Scaling Up AND/OR Abstraction Sampling

Kalev Kask, **Bobak Pezeshki**, **Filjor Broka**, **Alexander Ihler** and **Rina Dechter**
University of California, Irvine, Irvine, CA 92697, USA
{kkask, pezeshkb, fbroka, ihler, dechter}@ics.uci.edu

Abstract

Abstraction Sampling (AS) is a recently introduced enhancement of Importance Sampling that exploits stratification by using a notion of abstractions: groupings of similar nodes into abstract states. It was previously shown that AS performs particularly well when sampling over an AND/OR search space; however, existing schemes were limited to "proper" abstractions in order to ensure unbiasedness, severely hindering scalability. In this paper, we introduce AOAS, a new Abstraction Sampling scheme on AND/OR search spaces that allow more flexible use of abstractions by circumventing the properness requirement. We analyze the properties of this new algorithm and, in an extensive empirical evaluation on five benchmarks, over 480 problems, and comparing against other state of the art algorithms, illustrate AOAS's properties and show that it provides a far more powerful and competitive Abstraction Sampling framework.

1 Introduction

An important task in probabilistic graphical model inference is estimating the partition function (Z). Monte-Carlo methods have traditionally been used for generating Z estimates, with Importance Sampling (IS) being a primary framework [Rubinstein and Kroese, 2007; Liu *et al.*, 2015; Gogate and Dechter, 2011]. In IS, samples represent a single full configuration of the variables, with each given an *importance weight*. A newly introduced framework, Abstraction Sampling (AS) [Broka *et al.*, 2018], inspired by [Knuth, 1975; Chen, 1992], expands on IS by allowing each sample to represent *multiple* configurations rather than just one. These samples of multiple configurations are called *probes*. In Abstraction Sampling, probes are generated by grouping nodes together into *abstract states* based on an *abstraction function*, and then picking a representative node from each group to build a sampled subtree. As such, each probe is a subtree of the full search tree. Each probe is reweighted based on its chosen representative nodes and contributes to a Monte Carlo estimate of the partition function of the full tree.

In particular, Abstraction Sampling smoothly interpolates between standard IS (where all of a variable's states are abstracted into a single abstract state) and standard search (where only *provably equivalent* states are abstracted into a single abstract state in the search tree). As such, Abstraction Sampling can be considered a generalization of Stratified Sampling [Rubinstein and Kroese, 2007; Rizzo, 2007] combined with compact search [Dechter and Mateescu, 2007].

[Broka *et al.*, 2018] defined AS over OR and AND/OR search spaces and provided an analysis of complexity and conditions for unbiasedness. They introduced abstraction functions for which AS significantly improved performance over the baseline of Importance Sampling ***and was shown to be highly competitive against two state-of-the-art schemes: Weighted Mini-Bucket Importance Sampling*** [Liu *et al.*, 2015] and **IJGP-SampleSearch** [Gogate and Dechter, 2011].

However, a serious limitation of the existing AND/OR AS algorithm is that abstractions need to adhere to a property called *properness* to ensure unbiased estimation. This restriction (to be described) inhibits the algorithm's ability to group nodes into abstract states and thus to limit the size of its probes, ***severely hindering scalability***.

Contributions. 1. We introduce `AOAS`, a new Abstraction Sampling scheme over AND/OR search spaces. This new scheme lifts the properness restriction altogether, significantly enhancing the scalability of Abstraction Sampling for AND/OR spaces. 2. We provide new analysis and extensive empirical evaluation of all Abstraction Sampling schemes, and highlight how the new `AOAS` algorithm empowers formation of abstractions. 3. We compare against a new highly competitive scheme, Dynamic Importance Sampling (`DIS`), and show that `AOAS` is largely superior. We also strengthen the empirical results on instances lacking exact answers by placing the estimates within bounds produced by `DIS`.

2 Background and Definitions

A **graphical model**, such as a Bayesian or a Markov network [Pearl, 1988; Darwiche, 2009; Dechter, 2013], can be defined by a 3-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$, where $\mathbf{X} = \{X_i : i \in V\}$ is a set of variables indexed by a set V and $\mathbf{D} = \{D_i : i \in D\}$ is the set of finite domains of values for each X_i. Each function $\psi_\alpha \in \mathbf{\Phi}$ is defined over a subset of the variables called its scope, X_α, where $\alpha \subseteq V$ are the indices of variables in its scope and D_α denotes the Cartesian product of their domains, so that $\psi_\alpha : D_\alpha \rightarrow R^{\geq 0}$. The **primal graph** $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ of a graphical model associates each variable with a node

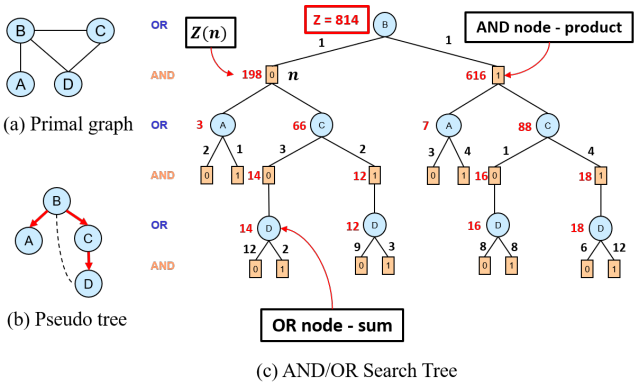


Figure 1: Example of an AND/OR search space. (a) A primal graph, (b) A possible pseudo tree for primal graph a), (c) An AND/OR search tree guided by pseudo tree b). To calculate the value of an OR node, we sum the values of its children multiplied by their arc weights, and for an AND node we multiply the values of it children. Partition function of the model is the resulting value of the root node.

($\mathbf{V} = \mathbf{X}$), while arcs $e \in \mathbf{E}$ connect nodes whose variables appear in the scope of the same local function. Graphical models can be used to represent a global function, often a probability distribution, defined by $Pr(X) \propto \prod_\alpha \psi_\alpha(X_\alpha)$. An important task is to compute the normalizing constant, also known as the partition function, $Z = \sum_X \prod_\alpha \psi_\alpha(X_\alpha)$.

Search Spaces of Graphical Models

A graphical model can be transformed into a weighted state space graph. In an OR search space, which is constructed layer-by-layer relative to a variable ordering, paths from the root to the leaves represent full configurations - or assignments to all variables - where each successive level corresponds to an assignment of the next variable in the ordering. A graphical model can also be transformed into a more compact AND/OR search space by capturing its conditional independencies, thus facilitating more effective algorithms [Dechter and Mateescu, 2007].

An AND/OR search space is defined relative to a *pseudo tree* of a primal graph. A **pseudo tree** $\mathcal{T} = (\mathbf{V}, \mathbf{E}')$ of a primal graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is a directed rooted tree that spans $\mathcal{G}$ such that every arc of $\mathcal{G}$ not in $\mathbf{E}'$ is a back-arc in $\mathcal{T}$ connecting a node to one of its ancestors (Figure 1(a),(b)). A variable is a **branching variable** if it has multiple children in $\mathcal{T}$.

Given a pseudo tree $\mathcal{T}$ of a primal graph $\mathcal{G}$, the *AND/OR search tree* $T_\mathcal{T}$ guided by $\mathcal{T}$ has alternating levels of OR nodes corresponding to variables, and AND nodes corresponding to an assignment from its domain with edge costs extracted from the original functions [Dechter and Mateescu, 2007]. Let n be an AND node in T_τ, also denoted n_X if X is the last variable of its partial configuration. Each arc into an AND node n has a cost $c(n)$ defined to be the product of all factors ψ_α in $\mathcal{M}$ that are instantiated at n but not before.

A **solution tree** is a subtree of $T_\mathcal{T}$ satisfying: (1) it contains the root of $T_\mathcal{T}$; (2) if an OR node is in the solution tree, exactly one of its AND child nodes is in the solution tree; (3) if an AND node is in the tree then all of its OR children are in the solution tree. Finally, its leaves are leaves of $T_\mathcal{T}$.

The cost of a solution tree is the product of its arc-costs and is equal to the cost of the corresponding full configuration of the graphical model [Dechter and Mateescu, 2007].

Value of A Node

The partition function, Z, of a graphical model $\mathcal{M}$ can be obtained by computing the value of the AND/OR search tree T, $V(T)$, by summing the costs of all its solution trees. This can be done recursively from leaves to root [Dechter and Mateescu, 2007] by associating a *value function* $V(n)$ with every node n in the AND/OR search tree. The **value of a node**, $V(n)$, is defined as the sum cost of all partial solution-trees rooted at n. In particular, $V(n_X)$ denotes the value of an AND node n_X of variable X. $V(Y_{n_X})$ denotes the value of an OR node Y that is the child of an AND node n_X, Y being the child of X in the pseudo tree. The value function can be computed by the following recursive expression:

$$V(n_X) = \prod_{Y \in ch_\mathcal{T}(X)} V(Y_{n_X}) \tag{1}$$

where

$$V(Y_{n_X}) = \sum_{n_Y \in ch_Y(n_X)} c(n_Y) \cdot V(n_Y) \tag{2}$$

and where $ch()$ denotes child variables either in the pseudo-tree or the search tree itself (depending on the context). Here, $ch_Y(n_X)$ are the child AND nodes of Y descended from AND node n_X. Note that $V(T) = V(root)$, where "$root$" is the root node of $T_\mathcal{T}$. It follows that $V(root) = Z_\mathcal{M}$, the partition function of the underlying model (see Figure 1c).

But what is the meaning of $V(n)$ for an arbitrary AND node n w.r.t the partition function of $\mathcal{M}$? Let $path(n)$ denote the configuration of variables on the path from the root to n in $T_\mathcal{T}$, let $Z_{|path(n)}$ be Z conditioned on the assignment to $path(n)$, and let $Out(path(n))$ be the set of OR nodes (corresponding to variables) emanating from $path(n)$, which are OUTside $path(n)$. Then Z, when restricting to the partial configuration of $path(n)$, is obtained by multiplying $g(n)$, the cost of $path(n)$, by $V(n)$ and by all the values from OR nodes branching out of $path(n)$ (the latter referred to as the value of the "ancestor branching").

It is easy to see that

Lemma 1.

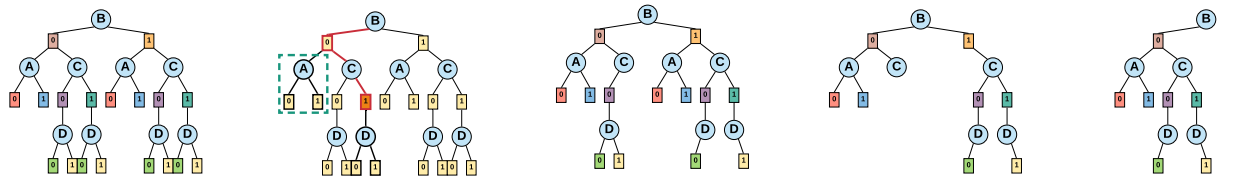
$$Z_{|path(n)} = g(n) \cdot V(n) \cdot \prod_{Y \in Out(n' \in path(n))} V(Y_{n'}) \tag{3}$$

where $V(Y_{n'})$ is as defined in EQ. 2.

Defining the product factor in Eq. 3, which we call *ancestral branching value* as

$$R(n) = \prod_{Y \in Out(n' \in path(n))} V(Y_{n'}) \tag{4}$$

we get that $Z_{|path(n)} = g(n) \cdot V(n) \cdot R(n)$, an expression that will be used in the algorithm for estimating the partition function at nodes of the search tree.



(a) Full AND/OR tree (b) ancestor branching (c) pAOAS probe (d) invalid probe (e) AOAS probe

Figure 2: An AND/OR search tree and possible probes generated by abstraction sampling. Nodes are abstracted based on having the same domain value (denoted by having the same color). (a) A full AND/OR tree, representing all 16 solutions, (b) Boxed in green is the ancestor branching subtree for the path $\rightarrow(B{=}0)\rightarrow(C{=}1)$, (c) A probe generated by proper abstractions, representing 8 solutions, (d) An invalid probe generated by improper abstractions with nodes abstracted across different sub-branches under the branching variable B (contains 4 *partial* configurations, and no full solution), (e) A valid probe generated by AOAS with improper abstractions, representing 4 solutions.

Example 1. *An example of expression (3) is given in Figure 2(b). Consider the path to node $n = (B{=}0, C{=}1)$ marked in red. There is only one branching variable ancestor (B) and, in this case, the only $Out(path(n))$ node is the A node emanating from under $(B = 0)$. Thus, $Z_{|path(B=0,C=1))} = g(B = 0, C = 1) \cdot V(B = 0, C = 1) \cdot V(B = 0, A)$.*

Stratified Importance Sampling.

Abstraction sampling builds on Importance Sampling and Stratified Sampling. *Importance Sampling* (IS) is a Monte Carlo scheme for approximating likelihood queries over graphical models. *Stratified Sampling* is a variance reduction technique for sampling a search space by first dividing it into disjoint strata. This can be used with importance sampling to further reduce variance. In *Stratified Importance Sampling*, we first divide the sample space into k strata of equal area under the distribution q, then choose re-weighted representatives from each strata. In order to maximize reduction in variance, the variance between strata should be maximized (see [Rizzo, 2007]).

3 AND/OR Abstraction Sampling

Abstraction Sampling (AS) algorithms [Broka *et al.*, 2018] emulate Stratified Importance Sampling with the modification that stratification and selection of representatives is done variable-by-variable along a search tree. Guided by an abstraction function a that groups nodes of a variable together, Abstraction Sampling generates a compact representative subtree of the full search tree $T_{\mathcal{T}}$ called a **probe**. An abstraction function, a, is formally defined as $a : T_{\mathcal{T}} \rightarrow I^{+}$, where I^{+} are integers, and it partitions the nodes in $T_{\mathcal{T}}$, layer by layer. From each partition (i.e. abstraction), a representative node is stochastically chosen based on a proposal p and is reweighted accordingly. AS can use any Importance Sampling proposal. For OR search trees we use p which is proportional to $w(n) \cdot g(n) \cdot h(n)$, where $h(n)$ is a heuristic that provides an upper bound on $V(n)$, $g(n)$ is the cost of $path(n)$, and $w(n)$ is the importance weight. The ORAS algorithm from [Broka *et al.*, 2018], which operates on an OR search tree, is unbiased for any Importance Sampling proposal. However, it was shown that using a heuristic-based proposal function can significantly impact convergence based on its accuracy. When $h(n) = h^{*} = V(n)$ the proposal is exact and only one sample is needed, as is the case for general Importance sampling.

pAOAS

Abstraction sampling algorithms for AND/OR search spaces operate by expanding and abstracting an AND/OR probe and can traverse the pseudo-tree $\mathcal{T}$ in a depth-first or breadth-first manner. However, performing these abstractions naively may leave partial (invalid) configurations within the resulting probe (ex. Figure 2(d)), potentially leading to biased estimates. To avoid this issue, [Broka *et al.*, 2018] forced abstractions to be *proper* with the scope of abstractions restricted to nodes descending from the same AND node of the most recent branching variable ancestor. A resulting such probe can be seen in Figure 2(c). More formally, consider a variable W whose most recent branching ancestor in $\mathcal{T}$ is U. When performing proper abstractions for W, only the AND nodes of W that are in the common subtree rooted under the same AND node of U can be grouped together [Broka *et al.*, 2018]. This ensure expansion to only valid probes. Unfortunately, properness limits compactness of the sampled AND/OR trees as it constrains the set of AND nodes that can be abstracted together. In particular, instead of bounding the size of the probe by $O(nm)$ AND nodes (as with ORAS), the number of AND nodes in a probe constructed via proper abstractions is bounded only by $O(n \cdot m^{b+1})$, where n is the number of variables, b bounds the number of branching variables along any path and m bounds the maximum number of abstract states per variable. This means that proper AND/OR abstraction schemes are not scalable whenever $b >> 0$, unless $m{=}1$, in effect limiting us to basic Importance Sampling. Various superficial schemes were used in [Broka *et al.*, 2018] to bound probe sizes in this case.

AOAS

Our solution to the above is, AOAS (Algorithm 1), a new AND/OR Abstraction Sampling algorithm. We will show the algorithm to be unbiased and effective far beyond the earlier versions. Given a pseudo tree $\mathcal{T}$ of a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ that is rooted at a dummy variable D, a heuristic function $h(n)$ providing upper bound on $V(n)$, and an abstraction function a, AOAS traverses the pseudo tree $\mathcal{T}$ variable-by-variable in a depth-first manner using the traversal to guide the generation of a partial search tree. The algo-

Algorithm 1: AOAS.

Input: Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$, a pseudo tree $\mathcal{T}$ for $\mathcal{M}$ rooted at a dummy singleton variable D, an abstraction function a, heuristic function h. For any node n, $g(n) =$ its path cost, $w(n) =$ its importance weight, and $\hat{V}(n) =$ its estimated value (initialized to $h(n)$).

Output: $\hat{Z}_{\mathcal{M}}$, an estimate of the partition function of $\mathcal{M}$

1 **Function** AOAS ($\mathcal{T}$, h, a)
2 **begin**
3 $PROBE \leftarrow n_D, g(n_D), w(n_D), r(n_D), \hat{V}(n_D) \leftarrow 1$
4 $STACK \leftarrow push(\text{empty stack}, D)$
5 **while** $STACK$ is not empty **do**
6 $X \leftarrow top(STACK)$
7 **if** X has unvisited children in $\mathcal{T}$ **then**
8 $Y \leftarrow$ the next unvisited child of X
9 **foreach** $n_X \in PROBE$ **do**
10 $PROBE \leftarrow PROBE$ expanded from n_X to Y
11 $F'_Y \leftarrow$ newly added AND nodes of $Y \in PROBE$
12 **foreach** $n_Y \in F'_Y$ **do**
13 $w(n_Y) \leftarrow w(n_X)$
14 $g(n_Y) \leftarrow g(n_X) \cdot c(n_Y)$
15 $r(n_Y) \leftarrow r(n_X) \cdot \prod_{\{S \neq Y \in ch_{\mathcal{T}}(X)\}} \hat{V}(S_{n_X})$
16 **end**
17 **end**
18 $A \leftarrow \{A_i \mid A_i = \{n_Y \in PROBE \mid a(n) = i\}\}$
19 **foreach** $A_i \in A$ **do**
20 **foreach** $n \in A_i$ **do**
21 $p(n) \leftarrow \frac{w(n) \cdot g(n) \cdot h(n) \cdot r(n)}{\sum_{m \in A_i} w(m) \cdot g(m) \cdot h(m) \cdot r(m)}$
22 **end**
23 $n_{Y_i} \propto_p A_i$; // randomly select
24 $w(n_{Y_i}) \leftarrow w(n_{Y_i}) / p(n_{Y_i})$
25 $\hat{V}(n_{Y_i}) \leftarrow 1$
26 $PROBE \leftarrow PROBE \setminus A_i \cup \{n_{Y_i}\}$
27 **end**
28 $push(STACK, Y)$
29 **else**
30 $pop(STACK)$, $W \leftarrow top(STACK)$
31 $PROBE \leftarrow PROBE$ s.t. all n_W without descendants are pruned
32 **foreach** n_W in $PROBE$ **do**
33 $\hat{V}(n_W) \leftarrow \hat{V}(n_W) \cdot \sum_{n_X \leftarrow child(n_W)} \hat{V}(n_X) \cdot c(n_X) \cdot \frac{w(n_X)}{w(n_W)}$
34 **end**
35 **if** $X = D$ **then** $\hat{Z}_{\mathcal{M}} = \hat{V}(D)$;
36 **end**
37 **end**
38 **return** $\hat{Z}_{\mathcal{M}}$
39 **end**

rithm uses a data structure referred to as $PROBE$ to store the generated subtree of the full search tree $T_{\mathcal{T}}$. As it progresses forward in the traversal, it expands and abstracts nodes within $PROBE$. Expansion is done by adding nodes of the newly visited variable across the tree in a breadth-first manner. During backtracking, the algorithm prunes nodes and updates values.

More specifically, in the forward step (lines 8-28), $PROBE$ is extended from all current leaf AND nodes of a variable X to the next variable Y (child of X in $\mathcal{T}$) in a breadth-first manner. The newly added AND nodes $n = n_Y$ are new leaves in $PROBE$. Each newly generated node n inherits the weight of its parent AND node n_X and has its path cost $g(n)$ computed. $r(n)$ is also computed (line 15), where it estimates $R(n)$ in the expression for $Z_{|path(n)}$ in EQ. (3, 4). Note that $r(n)$ is simply inherited from $r(n_X)$ if the parent variable X is not a branching variable. Otherwise, the $r(n_X)$ from its parent is multiplied by the estimated values contributed by its sibling branches, yielding an $r(n)$ which bounds the ancestor branching value for n.

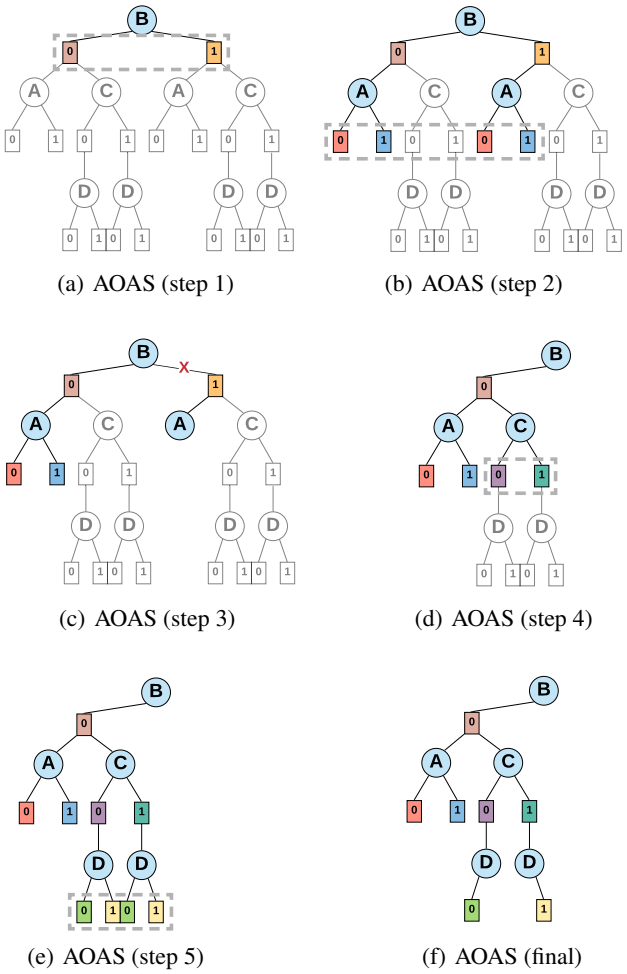
Once nodes are expanded, we perform abstractions (lines 18-27). Nodes are partitioned into abstract states according to an abstraction function a, and for each abstract state, proposals are generated from which a representative node is stochastically selected. The proposal function at node n is proportional to the quantity $w(n) \cdot g(n) \cdot h(n) \cdot r(n)$. The portion $g(n) \cdot h(n) \cdot r(n)$ estimates $Z_{|path(n)}$ (EQ. 3, 4), where $V(n)$ is approximated by $h(n)$ and $R(n)$ is estimated by $r(n)$. We multiply by $w(n)$ to account for all the mass n represents.

When backtracking to a variable W from variable X (lines 30-35), we prune any n_W that has no descendant n_X, thus preventing the creation of invalid probes (line 31). For the surviving nodes, we back up the values from their children from which we have backtracked (line 33). Once the algorithm backtracks to the dummy node, it will have backed up the estimated value of the full model, which is returned.

AOAS Trace

Figure 3 shows a step-by-step trace of AOAS. We follow a DFS traversal of the pseudo-tree $\mathcal{T}$ from Figure 1(b) and sample from the corresponding AND/OR search tree $T_{\mathcal{T}}$ (Figure 2(a)). The abstraction function we use groups nodes that have the same domain value, depicted by the color of the nodes. When performing abstractions, we box nodes that are being abstracted in gray. A red "X" marks a pruned subtree.

Starting with variable B (Figure 3(a)), each node belongs to a different abstraction and is therefore kept. Next, we expand to A and abstract across its nodes (Figure 3(b)). Not restricted to *proper* abstractions, we partition across *all* nodes of A, regardless of whether they fall under $B=0$ or $B=1$. We see two nodes in each abstract state (denoted by the red and blue coloring). Next we calculate their respective proposals (line 21). Note that the proposal of each node n relies on $r(n)$ (line 15), which captures the values of the nodes in its $Out(path(n))$, in this case nodes of C. Since the nodes of C have not been expanded yet, we use their heuristic values as an approximation of their values. We then stochastically choose a representative from each abstract state (line 23). Suppose that both red and blue representatives are stochas-



(a) AOAS (step 1) (b) AOAS (step 2)

(c) AOAS (step 3) (d) AOAS (step 4)

(e) AOAS (step 5) (f) AOAS (final)

Figure 3: A sample trace of AOAS based on the AND/OR tree in Figure 2(a). Nodes with the same domain values are abstracted (also indicated by node color). A red "X" indicates pruning. Transparent nodes indicate portions of the tree not pruned and yet to be explored.

tically chosen from under B=0 (Figure 3(c)). Since A has no descendant, we backtrack to B, updating its node values (line 33) and performing a pruning step (line 31). In pruning, we remove AND nodes of B that do not extend to AND nodes of A, and thus prune B=1 (denoted by the red "X" in Figure 3(c)), in order to ensure formation of proper AND/OR probes. Finally, we expand and abstract C and D (Figures 3(d)-3(f)). The $r(n)$ for D's nodes is inherited from the $r(n_C)$ of its respective n_C parent. We backtrack from D to the root updating values (no further pruning was necessary). The result is a valid probe (Figure 3(f)) containing four solutions: $(B=0, A=0, C=0, D=0)$, $(B=0, A=0, C=1, D=1)$, (B=0, A=1, C=0, D=0), and (B=0, A=1, C=1, D=1). We estimate the partition function by computing $\hat{V}(B)$.

Properties of AOAS

Algorithm AOAS does away with the properness requirement of pAOAS while ensuring unbiasedness. The proof is based on the fact that there is always a corresponding equivalent OR sampling algorithm, (e.g., using a DFS ordering of the pseudo-tree), such that there is a one-to-one correspondence between AND nodes in AND/OR sampling and equivalent OR sampling, and ensuring also that the proposal for the matching nodes is the same, implying that the two algorithms produce the same estimate. Since ORAS is known to be unbiased for any proposal, our claim follows. A key to this is that our choice of proposal, which includes $r(n)$, captures the estimated $\hat{Z}$ of the full configuration in the same way that would be done in the equivalent OR case. In other words we can show the following:

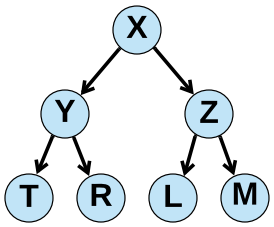
THEOREM **1** (simulation of AOAS with ORAS)**.** *Given a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$, pseudo-tree $\mathcal{T}_{\mathcal{AO}}$ of $\mathcal{M}$, an abstraction a^{AO}, and a proposal function p, then there exists a variable ordering d defining a chain-like pseudo-tree $\mathcal{T}_{\mathcal{OR}}$ of $\mathcal{M}$, and an abstraction function a^{OR} defined on $\mathcal{T}_{\mathcal{OR}}$ such that for every execution trace of* AOAS *on $\mathcal{T}_{\mathcal{AO}}$ using a^{AO} and proposal $p \cdot r$ and producing a corresponding $\hat{Z}_{\mathcal{M}}$ estimate, there is an execution trace of* ORAS *on $\mathcal{T}_{\mathcal{OR}}$ using a^{OR} and proposal p producing the same $\hat{Z}_{\mathcal{M}}$ estimate.*

THEOREM **2** (unbiasedness)**.** *Given a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$, algorithm* AOAS *provides an unbiased estimate for the partition function of $\mathcal{M}$.*

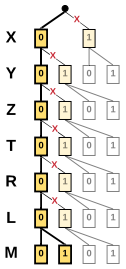
Complexity. It is easy to see that, like for ORAS, probe sizes for AOAS will not be larger than $O(m \cdot n)$, where n is the number of variables and m bounds the number of abstract states per variable. Note that theorem 1 does not imply that AOAS on $\mathcal{T}$ is the same as ORAS on the DFS ordering of $\mathcal{T}$ with the same a. In fact, the former can lead to more informed probes that express more solutions as we show next.

Contrasting Scalability. Figures 4(b), 4(c), and 4(d) show probes generated by ORAS, pAOAS, and AOAS, respectively, sampling over a search tree guided by the pseudo tree in Figure 4(a). ORAS ends up expanding 26 nodes, 8 of which are retained in the final probe constituting only two solutions: (X, Y, Z, T, R, L=0, M=0) and (X, Y, Z, T, R, L=0, M=1). pAOAS inevitably expands and keeps the entire AND/OR search tree of 42 nodes! However, since it samples a compact AND/OR search space, this nevertheless corresponds to a relatively many captured solutions (128 in this this case). AOAS only expands 20 nodes (fewer than both of the other schemes), and results in a probe size of 11 (comparable to ORAS). Further more, its final probe corresponds to 16 solutions, far more than that by ORAS. Thus, we see that AOAS is much more equipped than pAOAS to limit its probe size whilst able to take advantage of an AND/OR search space.

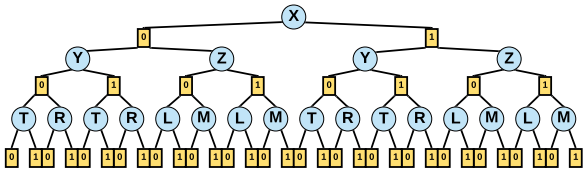
Cutset Sampling. An improvement to any IS sampling scheme can be made by sampling over a subset of the variables using *cutset decomposition* called *cutset sampling* [Dechter, 2013; Bidyuk and Dechter, 2007]. Using the weighted mini-bucket heuristic it is possible to identify when we reach a node for which the heuristic is exact (i.e. $h(n) = h^*(n) = V(n)$). When this occurs, there is no longer a need to sample below the node as the value of the subtree is known. Utilizing this method, the number of variables sampled is bounded by what is known as the i-*cutset*, which in turn bounds the size of the probes. Consequently, this leads



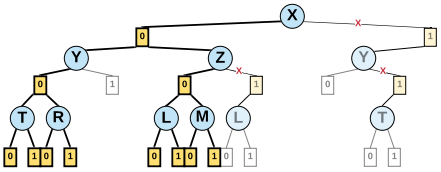
(a) Example pseudo tree



(b) ORAS; 8-nodes; 2-solutions



(c) pAOAS; 42-nodes, 128-solutions



(d) AOAS; 11-nodes, 16-solutions

Figure 4: Contrasting Scalability. AS probes sampled from a search space corresponding to the pseudo tree in (a). Nodes are abstracted based on having the same domain value. White nodes were discarded through the abstraction process. Light-yellow nodes are selected representatives that are later pruned. Pruning is depicted by a red "X". Dark-yellow nodes constitute the final probe. (b) a probe by ORAS; (c) a probe by pAOAS; (d) a probe by AOAS

to a greater number of samples and improves accuracy, as we reference in our empirical analysis.

4 Empirical Evaluation

Algorithms. Experiments were run on three classes of AS algorithms: AOAS, pAOAS, and ORAS (using a depth-first ordering of the pseudo tree). The AS algorithms were also tested against high performing state-of-the-art Dynamic Importance Sampling (DIS) [Lou *et al.*, 2019] using an "*equal-time*" policy. All algorithms were implemented in C++. All experiments were run for 1 hr on a 2.66 GHz processor with 8 GB of memory (24 GB for the Linkage-Type4 benchmark).

Abstraction Functions. We design abstractions to reduce variance within abstract states by using the notion of a variable's context. Formally, the **context** of a variable X identifies a subset $C(X)$ of its ancestor in a pseudo-tree $\mathcal{T}$ whose assignment uniquely determines the AND/OR subtree below it [Dechter and Mateescu, 2007]. Intuitively, the context of a variable is its ancestors that directly affect its value. Thus, abstractions based on a subset of the context aim to group nodes based on having similar values. AOAS and ORAS utilize relaxed context-based (**RelCB**) and randomized context-based (**RandCB**) abstractions as in [Broka *et al.*, 2018]. RelCB is parametrized by a level j, selecting the closest $j-1$ variables from a variable's context (ie. its *relaxed context*) plus itself. It abstracts nodes of the same domain value that also share the same assignment to the relaxed context. This yields k^j abstract states at each level, assuming domain size of k. RelCB-0 vacuously groups *all* nodes of a variable into a single abstract state and is referred to as a *Knuth Abstraction*. The randomized scheme, RandCB, is parameterized by a level d determining the number of abstractions per level, leading to a varying number of nodes in each abstract state. pAOAS utilizes a variant of RelCB that maintains *properness*. It is parameterized by levels j and k resulting in a j-level abstraction for nodes having no more than k branching variables in the path to the root and 0-level abstractions below it.

Problem	Size	Total	$\in$Bnds	AOAS$\geq$	AOAS$>$
DBN	small	66	62	57	47
	large	48	40	38	35
Grids	small	8	5	5	2
	large	19	7	7	6
Linkage[1]	large	82	82	82	82
Pedigree	small	24	24	24	19
Promedas	small	65	58	49	29
	large	173	165	141	113

Table 1: How often AOAS estimates: fall within DIS probabilistic bounds ($\in$*Bnds*), were comparable/better than DIS's ($AOAS \geq$), and were strictly better than DIS's ($AOAS >$)

Heuristics. To inform the sampling proposal, Weighted Mini-Bucket Elimination (WMBE) [Dechter and Rish, 2003; Liu and Ihler, 2011] is used as a heuristic. The i-bound parameter controls the strength of WMBE, where higher i-bounds generally lead to stronger heuristics and, thus, better proposals at the expense of higher computation and memory. We standardize our experiments by using i-bound 10.

Benchmarks. We perform high-throughput experiments on over 480 problems from five well known benchmarks: DBN, Grids, Linkage-Type4, Pedigree, and Promedas. For brevity, we show detailed aggregated statistics on only large problem instances, thus excluding Pedigree, whose problems were all small and results relatively uniform across all algorithms. Average statistics for the benchmarks can be found in Figure 5.

Performance Measure. To evaluate the performance of the various algorithms, we calculate error as: $error = log_{10}\hat{Z} - log_{10}Z^*$, where Z is the partition function, $log_{10}\hat{Z}$ is the log_{10} of the experimentally obtained Z estimate, and $log_{10}Z^*$ is the reference $log_{10}Z$ value. When the exact Z value is unknown, an empirical estimate based on an average over $100 \times$ 1hr of abstraction sampling is used as the reference. We verified that 98% of these estimates fell within the 95% probabilistic bounds determined by DIS.

i-Bound = 10 | DIS log(err) = -39.463 (3600 sec)

Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr. 0.5	error distr. 2	error distr. 10	#probes	#nodes/probe
DBN	(LARGE, n:1000, d:2, w:78, h:79/215)	AOAS	RelCB	0	48	-3.251	15	29	42	5.40E+05	434
				4	48	-4.045	5	20	41	5.52E+04	6352
				8	48	-5.180	5	12	40	4.08E+03	91614
			RandCB	16	**48**	**-2.358**	**8**	**31**	**45**	**5.20E+04**	**6249**
				256	**48**	**-2.606**	**10**	**30**	**45**	**1.38E+04**	**41131**
		pAOAS	RelCB (k=5)	0	48	-3.123	14	31	42	6.35E+05	434
				2	48	-3.591	10	23	42	2.00E+05	1690
				4	48	-3.712	7	22	43	5.80E+04	6352
			RandCB	16	**48**	**-2.328**	**12**	**31**	**44**	**5.41E+04**	**6276**
				256	**48**	**-2.591**	**11**	**29**	**46**	**1.41E+04**	**40782**
		ORAS -DFS	RelCB	0	48	-3.632	11	27	42	4.30E+05	434
				4	48	-4.377	4	19	42	3.42E+04	6586
				8	48	-5.284	2	11	40	2.66E+03	91881
			RandCB	16	**48**	**-2.704**	**7**	**28**	**44**	**3.71E+04**	**4621**
				256	48	-3.312	9	22	45	7.38E+03	30289

(a) DBN

i-Bound = 10 | DIS log(err) = -113.727 (3600 sec)

Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr. 0.5	error distr. 2	error distr. 10	#probes	#nodes/probe
Grids	(LARGE, n:3432, d:2, w:27, h:220/3431)	AOAS	RelCB	0	19	-167.080	0	0	4	1.92E+06	3326
				4	19	-67.780	0	4	7	2.34E+05	43449
				8	**19**	**-49.468**	**2**	**4**	**9**	**2.07E+04**	**407963**
			RandCB	16	19	-79.864	1	3	7	2.82E+05	32965
				256	19	-77.505	0	3	5	1.67E+05	66376
		pAOAS	RelCB (k=5)	0	19	-161.895	0	1	4	4.10E+06	3326
				2	19	-129.787	0	3	4	6.83E+03	2849697
				4	3	-2.972	0	1	3	4.00E+01	1.44E+08
			RandCB	16	3	-4.025	0	1	3	1.50E+02	41914557
				256	0	-	-	-	-	-	-
		ORAS -DFS	RelCB	0	19	-176.694	0	0	4	6.65E+05	6844
				4	19	-96.276	0	3	6	6.32E+04	78212
				8	19	-60.563	0	4	7	5.10E+03	886885
			RandCB	16	19	-96.497	0	0	4	1.02E+04	75095
				256	19	-95.435	0	1	5	1.47E+03	786178

(b) Grids

i-Bound = 10 | DIS Unable to Solve (3600 sec)

Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr. 0.5	error distr. 2	error distr. 10	#probes	#nodes/probe
Linkage-Type4	(LARGE, n:10822, d:1.8, w:24, h:581/5745)	AOAS	RelCB	0	7	-32.487	0	0	0	5.52E+05	829
				4	26	-14.174	1	1	10	1.92E+05	16472
				8	**41**	**-11.090**	**3**	**10**	**26**	**1.20E+04**	**463468**
			RandCB	16	26	-18.582	0	0	9	2.50E+05	10532
				256	27	-18.221	0	1	9	1.74E+05	18523
		pAOAS	RelCB (k=5)	0	8	-27.742	0	0	0	1.14E+07	759
				2	13	-27.201	0	0	1	6.96E+04	280176
				4	11	-25.427	0	0	0	8.90E+02	34177733
			RandCB	16	12	-24.204	0	0	1	4.24E+03	8820722
				256	14	-30.930	0	0	0	2.58E+03	24532116
		ORAS -DFS	RelCB	0	7	-29.820	0	0	0	1.42E+06	2520
				4	15	-20.802	0	0	2	5.14E+04	26863
				8	16	-18.971	0	0	5	2.44E+03	355404
			RandCB	16	13	-22.843	0	0	1	1.05E+04	23723
				256	12	-27.151	0	0	0	1.18E+02	288516

(c) Linkage-Type4

i-Bound = 10 | DIS log(err) = -66.595 (3600 sec)

Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr. 0.5	error distr. 2	error distr. 10	#probes	#nodes/probe
Promedas	(LARGE, n:1209, d:2, w:27, h:114/1193)	AOAS	RelCB	0	130	-4.781	10	46	111	2.10E+07	171
				4	145	-4.106	24	62	129	2.43E+06	2127
				8	139	-4.520	19	53	121	1.81E+05	23935
			RandCB	16	154	-4.142	26	71	140	1.89E+06	2495
				256	**173**	-3.531	29	78	161	4.83E+05	12845
		pAOAS	RelCB (k=5)	0	131	-4.664	18	46	112	3.31E+07	171
				2	158	-5.158	14	59	132	1.03E+06	11266
				4	145	-5.178	10	46	126	4.61E+04	925471
			RandCB	16	162	-3.306	35	89	150	5.58E+04	534634
				256	171	-2.050	34	107	168	2.36E+03	22671070
		ORAS -DFS	RelCB	0	128	-5.479	9	38	105	9.69E+06	361
				4	134	-5.261	10	36	111	1.78E+06	2566
				8	134	-5.073	14	37	118	1.96E+05	27440
			RandCB	16	144	-4.600	18	50	128	1.66E+05	4897
				256	**173**	**-2.821**	**15**	**70**	**171**	**2.66E+03**	**104091**

(d) Promedas

Figure 5: Aggregated statistics. Displayed are the number of problems solved (n*), average $log_{10}Z$ error (*log(err)*), count of problems solved within an error threshold (*error distr.*), average number of probes (*#probes*), and average size of a probe (*#nodes/probe*). Color bars visually show the magnitude of the values, and darker colors show greater values. Red n* cells indicate an algorithm's inability to solve relatively many problems. Lines in bold indicate the best performing algorithms. Each benchmark also displays the average number of nodes (n), domain size (d), tree width (w), and AND/OR and OR search tree height (h) of its problems.

4.1 Results

For extended results, please view the full paper[1].

Aggregation Tables. Data from experiments on the same benchmark using a specific AS scheme and abstraction level are combined in the aggregation tables (Figure 5). Importance Sampling corresponds to RelCB-0. At the top of each table (in red) we also report the average $log_{10}Z$ error for DIS. To help the reader identify AS algorithm configurations that performed particularly well, rows are bolded where (1) a relatively large number of problems are solved and (2) the error is within a factor of 0.2 of the minimum error within the set of algorithms solving the greatest number of problems.

Representative Plots. Figure 6 provides a representative plot from the Grids benchmark. The legend includes probe statistics and $log_{10}Z$ error values for each algorithm. A DIS plot is also overlaid onto each subplot for comparison. Although plots vary between benchmarks and problem instances, here we attempt to show a plot that captures the main trend of our data. In the majority of plots, we noticed AOAS converged towards the reference Z value faster, and had overall better performance than both ORAS-DFS and pAOAS. This was also true compared to DIS in many cases. We can also see the anytime nature of the algorithms as they continue to converge towards the reference Z value over time.

AOAS vs DIS Table. Table 1 compares AOAS RandCB-256 and DIS. For each benchmark (partitioned into small and large instances), we tally how often AOAS's Z estimates: (i) fall within DIS probabilistic bounds, (ii) were comparable or better than DIS's[2], or (iii) were strictly better than DIS's. Note that in order to compare Abstraction Sampling to DIS

[1] https://www.ics.uci.edu/~dechter/publications.html

[2] comparable means falling within ± 0.1 or ± 0.5 of the DIS $log_{10}Z$ value, for small and large problems respectively

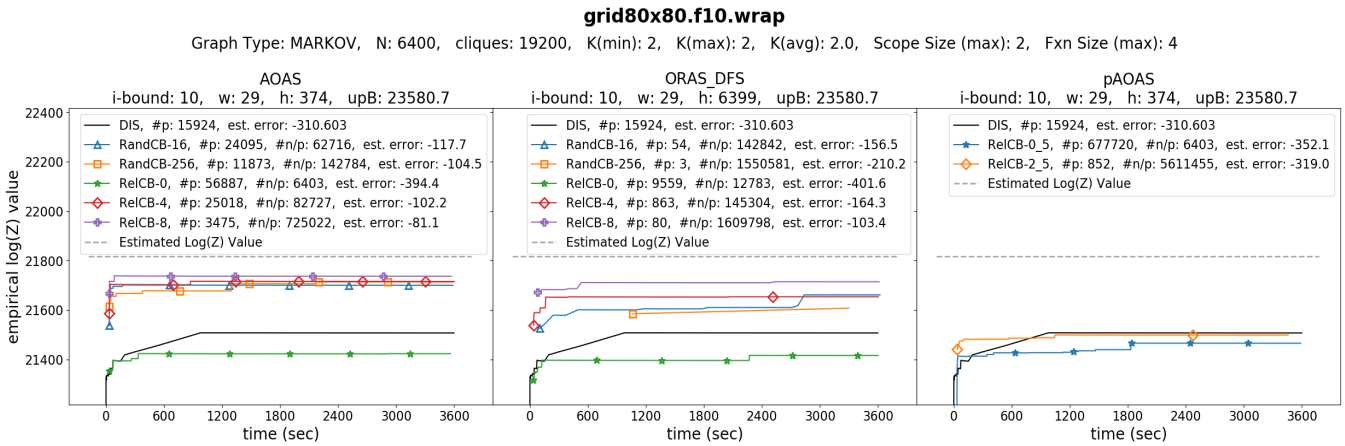


Figure 6: Plots of AS and DIS on a representative large Grid problem. The dashed line marks the reference $log_{10}Z$ value. The legend marks the algorithm and abstraction level, number of probes (*#p*), size of a probe (*#n/p*), and $log_{10}Z$ error (*est. error*).

fairly, we selected only a single set of Abstraction Sampling parameters (AOAS RandCB-256).

4.2 Analysis

AOAS Performance. The aggregation tables (Figure 5) show that for each benchmark (except Promedas) AOAS algorithms are among the top performers. Furthermore, AOAS stands out as the *only* best performing algorithm when solving both large Grids and Linkage-Type4 problems. Note that, for Linkage-Type4 (Figure 5(c)), AOAS with RelCB-8 is able to produce estimates for 41 problems, where the best of any other scheme, *ORAS-DFS*, is only able to produce estimates for 16 problems.

AOAS vs. pAOAS. A noticeable empirical difference between AOAS and pAOAS is their respective abilities to control their probe sizes with more refined abstractions. For example, within the Grids aggregation table (Figure 5(b)) both AOAS and pAOAS *RelCB-0* have average probe size of 3326, however, upon increasing the abstraction level of each individually, probe sizes for pAOAS explode. ***This is a crucial difference as being able to temper the growth of probe sizes is instrumental to the scalability of the algorithms with increasing abstraction levels***.

AOAS vs. ORAS-DFS. Despite the relationship stated in Theorem 1, AOAS regularly outperforms ORAS-DFS across the same abstraction level (see Figure 5). We believe that this is due to the smaller probe size of AOAS as compared with ORAS-DFS (for the same abstraction schema), which we believe is impacted significantly by the cutset effect [Mateescu and Dechter, 2005]. Namely, the number of i-cutset variables in the AND/OR space is smaller than that of the corresponding the OR space and, as noted earlier, since the heuristic is exact below the cutset, only the cutset variables are sampled by the abstraction sampling schemes.

Comparing with Non Abstraction Sampling Schemes. [Broka *et al.*, 2018] already showed that earlier versions of AS are highly competitive against Weighted Mini-Bucket Importance Sampling [Liu *et al.*, 2015] and IJGP-SampleSearch [Gogate and Dechter, 2011]. By superseding the performance of the earlier AS algorithms, we also further solidify superiority over these non-AS state-of-the-art schemes. In addition, we also test against a new competitive sampling scheme, *Dynamic Importance Sampling* (DIS) [Lou *et al.*, 2017; Lou *et al.*, 2019], which produces probabilistic bounds as well as estimates in an anytime manner. Through Table 1 we see that not only do AOAS estimates fall within the bounds produced by DIS in most cases, but also that the estimates are comparable or better than that of DIS. This is particularly true of hard problems. We also note that unlike AS algorithms (AOAS in particular), DIS was unable to generate estimates for Linkage-Type4 problems. Furthermore, the aggregated errors for AS are often smaller than that of DIS's (Table 5).

5 Summary and Conclusion

The paper presents a new algorithm for abstraction sampling over AND/OR search spaces that can accommodate any abstraction function. In particular it freed AND/OR abstraction sampling from its earlier handicapping restriction of properness and opens up the horizon for far more scalable and effective performance. We provided analysis and extensive empirical evaluation over 5 benchmarks with large and hard problems showing clearly that the new AOAS algorithm is overall superior to some of the most competitive state-of-the-art IS schemes and to previous AS schemes. We illustrated that, like previous AND/OR AS schemes, AOAS maintains the ability to exploit the decomposition expressed in AND/OR search spaces, yet it also has far better control of probe-size thus making it uniquely scalable. This gives AOAS the power to more smoothly interpolate between (stochastic) sampling and (systematic) search [Broka *et al.*, 2018]. With our findings, we can now turn our attention to advancing the scheme along its central components including the development of good abstraction functions beyond context-based abstractions, understanding the desired strengths of a heuristic function and how it should be balanced with the strength of abstractions, and finally, how we should fit an abstraction level to a problem instance.

References

[Bidyuk and Dechter, 2007] Bozhena Bidyuk and Rina Dechter. Cutset sampling for bayesian networks. *J. Artif. Intell. Res. (JAIR)*, 28:1–48, 2007.

[Broka *et al.*, 2018] Filjor Broka, Rina Dechter, Alexander T. Ihler, and Kalev Kask. Abstraction sampling in graphical models. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 632–641, 2018.

[Chen, 1992] P.-C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, 1992.

[Darwiche, 2009] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[Dechter and Mateescu, 2007] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

[Dechter and Rish, 2003] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003.

[Dechter, 2013] Rina Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[Gogate and Dechter, 2011] Vibhav Gogate and Rina Dechter. Samplesearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2):694–729, 2011.

[Knuth, 1975] D.E. Knuth. Estimating the efficiency of backtracking algorithms. *Math. Comput.*, 29:1121–136, 1975.

[Liu and Ihler, 2011] Qiang Liu and Alexander T. Ihler. Bounding the partition function using holder's inequality. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 849–856, 2011.

[Liu *et al.*, 2015] Qiang Liu, John W Fisher III, and Alexander T Ihler. Probabilistic variational bounds for graphical models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1432–1440, Montreal, Canada, 2015. Curran Associates, Inc.

[Lou *et al.*, 2017] Qi Lou, Rina Dechter, and Alexander T. Ihler. Dynamic importance sampling for anytime bounds of the partition function. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3199–3207, 2017.

[Lou *et al.*, 2019] Qi Lou, Rina Dechter, and Alexander Ihler. Interleave variational optimization with monte carlo sampling: A tale of two approximate inference paradigms. 2019.

[Mateescu and Dechter, 2005] Robert Mateescu and Rina Dechter. And/or cutset conditioning. In *International Joint Conference on Artificial Intelligence (Ijcai-2005)*, 2005.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[Rizzo, 2007] Maria L. Rizzo. *Statistical computing with R*. Chapman & Hall/CRC, 2007.

[Rubinstein and Kroese, 2007] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics)*. 2 edition, 2007.