathbf{x})$ i.e. $\lim_{N\to\infty} A_N^F(\mathbf{x}) = Q^F(\mathbf{x})$. Consequently, we can use $A_N^F(\mathbf{x})$ instead of $Q^F(\mathbf{x})$ in SampleSearch-SIR and still maintain correctness of Theorem 2 in the limit. Namely,

Proposition 1. *As $N \to \infty$, the samples generated by SampleSearch – SIR which uses A_N^F instead of Q^F to compute the sample weights would consist of independent draws from the uniform distribution over the solutions of F*[3].

Example 1. Figure 1(a) shows the probability tree associated with distribution Q. Each arc from a parent node to the child node in the probability tree is labeled with the probability of getting the child node given the assignment on the path from the root node to the parent node. The probability tree is also the complete search tree for the

[3] Although Theorem 2 and Proposition 1 are alike, it is known [11] that for a finite sample size the sampling error of SampleSearch-SIR which uses Q^F is likely to be less than SampleSearch-SIR which uses A_N^F. However, in general A_N^F requires a lot less time than Q^F.

formula shown. Assume that SampleSearch has generated three traces as shown in Figure 1(b). Note that in our example, we have 3 samples but only two distinct solutions (A=0,B=1,C=1) and (A=1,B=1,C=1). One can verify that $Q^F(\mathbf{x})$ of Traces 1, 2 and 3 is 0.8, 0.8 and 0.06 respectively. While $A_3^F(\mathbf{x})$ of Traces 1, 2 and 3 is 0.8, 0.8 and 0.036 respectively.

Finally, we summarize the time and space complexity of SampleSearch-SIR in which Q^F is replaced by the approximation A_N^F in the following proposition:

Proposition 2. *Given N samples (solutions) generated by the SampleSearch scheme, the time and space complexity of Steps (2) to (4) of Algorithm SampleSearch-SIR in which the approximation A_N^F is used instead of Q^F is $O(N*n+M*log(N))$ and $O(N*n)$ respectively where n is the number of variables.*

Proof. To compute and normalize the weights $1/A_N^F(\mathbf{x})$ of the N samples, we require $O(N*n)$ time (Steps (2) and (3)). Finally, to resample M samples from the distribution of the normalized weights over the N samples, we require an additional $O(M*log(N))$ [4] time yielding a time-complexity of $O(N*n+M*log(N))$. The space complexity of $O(N*n)$ is due to the space required by the approximation A_N^F to store the N samples. □

3.5 Extensions of basic SampleSearch-SIR

The integration of SampleSearch within the SIR framework allows using various improvements to the SIR framework presented in the statistics literature over the past decade. In this subsection, we consider two such improvements: (a) sampling without replacement and (b) Improved SIR.

Sampling without replacement The references to SIR in the literature use sampling with replacement in the re-sampling step i.e. the same solution sample may be drawn twice. An exception is Gelman et al. [3] who proposed sampling without replacement to avoid having too many replicates when Q^F is a poor approximation of $\mathscr{P}$. In this situation we may have a few very large weights and many small weights which causes the large weight samples to appear multiple times in the final set of samples. They showed that sampling without replacement yields a more desirable intermediate approximation somewhere between the starting (proposal) Q^F and target distribution $\mathscr{P}$.

Improved SIR Under certain restrictions [12] prove that the convergence of SIR is proportional to $O(1/N)$. To speed up this convergence to $O(1/N^2)$, they propose, what they call the Improved SIR framework. For our purposes, the only difference between Improved SIR and the basic SIR framework is the way in which each sample is weighed before the re-sampling step.

In case of sampling with replacement the basic SIR weighs each sample as $\frac{\mathscr{P}(\mathbf{x})}{Q^F(\mathbf{x})}$ while the Improved SIR framework weighs each sample using the following equation.

$$w(\mathbf{x}^i) \propto \frac{1}{S_{-i} * Q^F(\mathbf{x}^i)} \text{ where } S_{-i} = \sum_{j=1}^{M} \frac{1}{Q^F(\mathbf{x}^j)} - \frac{1}{Q^F(\mathbf{x}^i)} \tag{2}$$

[4] It is possible to use hashing to yield amortized complexity of $O(M)$. When binary search is used the complexity is $O(M*log(N))$

The first draw of Improved SIR without replacement is specified by the weights in Equation 2. For the kth draw, $k > 1$, the probability distribution of w is modified to:

$$w(\mathbf{x}^k) \propto \frac{\mathscr{P}(\mathbf{x}^k)}{Q(\mathbf{x}^k)(S_{(k)} - k\frac{\mathscr{P}(\mathbf{x}^k)}{Q(\mathbf{x}^k)})} \text{ where } S_{(k)} = \sum_{j=1}^{N} \frac{\mathscr{P}(\mathbf{x}^j)}{Q^F(\mathbf{x}^j)} - \sum_{j=1}^{k-1} \frac{\mathscr{P}(\mathbf{x}^j)}{Q^F(\mathbf{x}^j)} \quad (3)$$

When Improved SIR is used in conjunction with SampleSearch, we will refer to the resulting algorithm as $SampleSearch - ISIR$.

4 Experimental Evaluation

4.1 Competing Techniques

SampleSearch takes as input a distribution Q. The performance of sampling importance resampling algorithms is highly dependent on this choice of Q [10, 11]. It was shown that computing Q from the output of a generalized belief propagation scheme of Iterative Join graph propagation (IJGP) yields good empirical performance than other available choices [5]. Therefore, in our implementation we use the output of IJGP to compute Q. The complexity of IJGP is time and space exponential in a parameter i also called as i-bound. We tried i-bounds of 1, 2 and 3 and found that the results were not sensitive to the i-bound used and therefore we report results for i-bound of 3. The preprocessing time for computing the proposal distribution using IJGP $(i = 3)$ was negligible $(< 1s)$.

In a recent paper [4], we prove that Theorem 1 is correct even if we can replace the naive DPLL search in SampleSearch with any systematic SAT solver. Same holds true for Theorem 2 which is based on Theorem 1. Therefore, in order to increase the speed at which solution samples are generated, we chose to replace naive DPLL search in SampleSearch with the minisat [14] SAT solver (the winner of SAT competition 2006) in our implementation. Also, note that in our experiments we use the approximation A_N^F to compute the sample weights.

We experimented with three versions of SampleSearch (a) pure SampleSearch which is the same scheme as [5] denoted by SS, (b) SampleSearch with Improved SIR and with replacement denoted by SS-ISIR-wR , (c) SampleSearch with Improved SIR and without replacement denoted by SS-ISIR-woR. We also experimented with the WALKSAT [15] solution sampling scheme whose implementation is available from the first authors web-site [5]. All our experiments with WALKSAT are run with the *sabest* heuristic which was shown to perform better than other heuristics [15]. Also note that we use a resampling ratio $M/N = 0.1 = 10\%$ in all our experiments with the ISIR scheme (where N is the number of initial samples generated by SampleSearch and M is the number of re-sampled samples). The choice of the resampling ratio of 10% is arbitrary.

[5] We wanted to experiment with the XorSample scheme [6]. However the implementation is not publicly available

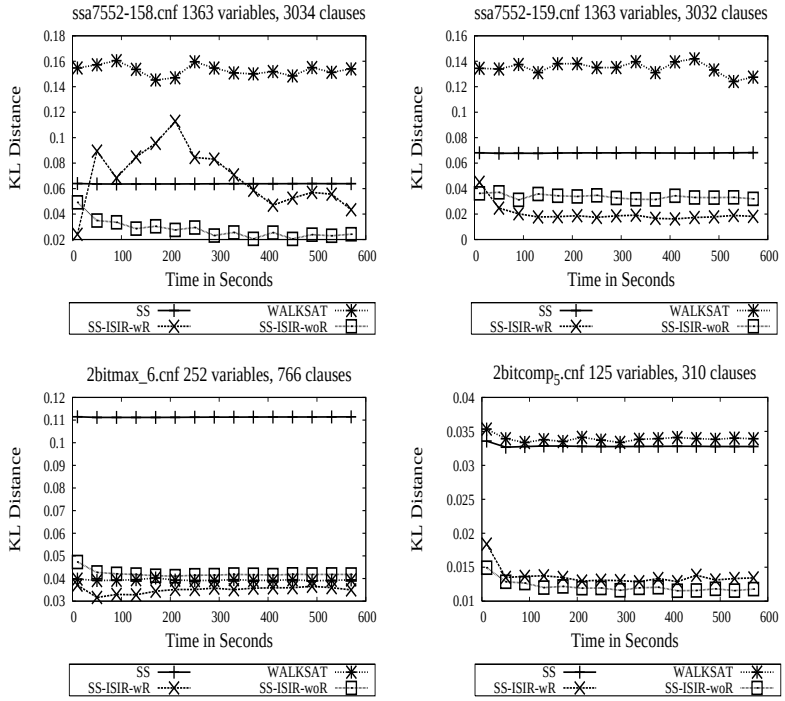


Fig. 2. Time vs KL distance for circuit instances

4.2 Evaluation Criteria

We first compute the exact marginal distribution for each propositional variable using $P_e(X_i = x_i) = |\mathbf{S}_{x_i}|/|\mathbf{S}|$ where $\mathbf{S}_{x_i}$ is the set of solutions that the assignment $X_i = x_i$ participates in and $\mathbf{S}$ is the set of all solutions. The number of solutions for the SAT problems were computed using RELSAT [9]. After running various sampling algorithms, we get a set of solution samples ϕ from which we compute the approximate marginal distribution: $P_a(X_i = x_i) = \phi(x_i)/|\phi|$ where $\phi(x_i)$ is the number of solutions in the set ϕ with X_i assigned the value x_i. We then compare the exact distribution with the approximate distribution using the Kullback-Leibler distance (KLD) $= P_e(x_i) ln(P_e(x_i)/P_a(x_i))$.

4.3 Results

Circuit Instances On the circuit instances $ssa7552-158$ and $ssa7552-159$ (see Figure 2), we find that SampleSearch-ISIR based algorithms have better performance after 100s than both SampleSearch and WALKSAT. On the $ssa7552-158$ instance, SampleSearch-ISIR without replacement is the best performing scheme while on the $ssa7552-159$ instance, SampleSearch-ISIR with replacement is the best performing scheme. On the $2bitmax\_6$ instance, SampleSearch-ISIR with replacement is slightly better than WALKSAT. However, SampleSearch-ISIR without replacement performs

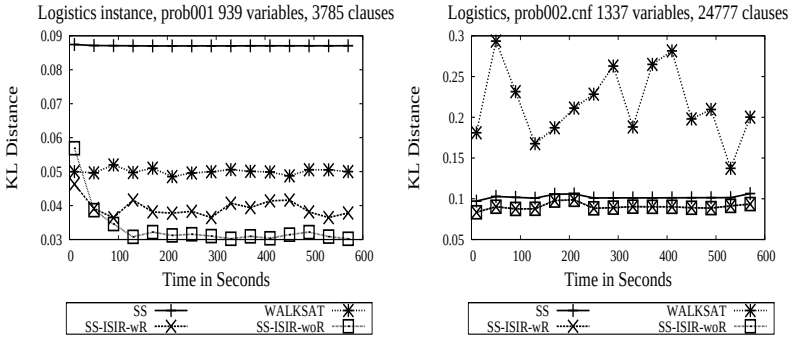


Fig. 3. Time vs KL distance for logistics instances

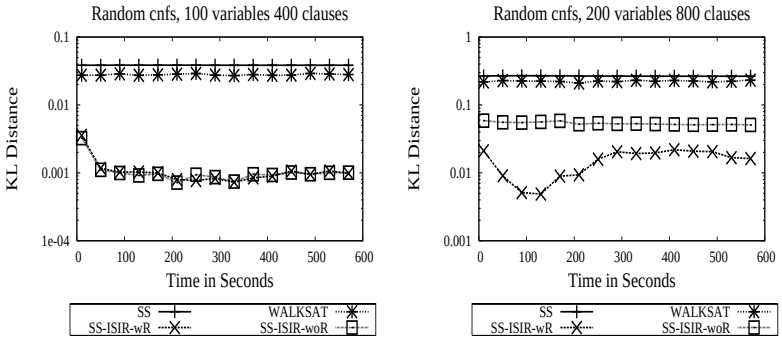


Fig. 4. Time vs KL distance for randomly generated 3-SAT instances

slightly worse than WALKSAT. Both SampleSearch-ISIR schemes clearly dominate the pure SampleSearch scheme. On the 2*bitcomp*_5 instance, we see that both the SampleSearch-ISIR schemes have significantly better performance than WALKSAT and pure SampleSearch.

Logistics instances On the logistics instance *prob*001.*cnf* (see Figure 3), we see that both the SampleSearch-ISIR schemes are better than WALKSAT and pure SampleSearch. WALKSAT performs better than pure SampleSearch scheme on this instance. SampleSearch-ISIR scheme without replacement is the best performing scheme. On the *prob*002.*cnf* instance, we notice that pure SampleSearch performs better than the WALKSAT scheme. Both SampleSearch-ISIR schemes are only slightly better than the pure SampleSearch scheme.

Randomly generated cnf instances We generated 100 random instances of 100-variable and 200-variable 3-cnf problems. Figure 4 shows the any-time performance of various schemes. Note that each point in Figure 4 is an average over 100 instances. On the 100-variable instances, both SampleSearch-ISIR schemes are significantly better in terms of the K-L distance than pure SampleSearch and WALKSAT. SampleSearch-ISIR without replacement shows similar performance to SampleSearch-ISIR with replacement.

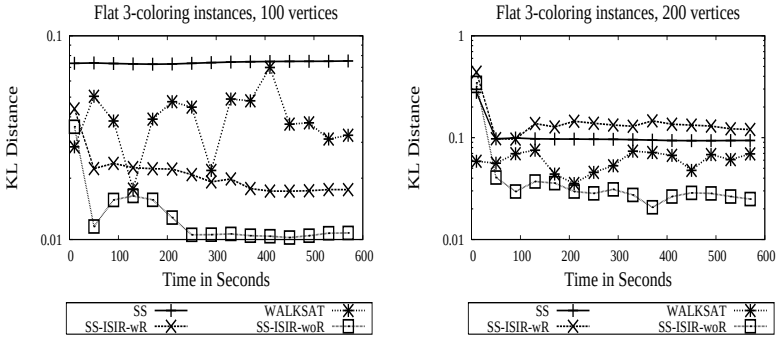


Fig. 5. Time vs KL distance for SAT encoded 3-coloring instances

On the 200-variable instances, we again see that both SampleSearch-ISIR scheme are significantly better than pure SampleSearch and WALKSAT. SampleSearch-ISIR with replacement is slightly better than SampleSearch-ISIR without replacement.

Sat Encoded Flat 3-coloring instances We generated 100 random instances of 100-vertex and 200-vertex Flat 3-coloring problem using Joseph Culberson's generator [6]. Figure 5 shows the effect of increasing time on the K-L distance on the graph coloring instances. Note that each point in Figure 5 is an average over 100 instances. On the 100-vertex 3-coloring instances, we see that both SampleSearch-ISIR schemes outperform WALKSAT after about 20s of run-time and they are always better than pure SampleSearch. SampleSearch-ISIR without replacement is the best performing scheme. On the 200-vertex 3-coloring instances, we see that WALKSAT is better than both pure SampleSearch and SampleSearch-ISIR with replacement. While SampleSearch-ISIR without replacement dominates WALKSAT after about 20s of run-time. We speculate that SampleSearch-ISIR without replacement is better than SampleSearch-ISIR with replacement because there are very few samples having large weights that dominate the set of re-sampled solutions.

Number of Samples generated by various methods Finally, in Table 1, we compare the time required by various algorithms to compute 100K samples. We can see that SampleSearch is superior to WALKSAT in terms of the time required to generate 100K samples. Note that the column titled "Resampling" in Table 1 shows the resampling time required by the Steps 2 through 4 of SampleSearch-SIR (see Algorithm 4). We observe that the resampling time is negligible compared with the time required to generate the initial samples using SampleSearch.

Effect of increasing the re-sampling rate The resampling rate (M/N) was set to 10% for all the results reported in Figures 2-5. Because, the resampling time is negligible, we can increase the number of samples generated by increasing the re-sampling rate. We therefore performed experiments to determine how increasing the re-sampling rate affects the K-L distance of the SampleSearch-ISIR scheme given constant time. For lack of space, we report results on only two circuit instances $ssa7552-158.cnf$ and

[6] Available at http://web.cs.ualberta.ca/∼joe/Coloring/Generators/generate.html

Table 1. Time required to generate 100K samples

Instance	Variables	clauses	SampleSearch Time	Resampling Time	WALKSAT Time
Circuit					
ssa-158	1363	3064	266.38	0.60	1879.70
ssa-159	1363	3032	257.20	0.60	1582.28
2bitcomp_5	125	310	23.82	0.20	67.82
2bitmax_6	252	766	49.20	0.20	87.44
Random					
100-var-cnf	100	400	17.93	0.10	438.60
200-var-cnf	200	800	55.80	0.20	226.86
3-coloring					
100-vertices	300	1117	550.66	0.40	3205.13
200-vertices	600	2456	2403.85	0.40	8620.69
Logistics					
prob001	939	3785	219.97	0.40	753.01
prob002	1337	24777	1326.26	0.50	7462.69

$ssa7552-159.cnf$. Figure 6 shows the effect of increasing the re-sampling rate from 10% to 90% on the K-L distance when each algorithm was given the same 100s of time. We observe that as the re-sampling rate is increased the K-L distance of SampleSearch-ISIR without replacement increases (linearly) and eventually becomes equal to the K-L distance of pure SampleSearch. This is because when sampling without replacement we do not draw samples which are already drawn and therefore with increase in the resampling rate the final set of samples in SampleSearch-ISIR would be the same as the initial set of samples. On the other hand, the K-L distance of SampleSearch-ISIR with replacement remains almost constant as the re-sampling rate is increased.

Therefore, in practice when Samplesearch generates N samples Samplesearch-ISIR with replacement can generate anywhere between 10% or 100% of N samples by demand, without any extra time cost or accuracy costs (if we use the with replacement version) because the time for resampling is negligible.

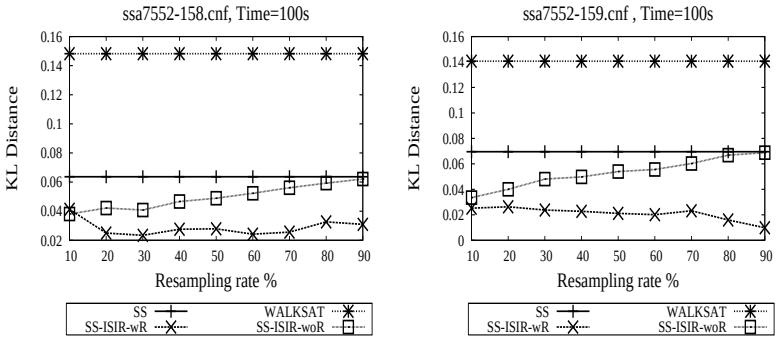


Fig. 6. Time vs KL distance for circuit instances

5 Conclusion and Summary

The paper presents a new algorithm for solving the random sampling task which is the task of generating random, uniformly distributed solutions from a satisfiability problem. The origin for this task is the use of satisfiability based methods in fields like functional verification, Distributed AI and probabilistic reasoning. Our algorithm builds on a recent negative result [4] where we show that the SampleSearch scheme [5] used to solve the random sampling task converges to the wrong distribution. We, then fix this problem by integrating the SampleSearch scheme with the sampling importance re-sampling (SIR) principle yielding the SampleSearch-SIR scheme which guarantees convergence to the uniform distribution over the solutions. To our knowledge, the current state-of-the-art schemes such as SampleSearch, WALKSAT [15] and XorSample [6] do not have convergence guarantees. Our empirical evaluation suggests that the SampleSearch-SIR scheme dominates both WALKSAT and the pure SampleSearch scheme in terms of K-L distance on most benchmark instances.

References

1. J Bergeron. *Writing Testbenches: Functional Verification of HDL Models*. Kluwer Academic Publishers, 2000.
2. Rina Dechter, Kalev Kask, Eyal Bin, and Roy Emek. Generating random solutions for constraint satisfaction problems. In *AAAI*, pages 15–21, 2002.
3. Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall Texts in Statistical Science. 2nd edition, June 1995.
4. Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. *AAAI-2007 (To appear)*.
5. Vibhav Gogate and Rina Dechter. A new algorithm for sampling csp solutions uniformly at random. *CP*, 2006.
6. Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Near-uniform sampling of combinatorial spaces using xor constraints. In *NIPS*. 2007.
7. M R Jerrum, L G Valiant, and V V Vazirani. Random generation of combinatorial structures from a uniform. *Theor. Comput. Sci.*, 43(2-3):169–188, 1986.
8. J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.
9. Jr. Roberto J. Bayardo and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI*, pages 157–162. AAAI Press / The MIT Press, 2000.
10. Donald B. Rubin. The calculation of posterior distributions by data augmentation. *Jornal of the American Statistical Association*, 82, 1987.
11. Reuven Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., New York, NY, USA, 1981.
12. Oivind Skare, Erik Bolviken, and Lars Holden. Improved sampling-importance resampling and reduced bias importance sampling. *Scandinavian Journal of Statistics*, 2003.
13. A. F. M. Smith and A. E. Gelfand. Bayesian statistics without tears: A sampling–resampling perspective. 46(2):84–88, May 1992.
14. Niklas Sorensson and Niklas Een. Minisat v1.13-a sat solver with conflict-clause minimization. In *SAT*, 2005.
15. Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: Exploiting random walk strategies. In *AAAI*, 2004.