Approximate Solution Sampling (and Counting) on AND/OR search space

Vibhav Gogate and Rina Dechter

Donald Bren School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA
{vgogate,dechter}@ics.uci.edu

Abstract. In this paper, we describe a new algorithm that approximately solves the problem of sampling solutions from a uniform distribution over the solutions of a constraint network. Our new algorithm improves upon the Sampling/Importance Resampling (SIR) component of our previous scheme of SampleSearch-SIR by taking advantage of the decomposition implied by the network's *AND/OR search space*. We describe how our new scheme can be modified to approximately count and lower bound the number of solutions of a constraint network. We demonstrate both theoretically and empirically that on networks which have a favorable decomposition, our new algorithm yields far better performance than competing approaches.

1 Introduction

The paper introduces a new method for approximately solving the $\S csp$ problem which is the problem of generating solutions from a uniform distribution over the solutions of a constraint network. As pointed out in earlier work [17, 3, 4], the $\S csp$ problem has tremendous applications in fields such as verification and probabilistic reasoning. Our main contribution is in improving the Sampling/Importance Resampling component of our previous scheme of SampleSearch-SIR [7] by exploiting problem decomposition via AND/OR search spaces for graphical models [2].

SampleSearch-SIR [7] approximately solves the $\S csp$ problem as follows. First, SampleSearch [4] is used to generate an initial set **A** of samples from the backtrack-free distribution Q^F. Second, each sample is weighed by the reciprocal of the probability of it being generated from Q^F. Third, a distribution $\mathscr{M}$ is formed over the initial set **A** by normalizing the weights and finally a smaller set of samples **B** is drawn from $\mathscr{M}$. The final step is often called as the *resampling* step. It was shown by [12] that as the size of the initial set **A** increases the distribution over the samples generated in the resampling step (i.e. **B**) will converge to the uniform distribution over the solutions.

In our new scheme, called *AO-SIR*, the first step of generating the initial set of samples from Q^F (using SampleSearch) remains the same. What changes is that the initial set **A** of samples is stored on an AND/OR structure. The AND/OR representation defines a new distribution over the initial set of samples, from which samples are drawn in the resampling step. We argue and show that the AND/OR structure imposed on the samples will lead to a more accurate sampling scheme than the earlier one we proposed in [7]. Intuitively, SampleSearch-SIR is a method for learning a multi-variate distribution over the constraint network (or any graphical model) from an initial set $\mathbf{A}$ of weighted samples. SampleSearch-SIR assumes no independencies among the variables in the constraint network. The AND/OR representation of the samples on the other hand captures some of the inherent conditional independencies, yielding a more compact sample representation that captures a larger set of virtual samples. Therefore, AO-SIR is likely to be more accurate and have better convergence properties.

Subsequently, we derive a new unbiased estimator using the samples stored on the AND/OR structure and show how it can be combined with SampleSearch to approximately count and lower bound the number of solutions of a constraint network, improving over our previous solution counter [5].

To evaluate whether exploiting decomposition via the AND/OR structure yields improved performance in practice, we performed an empirical evaluation on a wide range of satisfiability benchmarks . We found that on most problems our new schemes that operate in the AND/OR space are more accurate than SampleSearch-SIR for solution sampling while in the case of solution counting, we were able to improve upon the lower bounds reported in [5, 8] in most cases.

The rest of the paper is organized as follows. In section 2, we present preliminaries and previous work. In section 3, we present the AO-SIR algorithm and describe its properties. In section 4, we define a new unbiased estimator in AND/OR space. Empirical results are presented in section 5 and we end with a brief summary in section 6.

2 Preliminaries

Definition 1 (constraint network, counting and sampling). *A constraint network (CN) is defined by a 3-tuple, $\mathscr{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of variables associated with a set of discrete-valued domains, $\mathbf{D} = \{\mathbf{D_1}, \ldots, \mathbf{D_n}\}$, and $\mathbf{C} = \{C_1, \ldots, C_r\}$ is a set of constraints. Each constraint C_i is a relation $\mathbf{R_{S_i}}$ defined on a subset of variables $\mathbf{S_i} \subseteq \mathbf{X}$. The relation denotes all compatible tuples of the cartesian product of the domains of $\mathbf{S_i}$. A solution is an assignment of values to all variables $\mathbf{x} = (X_1 = x_1, \ldots, X_n = x_n)$, $x_i \in \mathbf{D}_i$, such that $\mathbf{x}$ belongs to the natural join of all constraints i.e. $\mathbf{x} \in \mathbf{R_{S_1}} \bowtie \ldots \bowtie \mathbf{R_{S_r}}$. The constraint satisfaction problem (CSP) is to determine if a constraint network has a solution, and if so, to find one. The* **primal graph** *(also called the constraint graph) of a constraint network is an undirected graph that has variables as its vertices and an edge connects any two variables involved in a constraint.*
The **solution counting problem** *#csp is the problem of counting the number of solutions of a constraint network. The* **solution sampling problem** *§csp is the problem of sampling solutions from a uniform distribution over the solutions of a constraint network.*

2.1 AND/OR search spaces for general inference

The AND/OR search space [2] is a generic inference scheme that can be used to exactly solve various combinatorial problems in graphical models. We can solve most combinatorial problems by search, by systematically enumerating all the possible combinations. In the simplest case, this process defines an OR search tree, whose nodes represent partial variable assignments. This search space does not capture the structure of the underlying

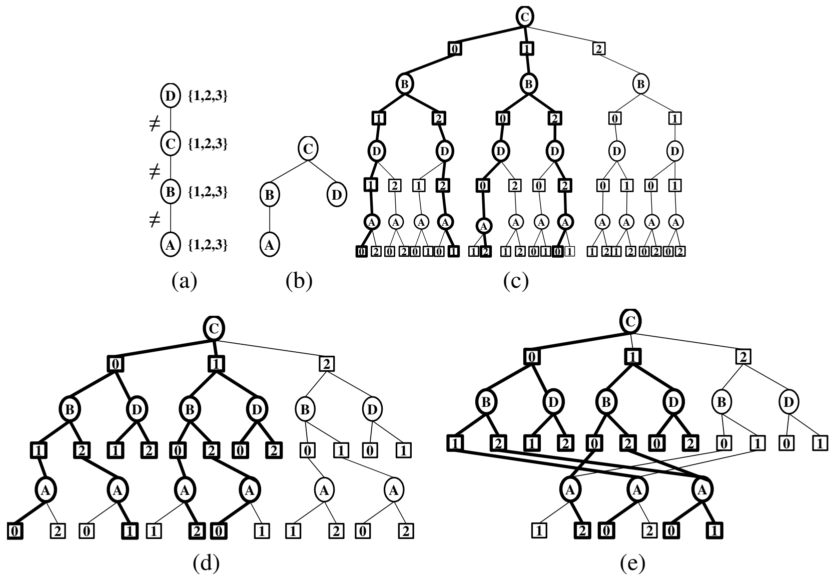

Fig. 1. (a) A 3-coloring problem, (b) Pseudo-tree (c) AND/OR tree whose pseudo-tree is a chain (d) AND/OR tree (e) AND/OR search graph

graphical model. To remedy this problem, [2] introduced the notion of AND/OR search spaces for graphical models. Given a constraint network $\mathscr{R} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$, its AND/OR search space is driven by a pseudo tree defined below.

Definition 2 (Pseudo Tree). *Given an undirected graph $G = (V, E)$, a directed rooted tree $T = (V, E)$ defined on all its nodes is called pseudo tree if any arc of G which is not included in E is a back-arc, namely it connects a node to an ancestor in T. The pseudo-tree of a constraint network is the pseudo-tree of its constraint graph.*

Definition 3 (Labeled AND/OR tree). *Given a constraint network $\mathscr{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ and a pseudo tree T, the AND/OR search tree S_{AOT}, has alternating levels of AND and OR nodes. The root of S_{AOT} is an OR node labeled by the root of T. The children of an OR node X_i are AND nodes labeled with assignment $X_i = x_i$ that are consistent with the assignment $(X_1 = x_1, \ldots, X_{i-1} = x_{i-1})$ along the path from the root. The children of an AND node $X_i = x_i$ are OR nodes labeled with the children of variable X_i in T. Each OR arc, emanating from an OR node to an AND node is associated with a* **labeling function** *which can be derived from the constraints.[2]. Each OR node and AND node is also associated with a* **value** *that is recursively used for computing the quantity of interest. An* **OR tree** *is an AND/OR tree whose pseudo-tree is a chain.*

Definition 4 (Solution Subtree). *A solution subtree of a labeled AND/OR tree contains the root node. For every OR node it contains one of its child nodes and for each of its AND nodes it contains all its child nodes.*

Semantically, the OR states represent alternative assignments, whereas the AND states represent problem decomposition into independent subproblems conditioned on some value assignments, all of which need be solved. When the pseudo-tree is a chain, the AND/OR search tree coincides with the regular OR search tree. We can count the number of solutions of a constraint network by labeling each consistent arc by 1 and each inconsistent arc by 0. In this case, the number of solutions is equal to the number of solution subtrees. For more details see [2].

AND/OR search graphs An AND/OR tree may contain nodes that root identical subtrees (i.e. their root nodes values are identical); called as unifiable nodes. When unifiable nodes are merged, the tree becomes a graph and its size becomes smaller. Some unifiable nodes can be identified using contexts detectable from the constraint graph as defined below.

Definition 5 (AND Context). *Given an AND/OR search tree S_{AOT} relative to a pseudo-tree T, the context of any AND node $\langle X_i, x_i\rangle \in S_{AOT}$, denoted by $context(X_i)$, is defined as the set of ancestors of X_i in T, that are connected to X_i and descendants of X_i.*

The context minimal AND/OR graph is obtained by merging all the context unifiable AND nodes.

Example 1. Figure 1(a) shows a constraint network for a 3-coloring problem over 4 variables. Figure 1(d) shows the AND/OR-search tree for the constraint network based on the Pseudo-tree in Figure 1(b). Figure 1(e) shows the AND/OR graph for the AND/OR search tree of Figure 1(d) which is formed by merging all context unifiable nodes. Notice that its size of the AND/OR graph is smaller than the AND/OR tree. Figure 1(c) shows the AND/OR tree for a chain pseudo-tree which is equivalent to the traditional OR search space.

2.2 Exact Solution to the §*csp* problem

In the following, we describe an approach due to Dechter et al. [3] to exactly solve the §*csp* problem. We first express the uniform distribution $\mathscr{P}(\mathbf{x})$ over the solutions in a product factored form: $\mathscr{P}(\mathbf{x} = (x_1, \ldots, x_n)) = \prod_{i=1}^{n} P_i(x_i|x_1, \ldots, x_{i-1})$ (this follows from standard probability theory). Here, the probability $P_i(X_i = x_i|x_1, \ldots, x_{i-1})$ can be obtained by computing the ratio between the number of solutions that the partial assignment $(x_1, \ldots, x_i)$ participates in and the number of solutions that $(x_1, \ldots, x_{i-1})$ participates in. Then, we use an ordered Monte Carlo (MC) sampler (also called logic sampling [11]) to sample along the ordering $O = \langle X_1, \ldots, X_n\rangle$ as described in Algorithm 1. In Example 2, we demonstrate how the ordered Monte Carlo sampler operates. Note that the ordered Monte Carlo sampler is a general technique which can be used to sample from any distribution expressed in a product form.

Example 2. Figure 2(a) shows a complete OR search tree (the AND/OR tree whose pseudo-tree is a chain) for the 3-coloring problem in Figure 1(a). Each arc from a parent $X_{i-1} = x_{i-1}$ to a child $X_i = x_i$ in the search tree is labeled with the ratio of the number of solutions below the child and the number of solutions below the parent which corresponds to the probability $P_i(X_i = x_i|X_1 = x_1, \ldots, X_{i-1} = x_{i-1})$. Given random numbers

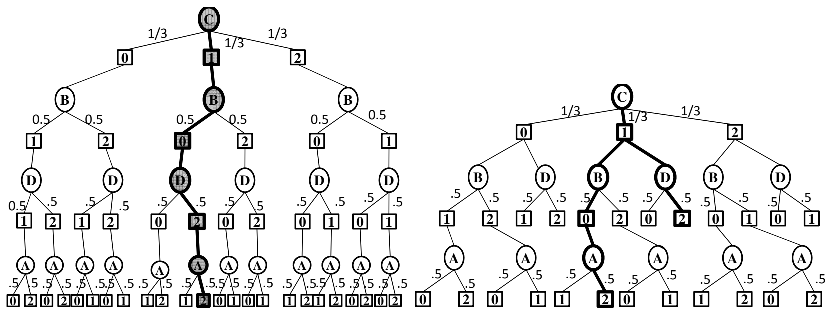

Fig. 2. Uniform distribution over the solutions of a CSP expressed in a product factored form on (a) an OR tree (b) an AND/OR tree

$\{0.6, 0.3, 0.7, 0.9\}$, for example, the solution highlighted in bold in Figure 2(a) will be generated by the ordered Monte Carlo sampler.

All algorithms described in this paper are devoted to finding an approximation to this exact probability $P_i(X_i = x_i | x_1, \ldots, x_{i-1})$ at each branch of the search tree.

2.3 SampleSearch-SIR to solve the $\S csp$ problem

Because constructing $\mathscr{P}(\mathbf{x})$ can be quite hard [3], in [7] we proposed to use Sampling Importance Resampling (SIR) [12] in conjunction with the SampleSearch scheme [4] to approximate it. This scheme operates as follows. First, given a proposal distribution Q, it uses SampleSearch to draw random solution samples $\mathbf{A} = (\mathbf{x}^1, \ldots, \mathbf{x}^N)$ from the backtrack-free distribution Q^F of Q. 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 $w(\mathbf{x}^i) = 1/Q^F(\mathbf{x}^i)$ (this step is referred to as the *re-sampling step*). For $N = 1$, the distribution of solutions is same as Q^F. For a finite N, the distribution of solutions is somewhere between Q^F and $\mathscr{P}$ improving as N increases and equals $\mathscr{P}$ as $N \to \infty$.

3 Sampling Importance Resampling on AND/OR search spaces

In this section, we describe the main contribution of this paper which lies in defining a novel Sampling/Importance Resampling (SIR) scheme on AND/OR search spaces. Before providing a formal description, we describe the main intuition involved in moving

Algorithm 1 Generate sample ($\mathscr{P} = \prod_{i=1}^{n} P_i(X_i | X_1, \ldots, X_n)$)

```
1: $\mathbf{x} = \phi$
2: for $i$= 1 to $n$ do
3:     Generate a random number $p$ between 0 and 1.
4:     Divide the number line between 0 and 1 into $|D_i|$ intervals such that each interval belonging
       to $X_i = x_i$ is proportional to $P_i(X_i = x_i | \mathbf{x})$.
5:     Let $X_i = x_i$ be the interval in which $p$ lies. $\mathbf{x} = \mathbf{x} \cup x_i$.
6: end for
```

to AND/OR space in the following example. Note that in the following, we will abuse notation and refer to SampleSearch-SIR as SIR.

Example 3. The bold edges and nodes in Figure 1 (c) show four solution samples arranged on an OR tree (an AND/OR tree whose pseudo-tree is a chain). The bold edges and nodes in Figures 1 (d) and 1 (e) show the same four samples arranged on an AND/OR tree and graph respectively. Visually, we can see that the same samples achieve larger coverage in the AND/OR space than in the OR space. One can verify that the 4 solution samples in OR-space correspond to 8 and 12 solution samples (solution sub-trees) respectively on the AND/OR tree and AND/OR graph. Thus, the AND/OR representation yields a larger initial set of (virtual) samples. It includes for example the assignment $(C=0, B=2, D=1, A=0)$ which is not represented in the initial set of SIR which operates in the OR space. From SIR theory [12], we know that the accuracy of SIR increases with the cardinality of $\mathbf{A}$. Therefore, we expect that the solutions resampled from the AND/OR tree and graph will be more accurate than conventional resampling performed over the OR tree.

An alternative intuition is provided in the next example.

Example 4. Figure 2(a) and 2(b) present the uniform distribution over the solutions expressed on an OR tree and AND/OR tree respectively. As pointed out earlier, any approximate sampling algorithm can be understood as a method that first approximates the probability labels on the arcs and then samples from this approximate structure. The conventional SIR algorithm approximates the probability labels on the OR-tree (proof appears later) using the initial set of samples while our new AO-SIR algorithm approximates probability labels on the AND/OR tree. We expect AO-SIR which approximates probability labels on the AND/OR tree to converge faster to the correct distribution than the conventional SIR scheme which operates on the OR tree, because the AND/OR tree is more compact than the OR tree and expresses more virtual samples.

For lack of space, we will describe the AO-SIR scheme on an AND/OR tree noting that it can be generalized to AND/OR graphs. In AO-SIR, the process of generating the initial set of samples from Q^F remains the same. What changes is the way in which we (a) store samples, (b) define the distribution over the initial set of samples and (c) perform the resampling step. We explain each of these modifications below. We first define the notion of an AND/OR sample tree (and graph) which can be used to store the initial set of samples. The arc labels on this AND/OR sample tree (or graph) are set to account for the weights.

Definition 6 (Arc Labeled AND/OR sample tree and graph). *Given (1) a constraint network $\mathscr{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, (2) a pseudo-tree $T(V,E)$, (3) the backtrack-free distribution $Q^F = \prod_{i=1}^{n} Q_i^F(X_i|\mathbf{Anc}(X_i))$ such that $\mathbf{Anc}(\mathbf{X_i})$ is a subset of all ancestors of X_i in T, (4) a sequence of samples $\mathbf{S}$ (assignments) generated from Q^F,* **an arc-labeled AND/OR sample tree** *S_{AOT} is its corresponding AND/OR tree (see definition 3) from which all assignments not in $\mathbf{S}$ are removed. The arc-label from an OR node X_i to an AND node $X_i = x_i$ in S_{AOT} is a pair $\langle \#, w \rangle$ where:*

– *#: the frequency of the arc is the number of times $(X_i = x_i, \mathbf{anc}(X_i))$ is seen in $\mathbf{S}$.*

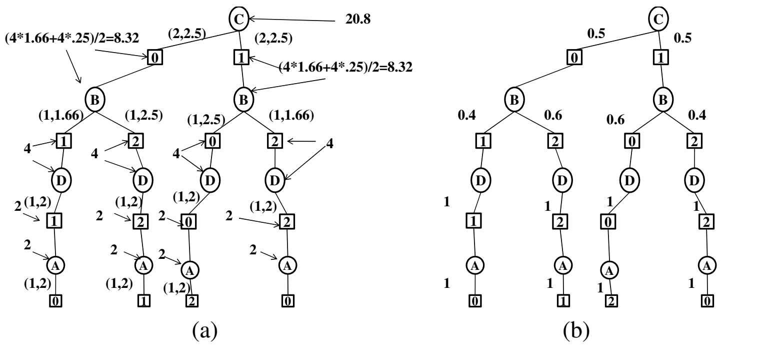
Fig. 3. (a) Computation of Value on an AND/OR sample tree (b) AND/OR sample probability tree

– $w = \frac{1}{Q_i^F(X_i = x_i | \mathbf{anc}(X_i))}$ *is called the weight of the arc.* $\mathbf{anc}(X_i)$ *is the assignment of values to all variables from the node* X_i *to the root node of* S_{AO}.

An **arc-labeled AND/OR sample graph** S_{AOG} *is constructed from an AND/OR sample tree* S_{AOT} *by merging all context unifiable nodes (we can show that the context based unification is sound in this case as well). In case of AND/OR sample graph, we require that the proposal distribution has the form:* $Q^F = \prod_{i=1}^{n} Q_i^F(X_i | \mathbf{context}(X_i))$.

Definition 6 describes how to store the samples generated from Q^F on an AND/OR tree (or graph). The next step is to compute a distribution over the stored samples from which we resample. Our aim is to approximate the exact distribution expressed on a complete AND/OR tree by approximating the labels on each arc from an OR node n to an AND node n' with a probability value that is (approximately) proportional to the number of solutions of the subtree rooted at n' [3]. SIR approximates these probability labels by a quantity that is proportional to its current estimate of the respective solution counts. AO-SIR will do the same with respect to the AND/OR sample tree. To that end, we define the notion of value of the node $v(n)$ which can be semantically understood as providing an unbiased estimate of the solution counts of the subtree rooted at n.

Definition 7 (Value of a node). *The value of a node in a arc-labeled AND/OR sample tree (see Definition 6) is defined recursively as follows. The value of leaf AND nodes is "1" and the value of leaf OR nodes is "0". Let* $\mathbf{C(n)}$ *denote the child nodes of n and* $v(n)$ *denotes the value of node n. If n is an AND node then:* $v(n) = \prod_{n' \in \mathbf{C(n)}} v(n')$ *and if n is a OR node then*

$$v(n) = \frac{\sum_{n' \in \mathbf{C(n)}} (\#(n,n') w(n,n') v(n'))}{\sum_{n' \in \mathbf{C(n)}} \#(n,n')} \tag{1}$$

The **weight contribution** *of an AND node* n' *is equal to* $\#(n,n')w(n,n')v(n')$.

In the following example, we show how the value of each node on a AND/OR sample tree and graph can be computed.

Example 5. Figure 3(a) demonstrates how to recursively compute the value of a node on an AND/OR sample tree. The value of all leaf AND nodes is set to 1. The value of an internal AND node is obtained by product of the values of its children. The value of an OR node is obtained by dividing the sum of the *weight contribution* of the child nodes by the sum of the frequencies on the incoming arcs. The weight contribution of a child AND node is equal to the product of the value of the node with the frequency and the weight of the arc.

In the following lemma, we provide the semantics of computing the value of a node on an AND/OR sample tree.

Lemma 1. *The value of a node n is an unbiased estimate of the number of solutions of the subtree rooted at n.*

Proof. Define $F(\mathbf{x}) = 1$ iff the assignment $\mathbf{x}$ is a solution and 0 otherwise. Given the backtrack-free distribution $Q^F(\mathbf{x}) = \prod_{i=1}^{n} Q_i^F(X_i|\mathbf{Anc}(X_i))$ which is defined only on the solutions of the constraint network, we can express the counting problem as follows:

$$\#csp = \sum_{X_1,\ldots,X_n} F(\mathbf{X}) = \sum_{X_1,\ldots,X_n} F(\mathbf{X}) \frac{Q^F(\mathbf{X})}{Q^F(\mathbf{X})} \tag{2}$$

From AND/OR theory [2], we know that we can decompose F as a product of several smaller functions based on the pseudo-tree T. Namely, $F(\mathbf{x}) = \prod_{i=1}^{n} F_i(X_i, \mathbf{Anc}(X_i))$. Therefore, we can express Equation 2 as follows:

$$\#csp = \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} \frac{F_i(X_i, \mathbf{Anc}(X_i))}{Q_i^F(X_i|\mathbf{Anc}(X_i))} Q_i^F(X_i|\mathbf{Anc}(X_i)) \tag{3}$$

By commutativity property of summation and product, we have:

$$\#csp = \sum_{X_1} \frac{F_1(X_1)}{Q_1^F(X_1)} Q_1^F(X_1) \times \ldots \times \sum_{X_n} \frac{F_n(X_n, \mathbf{Anc}(X_n))}{Q_n^F(X_n|\mathbf{Anc}(X_n))} Q_n^F(X_n|\mathbf{Anc}(X_n)) \tag{4}$$

Note that conditional expectation of a random variable A given B taken with respect to a distribution P is defined as: $\mathbf{E}_P[A|B] = \sum_A A \times P(A|B)$ while expectation is defined as: $\mathbf{E}_P[A] = \sum_A A \times P(A)$. By using the definition of expectation and conditional expectation, Equation 4 can be written as:

$$\#csp = \mathbf{E}\left[\frac{F_1(X_1)}{Q_1^F(X_1)} \times \ldots \times \mathbf{E}\left[\frac{F_n(X_n, \mathbf{Anc}(X_n))}{Q_n^F(X_n|\mathbf{Anc}(X_n))} |\mathbf{Anc}(X_n)\right]\right] \tag{5}$$

Let $\mathbf{Chi}(X_i)$ be the set of children of X_i in the Pseudo-tree. In general, the conditional expectations at a node X_i in Equation 5 given an assignment $\mathbf{anc}(x_i)$ is given by:

$$\mathscr{V}(X_i|\mathbf{anc}(X_i)) = \mathbf{E}\left[\frac{F_i(X_i, \mathbf{anc}(X_i))}{Q_i^F(X_i|\mathbf{anc}(X_i))} \prod_{X_j \in \mathbf{Chi}(X_i)} \mathscr{V}(X_j|\mathbf{anc}(X_j))|\mathbf{Anc}(X_i)\right] \tag{6}$$

Note that by definition, $\mathscr{V}(X_i|\mathbf{anc}(X_i))$ is equal to the number of solutions that the assignment $\mathbf{anc}(X_i)$ participates in. Because all samples generated from $Q_i^F(X_i|Anc(X_i))$ are solutions, we have $F_i(X_i, \mathbf{Anc}(X_i)) = 1$ for all the generated samples. Therefore, we can ignore this quantity.

Given samples generated from Q^F, we can estimate $\mathscr{V}(X_i|\mathbf{anc}(X_i))$ as follows. Let k be the domain size of X_i and let $\#(X_i = x_i, \mathbf{anc}(X_i))$ be the number of times the assignment $(X_i = x_i, \mathbf{anc}(X_i))$ is sampled and let $w(X_i = x_i, \mathbf{anc}(X_i)) = 1/Q^F(X_i = x_i|\mathbf{anc}(X_i))$ be the weight of this assignment. From standard sampling theory [13], we have the following estimate $v(X_i|\mathbf{anc}(X_i))$ of $\mathscr{V}(X_i|\mathbf{anc}(X_i))$:

$$v(X_i|\mathbf{anc}(X_i)) = \frac{\sum_{i=1}^{k} \#(X_i = x_i^k, \mathbf{anc}(X_i)) w(X_i = x_i^k, \mathbf{anc}(X_i)) \prod_{X_j \in \mathbf{Chi}(X_i)} v(X_j|X_i = x_i^k, \mathbf{anc}(X_i))}{\sum_{i=1}^{k} \#(X_i = x_i^k, \mathbf{anc}(X_i))} \tag{7}$$

Because in the expectation, the value of an OR node, $v(X_i|\mathbf{anc}(X_i))$ is equal to the number of solutions that the assignment $\mathbf{anc}(X_i)$ participates in, we have an unbiased estimator. □

From Lemma 1, we see that the value $v(n)$ of a node in the AND/OR sample tree stores the average weight (which provides an unbiased estimate of the solution counts) of the subproblem rooted at n, subject to the current variable instantiation along the path from the root to n. As described earlier, the distribution over the initial set of samples in SIR is obtained by simply normalizing the weights. We can replicate the same process on a AND/OR tree (or graph) by just normalizing the *weight contributions* at each OR node to yield a probability distribution over the AND/OR tree. We now define the notion of AND/OR sample probability trees (and graphs) which correspond to a graphical representation of the probability distribution over the initial set of samples.

Definition 8 (AND/OR sample probability tree and graph). *Given an arc labeled AND/OR sample tree (or graph) S_{AOT}, an AND/OR sample probability tree (or graph) has the same structure as S_{AOT} and the arc-label $p(n,n')$ from an OR-node n to an AND node n' is defined as follows. Let $\mathbf{C(n)}$ denote the child nodes and $v(n)$ denote the value of node n.*

$$p(n,n') = \frac{v(n')w(n,n')\#(n,n')}{\sum_{n'' \in \mathbf{C(n)}} (\#(n,n'')w(n,n'')v(n''))} \tag{8}$$

Example 6. Figure 3(b) shows an AND/OR sample probability tree constructed from the arc-labeled AND/OR sample tree of Figure 3(a).

The AND/OR sample probability tree represents the sampling distribution used for resampling. We can generate samples from it in the usual way using the ordered Monte Carlo sampler described in section 2.2.

We now have the required definitions to formally present algorithm AO-SIR (see Algorithm 2). Here, we first generate samples in the usual way from Q^F. We then store these samples on an arc labeled AND/OR sample tree or a graph. The AND/OR sample tree or graph is then converted to a AND/OR sample probability tree (or graph) (Steps $3-9$). Finally, the required M solution samples are generated by sampling from the AND/OR sample probability tree (or graph). We can prove that:

Theorem 1. *As $N \rightarrow \infty$, the samples generated by AO-SIR will consist of independent draws from the uniform distribution over the solutions.*

Proof. Below we provide a sketch of the proof. Let $M(X_1,\ldots,X_n) = \prod_{i=1}^{n} M_i(X_i|\mathbf{Anc}(X_i))$ be the sampling distribution defined by the AND/OR sample probability tree.

Algorithm 2 $AO-SIR(\mathscr{R},Q^F,N,M)$

```
1:  Generate $N$ i.i.d. samples $\mathbf{A}=\{\mathbf{x}^1,\ldots,\mathbf{x}^N\}$ from $Q^F$
2:  Store the $N$ solution samples on an AND/OR sample tree $S_{AOT}$ or a graph and label it using
    definition 6.
3:  FOR all leaf nodes $i$ of $S_{AOT}$ do
4:    IF And-node v(i)= 1 ELSE v(i)=0
5:  For every node $n$ from leaves to the root do
6:    Let $C(n)$ denote the child nodes of node $n$
7:    IF $n=\langle X,x\rangle$ is a AND node, then $v(n)=\prod_{n'\in C(n)} v(n')$
8:    ELSE if $n=X$ is a OR node then (from Definition 7)
          $$v(n)=\frac{\sum_{n'\in C(n)}(\#(n,n')w(n,n')v(n'))}{\sum_{n'\in C(n)}\#(n,n')}.$$
9:  Construct the AND/OR sample probability tree $P_{AOT}$ from $S_{AOT}$ by labeling each arc from an
    OR node to an AND node in $P_{AOT}$ as:
          $$p(n,n')=\frac{v(n')w(n,n')\#(n,n')}{\sum_{n''\in\mathbf{C(n)}}(\#(n,n'')w(n,n'')v(n''))}$$
10: Return $M$ solution samples generated from $P_{AOT}$ by using the ordered Monte Carlo sampler.
```

Let $\mathscr{P}(X_1,\ldots,X_n)=\prod_{i=1}^{n}P_i(X_i|\mathbf{Anc}(X_i))$ be the uniform distribution over the solutions. Note that given an assignment to the ancestors of X_i, it was proved in [3] that $P(X_i|\mathbf{Anc}(X_i)=\mathbf{anc}(X_i))$ is proportional to the number of solutions that $\mathbf{anc}(X_i)$ participates in. Using the notation from the proof of Lemma 1, we have:

$$P_i(X_i|\mathbf{anc}(X_i))\propto\mathscr{V}(X_i|\mathbf{anc}(X_i)) \tag{9}$$

From the definition of AND/OR sample probability tree, $M_i(X_i|\mathbf{Anc}(X_i)=\mathbf{anc}(X_i))$ is proportional to the value of the corresponding OR node. Therefore,:

$$M_i(X_i|\mathbf{anc}(X_i))\propto v(X_i|\mathbf{anc}(X_i)) \tag{10}$$

From sampling theory [13], we know that the unbiased estimate $v(X_i|\mathbf{Anc}(X_i)=\mathbf{anc}(X_i))$ would converge to $\mathscr{V}(X_i|\mathbf{Anc}(X_i)=\mathbf{anc}(X_i))$ as $N\rightarrow\infty$. Therefore, the quantities proportional to the two estimate v and the exact value $\mathscr{V}$ will be equal in the limit. That is,

$$\lim_{N\rightarrow\infty}M_i(X_i|\mathbf{anc}(X_i))=P_i(X_i|\mathbf{anc}(X_i)) \tag{11}$$

□

Note that when the pseudo-tree is a chain, that is when the AND/OR sample tree is equivalent to the OR sample tree, AO-SIR is same as conventional SIR. This result follows trivially from the analysis presented in Lemma 1 and Theorem 2.

Corollary 1. *When the pseudo-tree is a chain, the solution samples output by AO-SIR will have the same distribution as those output by conventional SIR.*

Theorem 2 shows that AO-SIR has the same properties of convergence in the limit as conventional SIR. However, as pointed out in [12, 15], when the number of samples is finite, the performance of SIR is highly dependent on the size of the initial set **A** in that the accuracy of SIR increases with the cardinality of **A**. Since, the AND/OR sample tree represents a larger initial set of virtual samples, we expect AO-SIR to be more accurate than conventional SIR. In summary,

Theorem 2. *Asymptotically, AO-SIR generates solution samples from a distribution that may have lower error than conventional SIR.*

Clearly, when the number of solution subtrees in AO-SIR is equal to the size of the initial set, AO-SIR will yield the same sampling distribution as SIR. Namely, after the required samples are drawn, we can provide a stronger condition for testing whether the distribution of AO-SIR will coincide with the distribution of SIR. In summary,

Theorem 3. *If the number of solution subtrees of AO-SIR is equal to the size of the initial set of conventional SIR, then the samples output by AO-SIR have the same distribution as those output by SIR.*

In the following theorem, we address the complexity of storing and resampling samples on a AND/OR tree.

Theorem 4. *Given N samples and n variables (with constant domain size), the time and space complexity of computing AND/OR sample probability tree is $O(nN)$ (same as conventional SIR). The time complexity of resampling M samples in AO-SIR is $O(nM)$ (the complexity of resampling of SIR is $O(Mlog(N))$).*

In the next section, we discuss how AO-SIR can be modified to count and lower bound the number of solutions of a constraint network.

4 Approximate Counting on AND/OR search spaces

From Lemma 1, we know that the value of each OR node n is an unbiased estimate of the number of solutions admitted by the subtree rooted at n. Therefore, it is easy to see that:

Proposition 1. *The value of the root node of the AND/OR sample tree or graph is an unbiased estimate of the number of solutions of the CSP.*

In our previous work[5], we derived an unbiased estimate based on importance sampling for computing the solution counts. This unbiased estimate is equivalent to the unbiased estimate derived on an AND/OR sample tree whose pseudo-tree is a chain (i.e. an OR space estimator). Namely,

Proposition 2. *If the pseudo-tree is a chain then the value of root node of the AND/OR sample tree is equal to the conventional importance sampling based unbiased estimator.*

The main virtue of using the AND/OR space estimator over the OR space estimator is that the former may have lower variance (and therefore likely to have better accuracy) than the latter as we described in the following theorem:

Theorem 5. *The variance of the AND/OR space estimator (used for solution counting) is less than or equal to the variance of the conventional OR space estimator of [5].*

Algorithm 3 $AO-LB(\mathscr{R}, Q^F, N, k, \alpha > 1)$

```
1: minCount ← ∞
2: for i = 1 to k do
3:    Execute Steps 1-8 of Algorithm 2.
4:    estimate = v(root − node)
5:    IF minCount > estimate THEN minCount = estimate
6: end for
7: Return minCount/α
```

Table 1. Results for Solution Sampling

Problem	#Var	#Cl	SampleSearch	SampleSearch-SIR	AO-SIR	SampleSat
			KL	KL	KL	KL
Pebbling						
grid-pbl-10	110	191	0.08	0.002	**0.00008**	0.110
grid-pbl-15	240	436	0.12	0.017	**0.00010**	0.127
grid-pbl-20	420	781	0.10	0.008	**0.00110**	0.153
grid-pbl-25	650	1226	0.13	0.027	**0.00600**	0.138
grid-pbl-30	930	1771	0.15	0.040	**0.00170**	0.154
Circuit						
2bitcomp_5	125	310	0.03	0.003	**0.00100**	0.033
2bitmax_6	252	766	0.11	0.006	**0.00100**	0.039
ssa7552-158	1363	3034	0.06	0.006	**0.00072**	0.130
ssa7552-159	1363	3032	0.06	**0.003**	0.00450	0.130
Logistics						
log-1	939	3785	0.08	0.011	**0.00800**	0.051
log-2	1337	24777	0.12	0.270	**0.11000**	0.203
log-3	1413	29487	0.20	0.128	**0.07000**	0.176
log-4	2303	20963	0.21	0.183	**0.01393**	0.290
log-5	2701	29534	0.22	0.270	**0.18900**	0.260
Coloring						
Flat-100	300	1117	0.08	**0.001**	0.00190	0.020
Flat-200	600	2237	0.11	0.016	**0.00800**	0.030

Proof. See [6] for a proof.

Following [8, 5], we can easily combine the unbiased AND/OR space estimator with the Markov inequality to obtain a lower bound on the solution count. For completeness sake, we provide a pseudo-code (see Algorithm 3) for this scheme called AO-LB which yields a probabilistic lower bound on solution counts with a given confidence $1-\alpha^k$. AO-LB first generates an estimate of the solution counts by performing computations on an AND/OR tree (Steps 1-7 of Algorithm 2). It then returns the minimum estimate divided by α (minCount in Algorithm 3) over the k iterations. We can show that:

Theorem 6. *[5][8] With probability of at least $1-1/\alpha^k$, AO-LB computes a lower bound on the number of solutions of the constraint network $\mathscr{R}$.*

5 Experimental Evaluation

The main focus of our empirical study is to evaluate the performance of our new schemes of AO-SIR and AO-LB which operate on AND/OR space relative to the SampleSearch-SIR and SampleSearch-LB schemes which operate on OR space. For comparison, we also include the SampleSat [17]scheme for solution sampling and the SampleCount scheme [8] for solution counting [1]. Following previous work [17], we set the parameters SA and $cutoff$ of SampleSat to 50% and ∞ respectively. In all our experiments for lower bounding the solution counts using SampleCount, SampleSearch-LB and AO-LB, we set $k = 7$ and $\alpha = 2$ giving us a correctness confidence of 99% on our lower bound. For the SampleCount scheme we set the $\#samples$ parameter to 50. Also following previous work, we use minisat [16] as an underlying search procedure for SampleSearch while the proposal distribution Q is computed using the output of a generalized belief propagation algorithm called Iterative Join Graph Propagation.

We experimented primarily with satisfiability problems because of the relative easy availability of exact solution counting algorithms such as cachet[14] for these problems. Note that in case of solution sampling, we need an exact counting algorithm in order to evaluate the accuracy of the competing sampling schemes. Another use of cachet is that when it is terminated after a specific time bound, it achieves a lower bound on the number of solutions; thereby acting as a competing scheme for solution counting.

5.1 Results for solution sampling

We evaluate the quality of various algorithms by computing the average KL distance between the exact marginal distribution $P_e(x_i)$ and the approximate marginal distribution $P_a(x_i)$ of each variable ($KL = P_e(x_i)ln(P_e(x_i)/P_a(x_i))$). The exact marginal for each variable X_i can be computed as: $P_e(X_i = x_i) = |\mathbf{S}_{x_i}|/|\mathbf{S}|$ 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 Cachet [14]. After running various sampling algorithms, we get a set of solution samples ϕ from which we compute the approximate marginal distribution as: $P_a(X_i = x_i) = \phi(x_i)/|\phi|$ where $\phi(x_i)$ is the number of solutions in the set ϕ in which $X_i = x_i$.

We experimented with four sets of benchmarks (see Table 1): (a) the grid pebbling problems, (b) the logistics planning instances, (c) circuit instances and (d) flat graph coloring instances available from Satlib [10]. The first three benchmarks were used to evaluate Cachet [14]. Note that we used SAT problems whose solutions can be counted in relatively small amount of time because to compute the KL distance we have to count solutions to $n+1$ SAT problems for each formula having n variables.

Table 1 summarizes the results of running each algorithm for exactly 1 hr on various benchmarks. The second and the third column report the number of variables and clauses respectively of each benchmark. For each sampling algorithm, we report the average KL distance. Note that lower the KL distance the more accurate the sampling algorithm is. From Table 1, we can see that our new scheme of AO-SIR is more accurate than SampleSearch-SIR on most benchmarks. Also the SIR-type methods are more accurate than pure SampleSearch and SampleSat.

5.2 Results for Solution Counting

Table 2 summarizes our results for lower bounding solution counts on various benchmark problems using the following competing schemes (a) Cachet [14](b) SampleCount [8],

[1] In future, we hope to achieve a comparison with the recently proposed $XorSample$ scheme [9].

Table 2. Results on benchmarks. A Timeout of 12 hrs is used. The best results for lower-bounding (except for cachet) are highlighted in each row. A '-' in the exact count column indicates that the solution count is not known.

Instances	#Vars	Exact Count	Cachet		SampleCount		SampleSearch-LB		AO-LB	
		If known	Count	Time	LB	Time	LB	Time	LB	Time
Circuit										
2bitcomp6	252	2.10E+29	2.10E+29	10s	6.50E+28	9s	7.79E+28	5s	**8.00E+28**	5s
apex7-w5(sat-02)	1983	-	3.00E+39	12hrs	4.40E+86	28min	3.45E+86	35min	**1.94E+87**	35min
k-cnfs										
wff-3-3.5	150	1.40E+14	1.40E+14	6min	1.60E+13	4min	2.32E+13	5s	**4.79E+13**	5s
wff-3.1.5	100	1.80E+21	1.80E+21	3hrs	1.00E+20	4min	1.55E+20	20s	**9E+20**	20s
wff-4.5.0	100		8.00E+12	12hrs	8.00E+15	2min	2.30E+16	28s	**3.50E+16**	28s
4-coloring										
100-200	400	-	4.92E+24	12hrs	8.77E+32	1min	9.34E+36	28s	**1.94E+37**	29s
100-250	400	-	2.72E+23	12hrs	4.43E+26	30s	3.38E+29	29s	**1.37E+30**	29s
200-400	800	-	1.50E+33	12hrs	3.14E+68	2min	6.27E+72	2min	**2.02E+73**	2min
200-500	800	-	3.07E+35	12hrs	2.48E+57	3min	5.75E+63	2min	**5.33E+64**	2min
300-600	1200	-	4.48E+37	12hrs	4.40E+92	3min	5.68E+98	5min	**2.00E+100**	5min
300-750	1200	-	4.59E+35	12hrs	9.83E+84	5min	1.48E+89	5min	**2.72E+91**	5min
Langford										
Langford-20-2	1600	2.60E+12	1.90E+04	12hrs	5.80E+09	1hr	6.90E+11	15min	**6.90E+11**	15min
Langford-23-2	2116	3.70E+15	1.75E+05	12hrs	1.60E+11	1.5hr	1.50E+14	25min	**1.50E+14**	25min
Langford-24-2	2304	-	1.20E+05	12hrs	4.10E+13	1.5hr	9.80E+14	30min	**9.80E+14**	30min
Langford-27-2	2916	-	9.80E+03	12hrs	5.20E+14	2hr	1.50E+16	20min	**1.50E+16**	20min
Langford-28-2	3136	-	1.20E+04	12hrs	4.00E+14	2hr	3.40E+16	1hr	**3.40E+16**	1hr

(c) SampleSearch-LB [5] and (d) the AO-LB scheme. All algorithms except cachet are probabilistic in that the lower bounds may be incorrect with a probability $< 1\%$. Each algorithm was terminated after 12 hrs if it did not terminate earlier. We experimented with 4 sets of benchmarks (a) Circuit benchmarks available from satlib [10] (b) Flat Graph coloring instances generated using Joseph Culberson's flat coloring generator [1], (c) Random k-cnf benchmarks and (d) Langford instances available from [8].

We can see that on the circuit, k-cnf and graph coloring benchmarks, AO-LB yields far better lower bounds than other competing schemes as highlighted by bold in Table 1. On the langford instances (and latin square instances which are not shown due to lack of space) however, we notice that the lower bounds output by SampleSearch-LB and AO-LB are identical. The reason for this is Theorem 3 in that AND/OR estimates are equal to OR estimates because the number of solution subtrees in the AND/OR space is equal to the number of samples in the OR space. From AND/OR theory [2], we can show that this condition would not occur if the constraint graph of the problem has at least one or more small separators [2]. The langford and latin square instances however have constraint graphs which have large cliques and consequently large separators. Note however that as already proved, the AND/OR estimates are likely to be as good or better than OR estimates and the two have the same time complexity. Therefore AND/OR estimators should always be preferred over OR estimators.

[2] Separators are a set of nodes which separate the primal graph of a constraint network into two or more independent components

6 Conclusion

We introduced a new Sampling/Importance Resampling (SIR) algorithm which uses an AND/OR tree (or graph) to take advantage of problem decomposition. The main virtue of the AO-SIR scheme is that it is likely to have lower error than the conventional SIR algorithm which operates on an OR tree. We showed how the AO-SIR scheme can be combined with SampleSearch to generate solutions that converge in the limit to the uniform distribution. We also derived a new importance sampling based unbiased estimator and Markov inequality based lower bounding schemes for solution counts. Our experiments are preliminary but promising in that on most instances our new statistical schemes which operate in the AND/OR space have better performance than conventional statistical schemes that operate in the OR space.

References

1. Joseph Culberson. Flat graph coloring generator. http://www.cs.ualberta.ca/∼joe/Coloring/.
2. R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
3. Rina Dechter, Kalev Kask, Eyal Bin, and Roy Emek. Generating random solutions for constraint satisfaction problems. In *AAAI*, pages 15–21, 2002.
4. Vibhav Gogate and Rina Dechter. A new algorithm for sampling csp solutions uniformly at random. *CP*, 2006.
5. Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. In *AAAI*, pages 198–203, 2007.
6. Vibhav Gogate and Rina Dechter. And/or importance sampling. *Under Review*, 2008.
7. Vibhav Gogate and Rina Dechter. Studies in solution sampling. 2008.
8. Carla Gomes, Jeorg Hoffmann, Ashish Sabharwal, and Bart Selman. From sampling to model counting. *IJCAI*, 2007.
9. Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Near-uniform sampling of combinatorial spaces using xor constraints. In *NIPS*. 2007.
10. Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. pages 283–292.
11. J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.
12. Donald B. Rubin. The calculation of posterior distributions by data augmentation. *Jornal of the American Statistical Association*, 82, 1987.
13. Reuven Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., New York, NY, USA, 1981.
14. Tian Sang, Paul Beame, and Henry A. Kautz. Heuristics for fast exact model counting. In *SAT*, pages 226–240, 2005.
15. A. F. M. Smith and A. E. Gelfand. Bayesian statistics without tears: A sampling–resampling perspective. 46(2):84–88, May 1992.
16. Niklas Sorensson and Niklas Een. Minisat v1.13-a sat solver with conflict-clause minimization. In *SAT*, 2005.
17. Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: Exploiting random walk strategies. In *AAAI*, 2004.